MEDIA STREAMING WITH IBM CLOUD VIDEO STREAMING

Phase 3 : Development Part 1

Team Members :

960621104030 : Prashanna VC

960621104021 : Gold Lidiya S

960621104012 : Arockia Sreeja A

960621104007 : Annie Christina A

960621104003 : Abitha J

960621104017 : Brintha R

Platform Features:

A platform's features refer to the functionalities and capabilities it offers to users. These may include user registration, content creation, communication tools, and more.

1. **User Registration:**

- Allow users to sign up with their email address or social media accounts.
- Collect essential information during registration, such as name, email, and password.
- Send a verification email to confirm the user's email address.

2. **User Profile:**

- Let users edit and update their profiles.
- Include profile pictures and personal information.
- Provide an option to set privacy settings for profile visibility.

3. **Dashboard:**

- Display personalized content and information.
- Show recent activity, notifications, and updates.

4. **Search and Discovery:**

- Enable users to search for content, users, or items.
- Implement filters, categories, and sorting options for ease of navigation.

5. Content Creation and Sharing:

- Allow users to create, upload, or post content (e.g., text, images, videos).
- Include options for adding tags and descriptions.
- Provide sharing options, including public, private, and restricted sharing

6. Interactions

- Support likes, comments, and shares on user-generated content.
- Enable direct messaging and communication between users.

7. Security and Privacy:

- Implement encryption for data transmission.
- Allow users to set privacy preferences for their content and profile.
- Regularly update security measures to protect user data.

8. Notifications:

- Send notifications for new messages, comments, likes, and relevant updates.
- Provide notification settings for user customization.

Intuitive User Interface:

An intuitive user interface is a design that is user-friendly and easy to understand. It ensures that users can navigate the platform effortlessly, with clear visuals, easy-to-use menus, and a logical layout.

1. **Clean and Minimalistic Design:**

- Use a clean and intuitive design with a consistent color scheme.
- Prioritize readability and accessibility.

2. **Navigation:**

- Place a navigation menu or sidebar for easy access to different sections of the platform.
- Use clear icons and labels for each section.

3. **User Profile:**

- Include a user profile picture and basic information.
- Show a list of recent activity and posts on the user's profile.

4. **Content Feed:**

- Display a personalized content feed with a mix of posts and recommendations.
- Use responsive card-based layouts for posts with images and captions.

5. **Search and Discovery:**

- Provide a search bar prominently at the top of the interface.
- Offer filtering and sorting options for search results.

6. **Content Creation:**

- Include a user-friendly content creation interface with options for adding media and tags.
- Use a WYSIWYG editor for text-based content.

7. **Notifications:**

- Place a notification icon or dropdown for instant access to alerts.
- Highlight unread notifications.

User Registration and Authentication Mechanisms:

1. **Registration:**

- Use HTTPS to encrypt data during registration.
- Verify email addresses by sending a confirmation link.
- Implement CAPTCHA or similar mechanisms to prevent bots.

2. **Authentication:**

- Use secure password hashing and salting techniques.
- Offer multi-factor authentication (MFA) options like SMS codes, email codes, or authenticator apps.
- Implement account lockout mechanisms after multiple failed login attempts to prevent brute force attacks.

3. **Session Management:**

- Use secure session tokens and set session timeouts.
- Provide a "Remember Me" option for convenience during subsequent logins.

4. **Security Measures:**

- Regularly update the platform to patch security vulnerabilities.

- Implement rate limiting to prevent login attempts from the same IP.

5. **Privacy Controls:**

- Allow users to set privacy preferences for their profiles and content.
- Clearly explain privacy settings to users.

6. **Data Protection:**

- Comply with data protection laws (e.g., GDPR, CCPA) and secure user data.

User Interface (HTML/CSS/JavaScript):

1. **HTML (index.html):**

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <header>
    <h1>Welcome to My Streaming Platform</h1>
  </header>
  <main>
    <video id="video-player" controls></video>
  </main>
```

```
    <footer>
        <button id="start-stream">Start Streaming</button>
        <button id="stop-stream">Stop Streaming</button>
    </footer>
    <script src="script.js"></script>
</body>
</html>
```

1. CSS (styles.css):

```
body {
    font-family: Arial, sans-serif;
}

header {
    text-align: center;
    background-color: #333;
    color: #fff;
    padding: 10px;
}

main {
    text-align: center;
    padding: 20px;
}

footer {
    text-align: center;
    background-color: #333;
    color: #fff;
    padding: 10px;
}
```

1. JavaScript (script.js):

