# Media Streaming with IBM Cloud Video Streaming

# Batch Member

# 621721243008: DAKSHNAMOORTHY S

**Project Title:** Virtual cinema platform using IBM cloud video streaming

**Phase 4:** Development Part 2

**Topic:** Start building the virtual cinema platform using IBM Cloud Video Streaming.

# Graphics-Top-Cloud-Video-Streaming-Platforms_BlogThumbnail-690x350-1

Phase 4 submission document

# Introduction:

Creating a virtual cinema platform using IBM Cloud Video Streaming involves several steps, from setting up the IBM Cloud services to configuring the streaming platform and creating a user interface for your virtual cinema.

1. Define the platform's features and design an intuitive user interface.

Firstly, we have to design a intuitive user interface with features such as register and login and so on. For that we have to create HTML page namely **Home.HTML**

<html>

<head>

<title></title>

<body>

<center>

<h1>Media Streaming with IBM Cloud Video Streaming</h1>

<img src="image3.png" alt=""width="30%"height="30%">

<br><br><a href="">Home</a>

<a href="login.html">Login</a>

<a href="register.html">Register</a>

</center>

</body>

</head>

</html>

2.To set up user registration and authentication mechanisms to ensure secure access to the platform.

To authenticate a user, We use flask framework in python.For that while the user try to authenticate, with information provided.It will authenticate if the information matches in the database else it will return the user is invalid.We have to create html page login namely **login.html**

<html>

<head></head>

<body>

{% if error %}

<p><strong>Error</strong>: {{error}}</p>

{% endif %}

<form action="/login",method ="post">

Enter name:

<p><input type="text" name="email"/></p>

<p><input type="password" name="pass"></p>

<p><input type="submit" name="LOGIN"></p>

</form>

</body>

</html>

We have created a login page for the authentication.In order to validate the user we must write a flask application to validate the user.The file must be saved with .py extension.

from flask import Flask, render\_template

app = Flask(\_\_name\_\_)

@app.route("/home")

@app.route("/index")

def home():

return render\_template("index.html")

@app.route("/login")

def login():

error = None;

if request.method == "POST":

if request.form["pass"] !="AAA":

error = "Invalid User"

else:

flash("successfully logged in")

return redirect(url\_for("home"))

return render\_template("log.html", error = error)

app.run()

**Registration in the virtual cinema platform**.

from flask import Flask, render\_template, request, redirect, url\_for, flash

from sqlalchemy import create\_engine, Column, Integer, String

from sqlalchemy.orm import sessionmaker

from sqlalchemy.ext.declarative import declarative\_base

app = Flask(\_\_name)

app.secret\_key = 'your\_secret\_key' # Replace with a secret key

# Connect to the IBM Db2 database

db2\_credentials = {

'hostname': 'your\_db2\_hostname',

'port': 'your\_db2\_port',

'database': 'your\_db2\_database\_name',

'user': 'your\_db2\_username',

'password': 'your\_db2\_password',

}

db\_uri = f"ibm\_db\_sa+pyodbc://{db2\_credentials['user']}:{db2\_credentials['password']}@{db2\_credentials['hostname']}:{db2\_credentials['port']}/{db2\_credentials['database']}"

engine = create\_engine(db\_uri)

Base = declarative\_base()

Session = sessionmaker(bind=engine)

class User(Base):

\_\_tablename\_\_ = 'users'

id = Column(Integer, primary\_key=True)

username = Column(String(80), unique=True, nullable=False)

email = Column(String(120), unique=True, nullable=False)

password = Column(String(80), nullable=False)

Base.metadata.create\_all(engine)

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

def register():

if request.method == 'POST':

username = request.form['username']

email = request.form['email']

password = request.form['password']

session = Session()

# Check if the user or email already exists

user\_exists = session.query(User).filter\_by(username=username).first()

email\_exists = session.query(User).filter\_by(email=email).first()

if user\_exists:

flash('Username already taken', 'danger')

elif email\_exists:

flash('Email already registered', 'danger')

else:

new\_user = User(username=username, email=email, password=password)

session.add(new\_user)

session.commit()

session.close()

flash('Account created successfully', 'success')

return redirect(url\_for('register'))

return render\_template('register.html')

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

app.run(debug=True)

This code sets up a simple Flask application with user registration functionality.You'll need to create a **templates** folder in your project directory and create an HTML template file named **register.html** for the registration form.

<!DOCTYPE html>

<html>

<head>

<title>User Registration</title>

</head>

<body>

<h2>User Registration</h2>

{% with messages = get\_flashed\_messages() %}

{% if messages %}

<ul class="flashes">

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

{% endwith }

<form method="POST">

<label for="username">Username:</label>

<input type="text" name="username" required><br><br>

<label for="email">Email:</label>

<input type="email" name="email" required><br><br>

<label for="password">Password:</label>

<input type="password" name="password" required><br><br>

<button type="submit">Register</button>

</form>

</body>

</html>

**Project Planning and Requirements Gathering:**

Define the scope of your project, including the types of content you plan to support (e.g., movies, user-generated videos).

from flask import Flask, render\_template, request, redirect, url\_for

app = Flask(\_\_name\_\_)

# Initialize an empty list to store project details and requirements.

projects = []

# Define a simple data structure to represent a project.

class Project:

def \_\_init\_\_(self, title, description, requirements):

self.title = title

self.description = description

self.requirements = requirements

@app.route('/')

def index():

return render\_template('index.html', projects=projects)

@app.route('/add\_project', methods=['GET', 'POST'])

def add\_project():

if request.method == 'POST':

title = request.form.get('title')

description = request.form.get('description')

requirements = request.form.get('requirements')

