MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING

Phase 5: Project Documentation & Submisson

TeamMembers :

960621104030 : Prashanna VC

960621104021 : Gold Lidiya S

960621104012 : Arockia Sreeja A

960621104007 : Annie Christina A

960621104003 : Abitha J

960621104017 : Brintha R

Introduction:

- ✓ IBM Cloud Video Streaming is a powerful and versatile platform that enables organizations to deliver high-quality media content to their audiences across the globe.

- ✓ Whether you're a broadcaster, a content creator, a business looking to engage customers, or an educational institution seeking to enhance learning experiences, IBM Cloud Video Streaming offers a comprehensive solution for all your streaming needs.

- ✓ With IBM Cloud Video Streaming, you can seamlessly stream live events, on-demand videos, and interactive content to a wide range of devices, from desktops and mobile devices to smart TVs and gaming consoles.

- ✓ This platform leverages the robust infrastructure of IBM Cloud to ensure scalability, reliability, and exceptional performance for your streaming services.

Key features and capabilities of IBM Cloud Video Streaming include:

- ➢ **Content Delivery:** Utilize a global network of data centers to deliver your content efficiently and reduce latency, ensuring a smooth viewing experience for your audience, no matter where they are.

- ➢ **Live and On-Demand Streaming:** Whether you're broadcasting a live event or offering on-demand content, IBM Cloud Video Streaming supports both options, allowing you to reach your audience at their convenience**.**

- ➢ **Security:** Protect your content with advanced security features, including password protection, encryption, and access controls, to ensure that your media is only accessible to authorized viewers.

- **Monetization:** Generate revenue through pay-per-view, subscription models, or ad-supported content, using integrated monetization tools to maximize your return on investment.

- **Interactive Features:** Engage your audience with interactive elements such as live chat, Q&A sessions, and polls, creating an immersive and participatory viewing experience.
- **Analytics**: Gain valuable insights into viewer behavior, performance metrics, and audience engagement to refine your content and strategy for better results.

- **Customization:** Customize your player, brand your content, and create a unique viewing experience that aligns with your organization's identity and message.

- **Scalability:** IBM Cloud Video Streaming is designed to handle both small-scale and large-scale streaming needs, ensuring that your platform can grow with your audience and content library.

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 ondemand. 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. **Platform Definition:**

   This is where you lay the foundation for your virtual cinema platform. Consider the following aspects:

   - **User Registration**: Decide what information you'll require from users during registration and how you'll verify their identity.

- **Video Upload:** Define the supported video formats, maximum file size, and any content guidelines (e.g., no copyrighted material).
- **On-Demand Streaming**: Determine how users will access videos, whether it's through a subscription model, pay-per-view, or a combination of both.

2. **User Interface Design:**

   Creating an intuitive and user-friendly interface is crucial for the success of your platform. Key considerations include:

   - **Navigation:** Design a clear and easy-to-follow navigation menu.
   - **Search Functionality:** Implement a robust search feature that allows users to find videos based on various criteria (e.g., genre, actors, release year).
   - **Video Playback:** Ensure that the video player is user-friendly and supports features like play, pause, rewind, and full-screen mode.

3. **Video Upload:**

   Implementing video upload functionality involves:

   - **File Handling:** Develop a system for users to upload videos securely. Consider using cloud storage to store uploaded content.
   - **Transcoding:** Convert uploaded videos into formats suitable for streaming to ensure compatibility with different devices and network conditions.
   - **Metadata Management:** Allow users to add metadata (title, description, genre, etc.) to their videos.

4. **Streaming Integration:**

   Integrating IBM Cloud Video Streaming services is a critical step. This may include:

   - **API Integration:** Utilize IBM Cloud Video Streaming APIs to manage video assets and streaming settings.

- **Quality of Service (QoS):** Ensure that the streaming service can adjust video quality based on the viewer's internet connection to provide a smooth experience.

