PHASE – 3

MEDIA STREAMING

USING IBM CLOUD

SUBMITTED BY:

CHITTETI SURENDRA

au723921104011

surendrachitteti1112@gmail.com

PROJECT : MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING.

Development Part – 1 :

Building a virtual cinema platform using IBM Cloud Video Streaming.

1. Set Up an IBM Cloud Account:

If you haven't already, sign up for an IBM Cloud account and create a project.

2. Choose IBM Cloud Video Streaming:

IBM Cloud Video Streaming provides the core infrastructure for your platform.

3. Plan Your Platform:

Define the features and capabilities of your virtual cinema platform. Consider aspects like user registration, content management, video streaming, and user interaction.

4. Design the Database:

Design a database to store information about movies, users, purchases, and other relevant data.

5. Develop the Backend:

Use a server-side technology (e.g., Node.js, Python, Java) to build the backend of your platform. This includes user authentication, content management, payment processing, and API endpoints for your frontend.

6. Implement User Registration and Authentication:

Use IBM Cloud services or custom authentication to allow users to create accounts and log in securely.

7. Content Management:

Implement a content management system to upload and categorize movies.

8. Payment Integration:

Integrate a payment gateway (e.g., Stripe) for users to purchase access to movies or events.

9. Video Encoding and Storage:

Utilize IBM Cloud Video Streaming for video encoding, storage, and delivery.

10. Build a Frontend:

Develop a user-friendly web or mobile frontend using HTML, CSS, and JavaScript frameworks like React, Angular, or Vue.js.

11. Implement Video Player:

Integrate a video player into your frontend to allow users to stream movies or live events.

12. Implement Social Features:

Add features like chat, comments, and user interactions to create a sense of community.

13. Testing:

Thoroughly test your platform to ensure it works smoothly, including streaming quality and payment processing.

14. Security and Compliance:

Implement security measures and ensure compliance with relevant regulations, especially regarding user data and payments.

15. Scalability:

Prepare your platform for scalability to handle increased traffic and demand.

16. Launch and Marketing:

Launch your virtual cinema platform and market it to your target audience.

17. Monitoring and Maintenance:

Continuously monitor the platform's performance and security, and provide ongoing support and updates.

Designing a platform's features and user interface requires careful planning and consideration of

user needs. While I can't physically design interfaces, I can provide general guidelines and features to consider for an intuitive user experience.

Platform Features:

1. User Profiles: Allow users to create personalized profiles with relevant information and preferences.

2. Authentication and Security: Implement secure login methods and data encryption to protect user information.

3. Content Creation and Sharing: Enable users to create, upload, and share various types of content such as text, images, videos, and documents.

4. Social Interaction: Include features for liking, commenting, and sharing content. Implement a messaging system for private communication.

5. Search and Filters: Provide a robust search functionality with filters to help users find specific content or users easily.

6. Notifications: Keep users informed about new messages, comments, likes, or relevant updates through notifications.

7. Customization: Allow users to customize their experience, such as choosing themes, display options, and notification preferences.

8. Community Building: Support the creation of groups or communities where users with similar interests can connect and interact.

9. Gamification: Implement gamification elements like badges, points, or levels to encourage user engagement and contribution.

10. Analytics: Provide users with insights into their activities, such as views, likes, and followers, to understand their impact on the platform.

User Interface Design:

1. Simplicity: Keep the interface clean and intuitive, avoiding clutter and unnecessary elements. Use a consistent layout throughout the platform.

2. Navigation: Use clear and logical navigation menus and buttons. Include breadcrumbs and categories to help users understand their location within the platform.

3. Visual Hierarchy: Emphasize important elements like buttons and headings using contrasting colors, sizes, or styles. Guide users' attention to essential actions and information.

4. Responsive Design: Ensure the platform is accessible and functional across various devices and screen sizes, including smartphones, tablets, and desktops.

5. Feedback and Validation: Provide instant feedback for user actions, such as successful content uploads or form submissions. Display clear error messages if something goes wrong.

6. Loading and Progress Indicators: Use loading spinners or progress bars to indicate when content is being loaded, preventing users from getting frustrated.

7. Accessibility: Design the platform to be accessible to users with disabilities, following accessibility standards to accommodate various needs.

8. User Assistance: Offer tooltips, hints, or tutorials to guide new users through the platform's features, especially if there are complex functionalities.

9. Consistent Typography: Use readable fonts, maintain consistent text sizes, and ensure proper line spacing to enhance readability.

10. Testing and Iteration: Continuously test the user interface with real users, gather feedback, and iterate on the design to improve usability and user satisfaction.

USER REGISTRATION AND AUTHENTICATION MEECHANISMS TO ENSURE SECURE ACCESS TO THE PLATFORM:

PROJECT : Code Generation

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from flask_bcrypt import Bcrypt

app = Flask(__name)
app.config['SQLALCHEMY_DATABASE_URI'] =
'sqlite:///users.db'
db = SQLAlchemy(app)
bcrypt = Bcrypt()

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20),
unique=True, nullable=False)
```

```
    password = db.Column(db.String(60), nullable=False)


@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    hashed_password = bcrypt.generate_password_hash(data['password']).decode('utf-8')
    new_user = User(username=data['username'], password=hashed_password)

    db.session.add(new_user)
    db.session.commit()

    return jsonify({'message': 'User registered successfully'})
```

```
# Login route
@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    user = User.query.filter_by(username=data['username']).first()

    if user and bcrypt.check_password_hash(user.password, data['password']):
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'message': 'Login failed'})

if _name_ == '_main_':
    app.run(debug=True)
```