MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING

AIM:

Media streaming with IBM cloud video streaming.

PROGRAM:

javascript

const express = require('express');

const app = express();

const fs = require('fs');

const path = require('path');

const videoPath = 'path\_to\_your\_video.mp4';

app.get('/stream', (req, res) => {

const stat = fs.statSync(videoPath);

const fileSize = stat.size;

const range = req.headers.range;

if (range) {

const parts = range.replace(/bytes=/, '').split('-');

const start = parseInt(parts[0], 10);

const end = parts[1] ? parseInt(parts[1], 10) : fileSize - 1;

const chunkSize = end - start + 1;

const file = fs.createReadStream(videoPath, { start, end });

const head = {

'Content-Range': `bytes ${start}-${end}/${fileSize}`,

'Accept-Ranges': 'bytes',

'Content-Length': chunkSize,

'Content-Type': 'video/mp4',

};

res.writeHead(206, head);

file.pipe(res);

} else {

const head = {

'Content-Length': fileSize,

'Content-Type': 'video/mp4',

};

res.writeHead(200, head);

fs.createReadStream(videoPath).pipe(res);

}

});

app.listen(3000, () => {

console.log('Server is running on port 3000');

});

PROGRAM:

from flask import Flask, render\_template, request

app = Flask(\_name\_)

# Route for the upload form

@app.route('/upload', methods=['GET', 'POST'])

def upload():

if request.method == 'POST':

# Check if a file was uploaded

if 'video' not in request.files:

return 'No file uploaded', 400

video\_file = request.files['video']

# Check if the file is empty

if video\_file.filename == '':

return 'No file selected', 400

# Perform additional validation here, such as file size, supported formats, etc.

# You can also generate a unique filename/key for the uploaded file

# Save the file to a cloud storage service

# Replace 'your\_storage\_bucket' with your actual cloud storage bucket name

video\_file.save('your\_storage\_bucket/' + video\_file.filename)

# Optionally, you can extract metadata from the uploaded video here

# Return a success message

return 'File uploaded successfully'

# Render the upload form template

return render\_template('upload.html')

if \_name\_ == '\_main\_':

    app.run()

In this example, we're using Flask to handle the web application and the /upload route for the upload functionality. When a user submits the upload form, the server checks if a file was uploaded and performs any necessary validation. The uploaded file is then saved to the cloud storage bucket, and a success message is returned.

Make sure to replace 'your\_storage\_bucket' with the actual name of your cloud storage bucket. Additionally, you'll need to create an HTML template file (upload.html) that contains the upload form.

Remember, this is a simplified example, and you'll need to adapt it to your specific requirements and chosen cloud storage service. You may also need to handle additional validation, error handling, and security measures based on your project needs.

Please note that this code assumes you have installed Flask and have a basic understanding of how to set up a Flask application. If you haven't done so, you can install Flask by running pip install flask in your Python environment.

I hope this helps you get started with implementing the upload functionality in your media streaming platform!