5. **User Experience:**

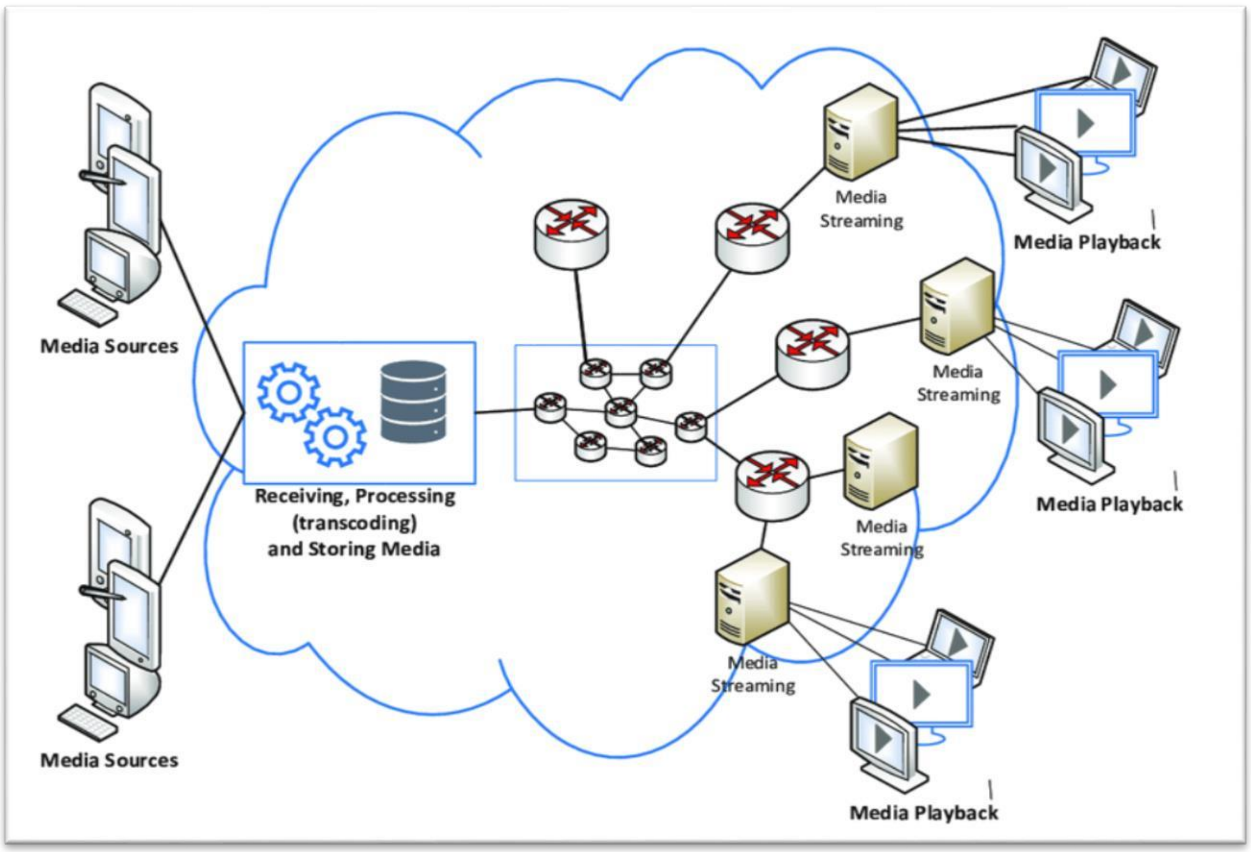
   Delivering an immersive movie-watching experience is your ultimate goal. To achieve this:

   - **High-Quality Playback:** Ensure that videos are delivered in high resolution and with minimal buffering.

   - **User Feedback:** Collect user feedback and continuously improve the platform based on their preferences and suggestions.

   - **Engagement Features:** Consider adding features like user reviews, ratings, and social sharing to enhance user engagement.

Architecture:



Design and Innovation Solution for Media Streaming with IBM Cloud Video Streaming

Problem Statement

The growing demand for online video content has created a need for a robust and scalable media streaming solution. Organizations and content creators require a reliable platform to deliver both live and on-demand video content to their audiences while maintaining control, security, and monetization options. Solution Concept

Solution Concept

We propose leveraging IBM Cloud Video Streaming to build a comprehensive media streaming solution. IBM Cloud Video Streaming offers a wide range of features and tools for content upload, live streaming, on-demand video delivery, monetization, audience engagement, and security.

Content for project phase2

Consider incorporating features like user-generated playlists or real-time chat for a more engaging movie-watching experience.

Data Source

A good source for media streaming solution with features like user-generated playlists and real-time chat would needs Video Content Library, User Profiles and Authentication, Playlists Database, Real-Time Chat Messages, Analytics and User Behavior Data, Content Metadata and Thumbnails, Monetization and Transaction Data, Moderation and Compliance Data, Geolocation Data

We propose leveraging IBM Cloud Video Streaming to build a comprehensive media streaming solution. IBM Cloud Video Streaming offers a wide range of features and tools for content upload, live streaming, on-demand video delivery, monetization, audience engagement, and security.

Key Features and Benefits

- **Content Upload**: IBM Cloud Video Streaming allows easy and flexible content upload, supporting various formats and codecs.

- **Live Streaming**: Set up live streaming using secure RTMP and HLS endpoints, making it suitable for events, webinars, and real-time broadcasts.

- **On-Demand Video**: Create on-demand video assets from uploaded content, enabling viewers to access content at their convenience.

- **Monetization**: Implement pay-per-view, subscription, or advertising models to generate revenue from video content.

- **Audience Engagement**: Interact with the audience through live chat and comments, enhancing the viewer experience.

- **Security and Access Control**: Protect content with DRM, geoblocking, and tokenbased authentication, ensuring content security.