```
const videoPlayer = document.getElementById('video-player');
```

```
const startStreamButton = document.getElementById('start-stream');
const stopStreamButton = document.getElementById('stop-stream');

// Event listeners for streaming buttons
startStreamButton.addEventListener('click', startStreaming);
stopStreamButton.addEventListener('click', stopStreaming);

function startStreaming() {
   const request = require('request');

// Replace these with your IBM Video Streaming API credentials
const apiKey = 'YOUR_API_KEY';
const accountId = 'YOUR_ACCOUNT_ID';
const accessToken = 'YOUR_ACCESS_TOKEN';

// Set up the endpoint for starting a stream
const startStreamEndpoint =
`https://api.video.ibm.com/streaming/v1/accounts/${accountId}/streams`;

// Define your streaming parameters
const streamParams = {
  name: 'MyStream',
  broadcasting: true,
  transcoding_profile: 'hd_4mbps',
};

// Create an HTTP POST request to start the stream
const options = {
  url: startStreamEndpoint,
  method: 'POST',
  json: true,
  body: streamParams,
  headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Content-Type': 'application/json',
    'Accept': 'application/json',
  },
};

// Send the request to start the stream
request(options, (error, response, body) => {
  if (!error && response.statusCode === 201) {
    console.log('Stream started successfully');
```

```
    console.log('Stream ID:', body.id);
  } else {
    console.error('Error starting the stream:', error);
  }
});
}

function stopStreaming() {
    const request = require('request');

// Replace these with your IBM Video Streaming API credentials and stream
ID
const apiKey = 'YOUR_API_KEY';
const accountId = 'YOUR_ACCOUNT_ID';
const accessToken = 'YOUR_ACCESS_TOKEN';
const streamId = 'YOUR_STREAM_ID';

// Set up the endpoint to stop the stream
const stopStreamEndpoint =
`https://api.video.ibm.com/streaming/v1/accounts/${accountId}/streams/${s
treamId}`;

// Create an HTTP DELETE request to stop the stream
const options = {
  url: stopStreamEndpoint,
  method: 'DELETE',
  headers: {
    'Authorization': `Bearer ${accessToken}`,
    'Accept': 'application/json',
  },
};

// Send the request to stop the stream
request(options, (error, response, body) => {
  if (!error && response.statusCode === 204) {
    console.log('Stream stopped successfully');
  } else {
    console.error('Error stopping the stream:', error);
  }
});
}
```

User Registration:

```
const express = require('express');
const bodyParser = require('body-parser');
const app = express();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const app = express();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

// Connect to your MongoDB database
mongoose.connect('mongodb://localhost/your-database-name', {
useNewUrlParser: true, useUnifiedTopology: true });

// Create a user schema
const userSchema = new mongoose.Schema({
  username: String,
  password: String,
  email: String,
});

const User = mongoose.model('User', userSchema);

// User registration endpoint
app.post('/register', async (req, res) => {
  const { username, password, email } = req.body;
```

```
  try {
    const user = new User({ username, password, email });
    await user.save();
    res.status(201).json({ message: 'User registered successfully', user });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'User registration failed' });
  }
});
const users = [];

// User registration endpoint
app.post('/register', (req, res) => {
  const { username, password, email } = req.body;

  const express = require('express');
const bodyParser = require('body-parser');
const app = express();
const pgp = require('pg-promise')();

app.use(bodyParser.json());

const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});

// Define the PostgreSQL database connection
const db = pgp({
  user: 'your-username',
  password: 'your-password',
  host: 'localhost',
  port: 5432,
  database: 'your-database-name',
});

// Create a user table schema
const createTable = `
  CREATE TABLE IF NOT EXISTS users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
```

```
    email VARCHAR(255) NOT NULL
  );
`;

db.none(createTable)
  .then(() => {
    console.log('User table created successfully');
  })
  .catch((error) => {
    console.error('Error creating user table:', error);
  });

// User registration endpoint
app.post('/register', async (req, res) => {
  const { username, password, email } = req.body;

  try {
    await db.none('INSERT INTO users(username, password, email)
VALUES($1, $2, $3)', [username, password, email]);
    res.status(201).json({ message: 'User registered successfully' });
  } catch (error) {
    console.error('Error registering user:', error);
    res.status(500).json({ message: 'User registration failed' });
  }
});
  const user = { username, password, email };
  users.push(user);

  res.status(201).json({ message: 'User registered successfully', user });
});
```