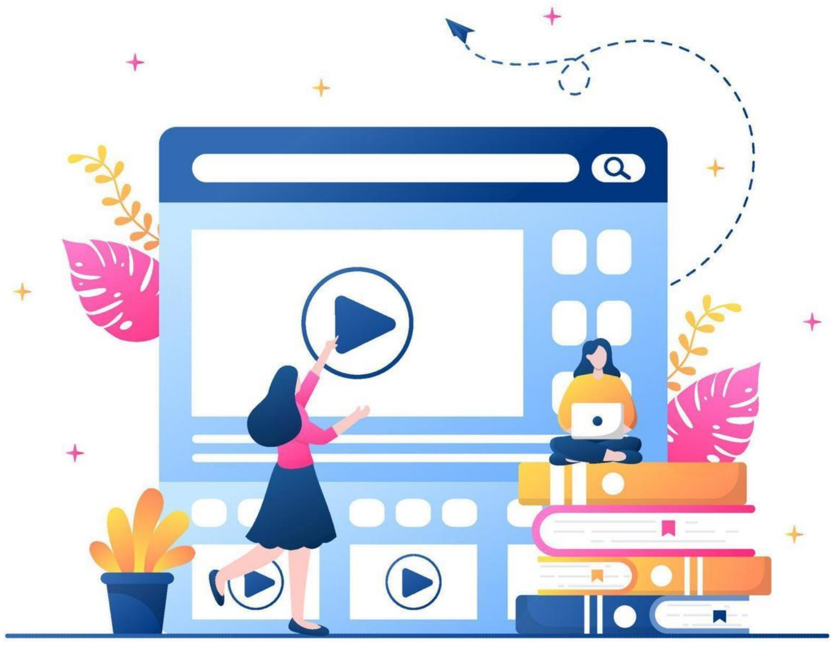
Media Streaming with Cloud Video Streaming



Submitted by

H. Aaliya Samira

P. Anitha

S. Deepa

S. Devisri

R. Dharanisri

Introduction

Project Title: Media Streaming Platform with Cloud Video Integration

Project Objective: Create a robust media streaming platform that allows users to upload, share, and watch videos, with high-quality playback through IBM Cloud Video Streaming.

Project Overview

Design Thinking Process: Explain the approach, design thinking principles, problem identification, ideation, and prototyping.

Target Audience: Define the target user base, their characteristics, and the expectations of the platform.

Features and Functionality

User Authentication: Describe how users create accounts and log in securely.

Video Upload: Explain the process for users to upload videos and how video data is stored and processed.

Video Streaming Integration: Detail the integration with IBM Cloud Video Streaming services for smooth and high-quality playback.

User Profiles: Describe how users can manage their profiles and content.

Content Discovery: Explain how users can discover new videos and explore content.

User Interaction: Detail how users can engage with content, including likes, comments, and sharing.

Content Moderation: Explain how content adherence to community guidelines is maintained.

Scalability and Performance: Describe how the platform ensures optimal performance under high traffic.

Testing and Quality Assurance: Explain how the platform is tested for bugs and performance.

User Interface Design

User Interface Elements: Provide an overview of the platform's UI components.

Wireframes and Mock-ups: Include visual representations of key screens.

User Experience (UX) Design: Describe how the UI design enhances the user experience.

Video Upload Process

Video Validation: Explain how the platform ensures that uploaded videos meet specifications.

Video Processing and Encoding: Describe the process of converting videos into suitable streaming formats.

Video Storage: Explain how and where uploaded videos are securely stored.

Code Example - Video Upload

```
``              JavaScript
                    // Example Node.js code for handling video uploads
                      const multer = require('multer');
                      const storage = multer.diskStorage({
                      destination: (req, file, cb) => {
                      cb(null, 'uploads/');
                 },
                     filename: (req, file, cb) => {
                    cb(null, file.fieldname + '-' + Date.now() +
path.extname(file.originalname));
                  }
                });
                    const upload = multer({ storage });
``
```

Streaming Integration

IBM Cloud Video Streaming: Detail the integration process with IBM Cloud Video Streaming services.

Streaming Server Setup: Explain the setup for video streaming on the platform.

Code Example - IBM Cloud Video Streaming Integration

```
        ``JavaScript
                // Example code for integrating with IBM Cloud Video Streaming services
                  Const ibmVideo = require('ibm-cloud-video-sdk');
                  const streamingClient = new ibmVideo.StreamingClient(
                 { apiKey:  'YOUR_API_KEY' });
      ``
```

Coding for the Video streaming

```
     <!DOCTYPE html>
    <html>
    <head>
    <title>Video Upload and Playback</title>
    </head>
    <body>
    <h1>Video Upload and Playback</h1>
   <form action="/upload" method="POST" enctype="multipart/form-data">
   <input type="file" name="video" accept=".mp4">
   <button type="submit">Upload Video</button>
   </form>
  <h2>Uploaded Videos</h2>
  <video controls>
  <source src="/uploads/your-uploaded-video.mp4" type="video/mp4">
  </video>
 </body>
</html>
const express = require('express');
const multer = require('multer');
const path = require('path');
const app = express();
const port = 3000;
// Set up storage for video uploads
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, 'uploads/');
  },
  filename: (req, file, cb) => {
    cb(null, file.fieldname + '-' + Date.now() + path.extname(file.originalname));
  }
});
const upload = multer({ storage });
// Serve uploaded videos
app.use('/uploads', express.static('uploads'));
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/index.html');
});
app.post('/upload', upload.single('video'), (req, res) => {
  // Handle video upload here
  res.send('Video uploaded successfully.');
});
app.listen(port, () => {
  console.log(`Server is running on http://localhost:${port}`);
});
```

Output:

Immersive Streaming Experience

Adaptive Streaming: Describe how the platform adjusts video quality based on user internet speed.

Content Discovery and Recommendation: Explain how users discover new content.

User Interaction and Engagement: Detail how users can interact with content.

Content Moderation: Describe how the platform maintains content quality.

Analytics and Monitoring: Explain how user behaviour and video performance are monitored.

Development Phases

Project Planning: Explain the initial project planning, including requirements gathering and project timeline.

Development Phases: Outline the key phases of development, including design, development, testing, and deployment.

Challenges and Solutions: Detail any challenges faced during development and how they were resolved.

Lessons Learned: Reflect on what worked well and what could be improved for future projects.

Conclusion

Project Outcomes: Summarize the achievements and outcomes of the project.

Future Improvements: Mention any future enhancements or features that could be added.

Acknowledgments: Thank individuals or organizations that contributed to the project.

Appendices

Code Samples: Include relevant code snippets or files.

Screenshots: Add screenshots of key platform features.

References: List any resources, tools, or frameworks used in the project.