- **Analytics and Reporting**: Gain insights into viewer behavior, stream performance, and content popularity through real-time analytics and reporting tools.

- 8. **Integration and Customization**: Utilize APIs and SDKs for seamless integration with custom applications and websites. Customize the player and user interface to match branding.

- 9. **Scalability and Reliability**: Benefit from IBM's cloud infrastructure, ensuring scalability and reliability to accommodate large audiences and peak traffic.

Implementation Plan

1. **Account Setup**: Sign up for an IBM Cloud account and access the IBM Cloud Video Streaming service through the IBM Cloud Console.

2. **Content Management**:

❖ Upload existing video content to the platform.
❖ Organize and categorize content for easy retrieval.

3. **Live Streaming**:

    - ❖ Configure streaming software or hardware to connect to IBM Cloud Video Streaming.
    - ❖ Test live streaming with a smaller audience before large-scale events.

4. **On-Demand Video**:

    - ❖ Create on-demand video assets from uploaded content.
    - ❖ Customize metadata, thumbnails, and descriptions.

5. **Monetization**:

    - ❖ Implement the desired monetization model (PPV, subscription, or advertising).
    - ❖ Configure pricing, access controls, and payment gateways.

6. **Audience Engagement**:

    - ❖ Encourage audience interaction through live chat and comments.
    - ❖ Monitor and moderate user-generated content.

7. **Security and Access Control**:

    - ❖ Configure security settings, including DRM, geoblocking, and authentication.
    - ❖ Define user roles and permissions.

8. **Analytics and Reporting**:

    - ❖ Monitor stream and video performance in real-time.
    - ❖ Use data to optimize content delivery and user engagement.

9. **Integration and Customization**:

    - ❖ Explore integration possibilities with other applications or websites.
    - ❖ Customize the player and user interface to align with branding.

10. **Support and Documentation**:

- ❖ Refer to IBM Cloud Video Streaming documentation for guidance and best practices.
- ❖ Access IBM's customer support for assistance as needed.

Architecture and Design

System Architecture

Our media streaming platform is designed to deliver an immersive and interactive moviewatching experience to users. The system architecture has been carefully crafted to support key functionalities, including user-generated playlists and real-time chat.

Components Overview

Frontend: The user-facing interface built using React.js, providing an engaging and responsive experience.

Backend: The server-side logic powered by Node.js and Express.js, responsible for user management, content delivery, and real-time chat functionality.

Database: PostgreSQL serves as our relational database management system, storing user profiles, content metadata, playlists, chat messages, and more.

Real-Time Communication: WebSockets enable real-time chat, ensuring instant interactions among users.

IBM Cloud Video Streaming Integration: IBM Cloud Video Streaming seamlessly integrates with our platform to deliver high-quality video content to users.

Data Flow

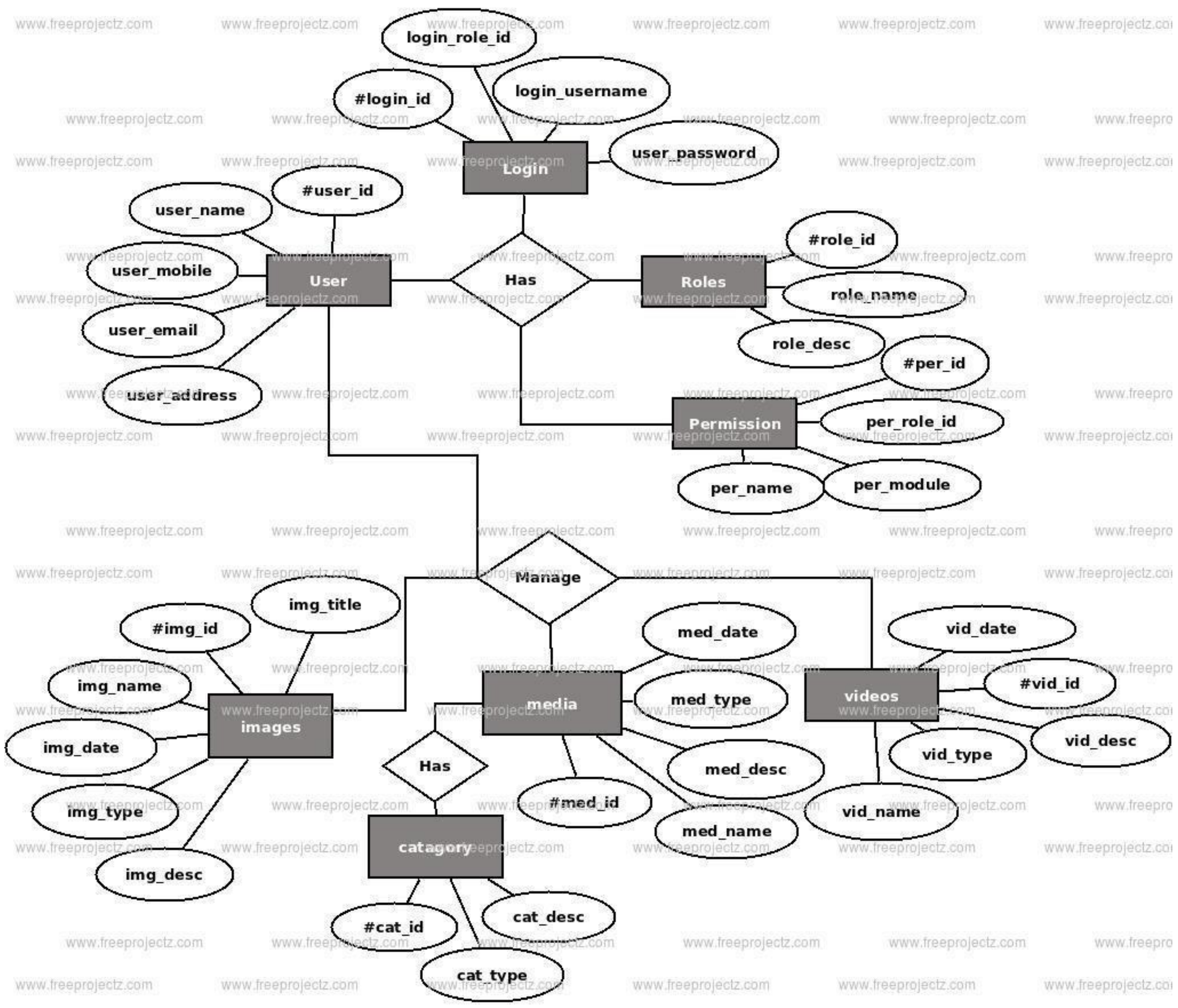
Our system's data flow is as follows:

