# Media Streaming with IBM Cloud Video Streaming

# Batch Member

# 621721243007 : DAKSHNA MOORTHY.S

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

**Phase 3:** Development Part 1

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

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

Phase 3 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>

**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.