# Create a new project object and add it to the list.

project = Project(title, description, requirements)

projects.append(project)

return redirect(url\_for('index'))

return render\_template('add\_project.html')

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

app.run(debug=True)

Now, you need to create two HTML templates for rendering the web pages. Create two HTML files named index.html and add\_project.html

<!DOCTYPE html>

<html>

<head>

<title>Project List</title>

</head>

<body>

<h1>Project List</h1>

<ul>

{% for project in projects %}

<li>

<h3>{{ project.title }}</h3>

<p>{{ project.description }}</p>

<p>Requirements: {{ project.requirements }}</p>

</li>

{% endfor %}

</ul>

<a href="{{ url\_for('add\_project') }}">Add Project</a>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<title>Add Project</title>

</head>

<body>

<h1>Add Project</h1>

<form method="post">

<label for="title">Title:</label>

<input type="text" name="title" required><br>

<label for="description">Description:</label>

<textarea name="description" required></textarea><br>

<label for="requirements">Requirements:</label>

<textarea name="requirements" required></textarea><br>

<input type="submit" value="Submit">

</form>

<a href="{{ url\_for('index') }}">Back to Project List</a>

</body>

</html>

**Platform Architecture and Design:**

Designing the platform architecture in Flask involves structuring your Flask application for scalability, maintainability, and separation of concerns.

from flask import Flask, render\_template, request, redirect, url\_for

from flask\_sqlalchemy import SQLAlchemy

from flask\_migrate import Migrate

app = Flask(\_\_name\_\_)

# Configuration for the database

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///your\_database.db'

db = SQLAlchemy(app)

migrate = Migrate(app, db)

# Models (for SQLAlchemy)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

username = db.Column(db.String(80), unique=True, nullable=False)

email = db.Column(db.String(120), unique=True, nullable=False)

class Video(db.Model):

id = db.Column(db.Integer, primary\_key=True)

title = db.Column(db.String(100), nullable=False)

description = db.Column(db.Text, nullable=True)

user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)

user = db.relationship('User', backref=db.backref('videos', lazy=True))

# Views (Flask routes)

@app.route('/')

def index():

videos = Video.query.all()

return render\_template('index.html', videos=videos)

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

def upload():

if request.method == 'POST':

title = request.form['title']

description = request.form['description']

user\_id = 1 # You would typically associate this with the logged-in user.

video = Video(title=title, description=description, user\_id=user\_id)

db.session.add(video)

db.session.commit()

return redirect(url\_for('index'))

return render\_template('upload.html')

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

app.run(debug=True)

1. **Set Up IBM Cloud Video Streaming Services:**
   * Sign up for IBM Cloud and access the IBM Video Streaming services.
   * Create an IBM Cloud Video Streaming account.
   * Set up channels, configure encoding settings, and obtain API keys and credentials.

**Backend Development:**

Backend development in a Flask application involves creating the server-side logic to handle various functionalities, including handling HTTP requests, interacting with databases, and implementing business logic.

from flask import Flask, request, jsonify

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

# Configure the database

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///your\_database.db'

db = SQLAlchemy(app)

# Create a database model

class Todo(db.Model):

id = db.Column(db.Integer, primary\_key=True)

task = db.Column(db.String(100), nullable=False)

done = db.Column(db.Boolean, default=False)

# API route to get all tasks

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

def get\_tasks():

tasks = Todo.query.all()

task\_list = [{'id': task.id, 'task': task.task, 'done': task.done} for task in tasks]

return jsonify(task\_list)

# API route to create a new task

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

def create\_task():

data = request.get\_json()

new\_task = Todo(task=data['task'])

db.session.add(new\_task)

db.session.commit()

return jsonify({'message': 'Task created successfully'})

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

app.run(debug=True)

**Video Upload and Processing:**

Video upload and processing in Flask typically involve a combination of libraries and tools, including Flask-Uploads for handling file uploads and FFmpeg for video processing.

from flask import Flask, request, render\_template, redirect, url\_for

from flask\_uploads import UploadSet, configure\_uploads, VIDEO

import os

import subprocess

app = Flask(\_\_name\_\_)

# Configure file uploads

app.config['UPLOADED\_VIDEOS\_DEST'] = 'uploads/videos'

videos = UploadSet('videos', extensions=VIDEO)

configure\_uploads(app, videos)

# Route for video upload and processing

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

def upload\_video():

if request.method == 'POST' and 'video' in request.files:

video = request.files['video']

if video:

# Save the uploaded video

video\_path = os.path.join(app.config['UPLOADED\_VIDEOS\_DEST'], video.filename)

video.save(video\_path)

# Process the video (e.g., convert to MP4)

process\_video(video\_path)

return redirect(url\_for('index'))

return render\_template('upload.html')

# Function to process the uploaded video (using FFmpeg)

def process\_video(video\_path):

output\_path = os.path.splitext(video\_path)[0] + '.mp4'

subprocess.call(['ffmpeg', '-i', video\_path, output\_path])

# Route for the homepage

@app.route('/')

def index():

return render\_template('index.html')

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

app.run(debug=True)

<!DOCTYPE html>

<html>

<head>

<title>Video Upload</title>

</head>

<body>

<h1>Upload Video</h1>

<form method="POST" enctype="multipart/form-data">

<input type="file" name="video" accept="video/\*" required>

<button type="submit">Upload</button>

</form>

<a href="{{ url\_for('index') }}">Go to Homepage</a>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<title>Homepage</title>

</head>

<body>

<h1>Welcome to the Video Platform</h1>

<a href="{{ url\_for('upload\_video') }}">Upload Video</a>

</body>

</html>

**Conclusion:**

This is the basic program for user registration and the authentication.. Additionally, regularly test and review your authentication mechanisms for vulnerabilities and ensure that your platform complies with the latest security and privacy standards.