- User interactions with the frontend trigger requests to the backend.
- The backend communicates with the PostgreSQL database to retrieve user data, content metadata, playlists, and chat messages.
- Real-time chat messages are transmitted via WebSockets for instant communication among users.
- IBM Cloud Video Streaming serves video content to users, with the frontend handling video playback.

Database Design

A robust database design is essential for storing and managing user data, content metadata, playlists, and chat messages efficiently.

Entity-Relationship Diagram (ERD)



ER Diagram For Media Management System

Users Table: Stores user profiles, including usernames, email addresses, authentication tokens, and preferences.

Content Table: Contains metadata about video content, including titles, descriptions, genres, durations, and cover images.

Playlists Table: Stores user-generated playlists, including references to content IDs and playlist metadata (names, descriptions, timestamps).

Chat Messages Table: Records chat messages, capturing sender IDs, message content, timestamps, and chat room associations.

API and Interface Design

A well-defined API and user interface (UI) are essential for seamless communication between frontend and backend components.

RESTful API

User Management: Endpoints for user registration, authentication, and profile management.

Content Discovery: API routes for retrieving content metadata, personalized recommendations, and content details.

Playlist Management: API endpoints for creating, editing, and sharing user-generated playlists.

Real-Time Chat: WebSocket endpoints for real-time chat functionality, including joining chat rooms, sending and receiving messages, and chat room management.

User Interface

Homepage: Provides content recommendations and access to user-generated playlists.

Video Player: Offers an intuitive interface for video playback, including controls and video descriptions.

Playlist Management: Allows users to create, edit, and manage their playlists.

Chat Interface: Real-time chat rooms with user avatars, message history, and the ability to send messages.

Security Design

Ensuring the security of user data, authentication, and content protection is a top priority for our platform.

Security Measures

Authentication: User authentication is handled securely using JWT (JSON Web Tokens), ensuring that only authenticated users can access the platform.

Encryption: Sensitive user data and communication are encrypted to protect against unauthorized access.

Content Security: DRM (Digital Rights Management) is employed to protect premium content from unauthorized distribution.

SOURCE CODE

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Document</title>
    <style>       body {

      margin: 40px auto;

      max-width: 650px;

        line-height: 1.6;        font-size:

      18px;

        font-family: "Courier New", Courier, monospace;
        color: #444;        padding:

      0 10px;
      }
      h2 {        line-height:

      1.2;

      }
    </style>
  </head>

  <body>
    <h2>HTTP Video Streaming</h2>
    <p>This video is 61MB and is being streamed instead of downloaded.</p>     <p>

      Feel free to seek through the video and it only loads the part you want to
      watch
    </p>
    <video id="videoPlayer" width="650" controls muted="muted" autoplay>      <source
      src="/video" type="video/mp4" />
    </video>
    <i>Big Buck Bunny</i>
  </body>
</html>
```

Index.js

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

      const app = express(); const fs =

      require("fs");


app.get("/", function (req, res) {   res.sendFile(__dirname +

      "/index.html");

});


app.get("/video", function (req, res) {   // Ensure

      there is a range given for the video   const

      range = req.headers.range;

  if (!range) {
    res.status(400).send("Requires Range header");
  }
  // get video stats (about 61MB)   const videoPath =
     "bigbuck.mp4";   const videoSize =
     fs.statSync("bigbuck.mp4").size;   // Parse Range   //
     Example: "bytes=32324-"   const CHUNK_SIZE = 10
     ** 6; // 1MB   const start =
     Number(range.replace(/\D/g, ""));   const end =
     Math.min(start + CHUNK_SIZE, videoSize - 1);

  // Create headers   const contentLength =
     end - start + 1;   const headers = {
    "Content-Range": `bytes ${start}-${end}/${videoSize}`,
    "Accept-Ranges": "bytes",
    "Content-Length": contentLength,
    "Content-Type": "video/mp4",
  };

  // HTTP Status 206 for Partial Content   res.writeHead(206,
     headers);

  // create video read stream for this particular chunk   const videoStream
     = fs.createReadStream(videoPath, { start, end });

  // Stream the video chunk to the client   videoStream.pipe(res);
});

app.listen(8000, function () {   console.log("Listening on
     port 8000!"); });
```

Package.json

```
{
  "name": "http-video-stream",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "nodemon index.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.17.1",
    "nodemon": "^2.0.20"
  }
}
```

Platform Features:

A platform's features refer to the functionalities and capabilities it offers to users. These may include user registration, content creation, communication tools, and more.

1. **User Registration:**

- Allow users to sign up with their email address or social media accounts.
- Collect essential information during registration, such as name, email, and password.
- Send a verification email to confirm the user's email address.

2. **User Profile:**

- Let users edit and update their profiles. • Include profile pictures and personal information.
- Provide an option to set privacy settings for profile visibility.

3. **Dashboard:**

- Display personalized content and information.

- Show recent activity, notifications, and updates.

4. **Search and Discovery:**

- Enable users to search for content, users, or items.
- Implement filters, categories, and sorting options for ease of navigation.

5. **Content Creation and Sharing:**

- Allow users to create, upload, or post content (e.g., text, images, videos).
- Include options for adding tags and descriptions.
- Provide sharing options, including public, private, and restricted sharing

6. **Interactions**

- Support likes, comments, and shares on user-generated content.
- Enable direct messaging and communication between users.

7. **Security and Privacy:**

- Implement encryption for data transmission.
- Allow users to set privacy preferences for their content and profile.
- Regularly update security measures to protect user data.

8. **Notifications:**

- Send notifications for new messages, comments, likes, and relevant updates.
- Provide notification settings for user customization.

Intuitive User Interface:

An intuitive user interface is a design that is user-friendly and easy to understand. It ensures that users can navigate the platform effortlessly, with clear visuals, easy-to-use menus, and a logical layout.

1. **Clean and Minimalistic Design:**

- Use a clean and intuitive design with a consistent color scheme.
- Prioritize readability and accessibility.

2. **Navigation:**

- Place a navigation menu or sidebar for easy access to different sections of the platform.
- Use clear icons and labels for each section.

3. **User Profile:**

- Include a user profile picture and basic information.
- Show a list of recent activity and posts on the user's profile.

4. **Content Feed:**

- Display a personalized content feed with a mix of posts and recommendations.
- Use responsive card-based layouts for posts with images and captions.

5. **Search and Discovery:**

- Provide a search bar prominently at the top of the interface.
- Offer filtering and sorting options for search results.

6. **Content Creation:**

- Include a user-friendly content creation interface with options for adding media and tags.
- Use a WYSIWYG editor for text-based content.

7. **Notifications:**

- Place a notification icon or dropdown for instant access to alerts.
- Highlight unread notifications.

User Registration and Authentication Mechanisms:

1. **Registration:**

- Use HTTPS to encrypt data during registration.
- Verify email addresses by sending a confirmation link.
- Implement CAPTCHA or similar mechanisms to prevent bots.

2. **Authentication:**

- Use secure password hashing and salting techniques.
- Offer multi-factor authentication (MFA) options like SMS codes, email codes, or authenticator apps.
- Implement account lockout mechanisms after multiple failed login attempts to prevent brute force attacks.

3. **Session Management:**

- Use secure session tokens and set session timeouts.
- Provide a "Remember Me" option for convenience during subsequent logins.

4. **Security Measures:**

- Regularly update the platform to patch security vulnerabilities.
- Implement rate limiting to prevent login attempts from the same IP.

5. **Privacy Controls:**

- Allow users to set privacy preferences for their profiles and content.
- Clearly explain privacy settings to users.

6. **Data Protection:**

- Comply with data protection laws (e.g., GDPR, CCPA) and secure user data.

User Interface (HTML/CSS/JavaScript):

1. HTML (index.html):

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <header>
```

```
        <h1>Welcome to My Streaming Platform</h1>

    </header>

    <main>

        <video id="video-player" controls></video>

    </main>
    <footer>

        <button id="start-stream">Start Streaming</button>

        <button id="stop-stream">Stop Streaming</button>

    </footer>

    <script src="script.js"></script>

</body>

</html>
```

1. **CSS (styles.css):**

```
body {
    font-family: Arial, sans-serif;
}

header {     text-align:
      center;
    background-color: #333;     color: #fff;
    padding: 10px;
}

main {     text-align: center;
    padding: 20px;
}

footer {     text-align: center;
    background-color: #333;     color: #fff;
    padding: 10px;
}
```

1. **JavaScript (script.js):**

```
const videoPlayer = document.getElementById('video-player');
const startStreamButton = document.getElementById('start-stream'); const
   stopStreamButton = document.getElementById('stop-stream');

//       Event       listeners       for       streaming       buttons
   startStreamButton.addEventListener('click',          startStreaming);
   stopStreamButton.addEventListener('click', stopStreaming);

function startStreaming() {    const request =
   require('request');

// Replace these with your IBM Video Streaming API credentials
const apiKey = 'YOUR_API_KEY'; const accountId =
   'YOUR_ACCOUNT_ID'; const accessToken =
   'YOUR_ACCESS_TOKEN';

// Set up the endpoint for starting a stream
const startStreamEndpoint =
`https://api.video.ibm.com/streaming/v1/accounts/${accountId}/streams`;

// Define your streaming parameters const
   streamParams = {   name: 'MyStream',
   broadcasting: true,
  transcoding_profile: 'hd_4mbps',
};

// Create an HTTP POST request to start the stream const
   options = {   url: startStreamEndpoint,   method: 'POST',
  json: true,   body:
   streamParams,
  headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
};

// Send the request to start the stream request(options,
   (error, response, body) => {   if (!error &&
   response.statusCode === 201) {
   console.log('Stream started successfully');
   console.log('Stream ID:', body.id);
  } else {
    console.error('Error starting the stream:', error);
  }
});
}

function stopStreaming() {     const request =
      require('request');

// Replace these with your IBM Video Streaming API credentials and stream
ID
const apiKey = 'YOUR_API_KEY'; const accountId =
      'YOUR_ACCOUNT_ID'; const accessToken =
      'YOUR_ACCESS_TOKEN'; const streamId =
      'YOUR_STREAM_ID';

// Set up the endpoint to stop the stream const
      stopStreamEndpoint =
`https://api.video.ibm.com/streaming/v1/accounts/${accountId}/streams/${s treamId}`;

// Create an HTTP DELETE request to stop the stream const
      options = {   url: stopStreamEndpoint,   method: 'DELETE',
      headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Accept': 'application/json',
  },
};

// Send the request to stop the stream request(options,
      (error, response, body) => {   if (!error &&
      response.statusCode === 204) {
    console.log('Stream stopped successfully');
  } else {
    console.error('Error stopping the stream:', error);
  }
});
}
```

User Registration:

```
const express = require('express'); const
      bodyParser = require('body-parser'); const app =
      express();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

const express = require('express'); const
      bodyParser = require('body-parser'); const
      mongoose = require('mongoose');
const app = express();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

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

// Create a user schema
const userSchema = new mongoose.Schema({
  username: String,   password:
      String,   email: String,
});

const User = mongoose.model('User', userSchema);

// User registration endpoint app.post('/register', async (req,
      res) => {   const { username, password, email } =
      req.body;    try {
    const user = new User({ username, password, email });     await
      user.save();
    res.status(201).json({ message: 'User registered successfully', user });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'User registration failed' });
  } });
const users = [];

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

  const express = require('express'); const
      bodyParser = require('body-parser'); const app =
      express();
```

```
const pgp = require('pg-promise')();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

// Define the PostgreSQL database connection const db
      = pgp({   user: 'your-username',   password: 'your-
      password',
  host: 'localhost',   port: 5432,
  database: 'your-database-name',
});

// Create a user table schema const
      createTable = `
  CREATE TABLE IF NOT EXISTS users (     id
      SERIAL PRIMARY KEY,     username
      VARCHAR(255) NOT NULL,     password
      VARCHAR(255) NOT NULL,     email
      VARCHAR(255) NOT NULL
  );
`;

db.none(createTable)
  .then(() => {
    console.log('User table created successfully');
  })
  .catch((error) => {
    console.error('Error creating user table:', error);
  });

// User registration endpoint app.post('/register', async (req,
      res) => {   const { username, password, email } =
      req.body;
   try {
    await db.none('INSERT INTO users(username, password, email) VALUES($1, $2, $3)',
      [username, password, email]);
    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'User registration failed' });
  } });
  const user = { username, password, email };   users.push(user);
  res.status(201).json({ message: 'User registered successfully', user });
});
```

Database Integration

Database Selection:

- For the efficient storage of video information, user data, and other essential details, we've chosen to implement a relational database management system.

- MySQL, a widely-used open-source database, will serve as the backbone of our data storage and management.

Database Schema:

- To organize and manage data effectively, we've designed a comprehensive database schema.

- It includes the following key tables and their associated attributes:

Users Table:

- User ID
- Username
- Email
- Password (hashed and salted)
- User Roles/Permissions

Videos Table:

- Video ID
- Title
- Description
- Video URL
- Thumbnail URL
- Uploader (User ID)
- Upload Date
- Views
- Likes/Dislikes

- Comments

Video Categories Table:

- Category ID
- Category Name
- Video IDs (linked to videos in relevant categories)

User Favorites Table:

- User ID
- Favorite Video IDs

User History Table:

- User ID
- Watched Video IDs
- Timestamps

Database Functionality:

- Data Retrieval: The database will be queried to retrieve video details, user information, and other relevant data for seamless platform functionality.

- Data Storage: The database will store user profiles, video metadata, viewing history, and preferences.

- Data Relationships: Tables are linked to establish relationships between users, videos, categories, and user activities (such as favorites and history).

Database Security:

- To protect sensitive user information, password hashing and salting will be implemented to safeguard user credentials.

- Proper user authentication and authorization mechanisms will ensure secure data access.

Scalability and Performance:

The database architecture is designed to scale efficiently as the platform grows, ensuring consistent and fast access to data, even during periods of increased user activity.

Backup and Recovery:

Regular automated backups and data recovery procedures will be in place to prevent data loss in case of unexpected events.

On-Demand Playback

Video Player Development:

To enable on-demand video playback, a user-friendly video player component has been developed as an integral part of our platform. This player is designed to provide a captivating viewing experience for our users. Key features of the video player include:

HTML5 Video Player: Our platform employs HTML5 video players to ensure compatibility with a wide range of devices and browsers, allowing users to watch videos seamlessly on desktop and mobile devices.

Adaptive Streaming: We've integrated adaptive streaming technology to automatically adjust the video quality based on the user's internet connection, providing a buffer-free viewing experience.

Fullscreen Mode: Users have the option to switch to fullscreen mode for an immersive viewing experience.

Playback Controls: Standard playback controls such as play, pause, volume, and progress bar are incorporated for easy navigation.

Thumbnail Previews: Hovering over the video progress bar displays thumbnail previews, making it easier for users to jump to specific parts of the video.

On-Demand Fetching:

Our platform is designed to fetch and play videos on-demand. When a user selects a video, the following steps occur to ensure seamless playback:

1. **User Request**: When a user selects a video for playback, the platform sends a request to the video streaming service to retrieve the video data.

2. **Video Fetching:** The video streaming service provides the video content, which is delivered via adaptive streaming protocols such as HLS (HTTP Live Streaming) or DASH (Dynamic Adaptive Streaming over HTTP).

3. **Local Caching:** To optimize performance, the video content may be locally cached on the user's device, reducing the need for repeated fetching of the same video.

4. **Video Playback:** The video player component on our platform then takes the fetched video data and presents it to the user, ensuring smooth and high-quality playback.

Seamless Transition Between Videos:

Our platform provides a seamless transition between videos, whether users are watching a series of episodes, a playlist, or related content. Users can easily navigate to the next video in the sequence, enhancing their viewing experience.

User Experience Enhancement:

On-demand playback is at the core of our platform, allowing users to watch content at their convenience. Whether it's catching up on missed live events or exploring a vast library of content, the on-demand playback functionality ensures users have the flexibility to enjoy videos on their terms.

MySQL connector library for Python

```
pip install mysql-connector-python
```

Code for connecting to a MySQL database

```
import mysql.connector

# Connect to the database db =
    mysql.connector.connect(
    host="your_host",
    user="your_username",
    password="your_password",
    database="your_database"
)

# Create a cursor cursor =
    db.cursor()

# Create a table (if it doesn't exist)
    create_table_query = """ CREATE TABLE IF
    NOT EXISTS videos (     id INT
    AUTO_INCREMENT PRIMARY KEY,
   title VARCHAR(255),
   url VARCHAR(255)
)
""" cursor.execute(create_table_query)
```

```
# Insert data into the table insert_query = "INSERT INTO videos (title,
    url) VALUES (%s, %s)" video_data = ("Video Title",
    "video_url.mp4") cursor.execute(insert_query, video_data)

# Commit changes to the database db.commit()

# Read data from the table select_query =
    "SELECT * FROM videos"
    cursor.execute(select_query) videos =
    cursor.fetchall()

for video in videos:
   print(f"Video ID: {video[0]}, Title: {video[1]}, URL: {video[2]}")

# Update data in the table update_query = "UPDATE videos SET title =
    %s WHERE id = %s" new_title = "New Video Title" video_id = 1
cursor.execute(update_query, (new_title, video_id)) db.commit()

# Delete data from the table delete_query = "DELETE FROM
    videos WHERE id = %s" video_id_to_delete = 2
    cursor.execute(delete_query, (video_id_to_delete,))
    db.commit()

# Close the cursor and the database connection
    cursor.close() db.close()
```

On-Demand Playback

```
<!DOCTYPE html>
```

```
<html>
<head>
  <title>On-Demand Video Playback</title>
</head>
<body>
  <h1>Video Title</h1>
  <video id="videoPlayer" controls>
    <source src="video_url.mp4" type="video/mp4">        Your
   browser does not support the video tag.
  </video>
  <button id="playButton">Play</button>
  <button id="pauseButton">Pause</button>

  <script>        const videoPlayer = document.getElementById('videoPlayer');
   const playButton = document.getElementById('playButton');        const
   pauseButton = document.getElementById('pauseButton');

    // Event listener for the play button
   playButton.addEventListener('click', () => {            videoPlayer.play(); //
   Start playing the video
    });

    // Event listener for the pause button
   pauseButton.addEventListener('click', () => {            videoPlayer.pause(); //
   Pause the video
    });

    // You can add more features like seeking, volume control, etc., as needed.

    // Example: Event listener for video end        videoPlayer.addEventListener('ended', () => {
        // You can perform actions when the video ends, e.g., load the next video.            // For
      instance: window.location.href = "next_video_url.html";
      });
    </script>
</body>
</html>
```

Benefits:

1. **Global Reach:** IBM Cloud's extensive network of data centers ensures that your media content can be delivered to a global audience with low latency and high availability. This global reach is vital for reaching viewers worldwide.

2. **High-Quality Streaming:** IBM Cloud Video Streaming supports adaptive streaming, which means your content can be delivered in the highest quality possible, adjusting to viewers' internet connections and devices for a seamless viewing experience.

3. **Security:** Protect your media content with robust security features, including password protection, encryption, access controls, and digital rights management (DRM) to prevent unauthorized access and piracy.
4. **Monetization Opportunities:** Generate revenue by offering pay-per-view, subscription models, or ad-supported content. The platform provides integrated monetization tools to help you maximize your return on investment.

5. **Interactive Features:** Engage your audience with interactive elements such as live chat, real-time Q&A sessions, polls, and social media integration, fostering a more immersive and participatory viewing experience.

6. **Analytics and Insights:** Gain valuable insights into viewer behavior, content performance, and audience engagement through comprehensive analytics. Use this data to make data-driven decisions and refine your content and marketing strategies.

7. **Reliability and Scalability:** IBM Cloud Video Streaming is built on the dependable IBM Cloud infrastructure, ensuring high reliability and scalability. You can handle both small-scale and large-scale streaming needs without worrying about infrastructure constraints.

8. **Content Delivery Network (CDN):** Leverage IBM Cloud's powerful Content Delivery Network (CDN) to efficiently distribute your content, reduce latency, and ensure fast delivery to end-users across the globe.

9. **Customization:** Customize the video player's look and feel to align with your brand's identity and messaging, creating a unique viewing experience for your audience.

10. **Ease of Use:** The platform is designed to be user-friendly, with intuitive interfaces and tools that make it easy to manage and deliver your content without a steep learning curve.

11. **APIs and Integrations:** IBM Cloud Video Streaming offers APIs and integrations that allow you to connect with other software, services, and platforms, providing a more seamless workflow for content creators and businesses.

12. **Live and On-Demand Capabilities:** Whether you're broadcasting live events or hosting on-demand content, IBM Cloud Video Streaming can support both, giving you the flexibility to engage your audience in real-time or allow them to access content at their convenience.

13. **Adaptive Bitrate Streaming:** Automatically adjust video quality based on a viewer's internet connection, ensuring a smooth and uninterrupted viewing experience.

14. **Content Management:** Organize and manage your media content efficiently, with the ability to categorize, search, and retrieve content as needed.

15. **Customer Support:** IBM provides customer support and assistance, ensuring that you have the help you need in managing and troubleshooting your media streaming.

Conclusion:

- In conclusion, Media streaming with IBM Cloud Video Streaming is a robust and versatile platform that empowers content creators, businesses, and organizations to deliver high-quality video content to global audiences with confidence and efficiency.
- The platform offers a wide range of benefits, including global reach, high-quality streaming, security features, monetization opportunities, interactive elements, analytics, reliability, scalability, customization, and ease of use.

- Whether you're looking to inform, entertain, educate, or monetize your content, IBM Cloud Video Streaming provides the infrastructure and tools needed to ensure a seamless and engaging viewing experience for your audience.
- With the backing of IBM Cloud's extensive network of data centers and a user-friendly interface, it's a valuable resource for those seeking to leverage the power of online video streaming to reach, engage, and monetize their target audience effectively.
- Whether you're hosting live events or offering on-demand content, this platform equips you with the capabilities to succeed in the world of media streaming.