

# Final Project: Social Media Dashboard



**Estimated time needed:** 3 hours

In this hands-on learning experience, you will delve into the world of web development and build a robust Social Media Dashboard using Node.js, Express, and MongoDB. This project will guide you through essential concepts such as user authentication, API development, middleware usage, and error handling. This will provide you with a practical understanding of building modern web applications.

## Learning Objectives

After completing this lab you will be able to:

1. Create Users and Posts collections in MongoDB.
2. Secure JWT and middleware function for authentication.
3. Implement API endpoints for user management and social media posts.
4. Develop the frontend component for a social media dashboard.
5. Deploy the Social Media Dashboard app on Docker.
6. Work the Social Media Dashboard app.

## About Skills Network Cloud IDE

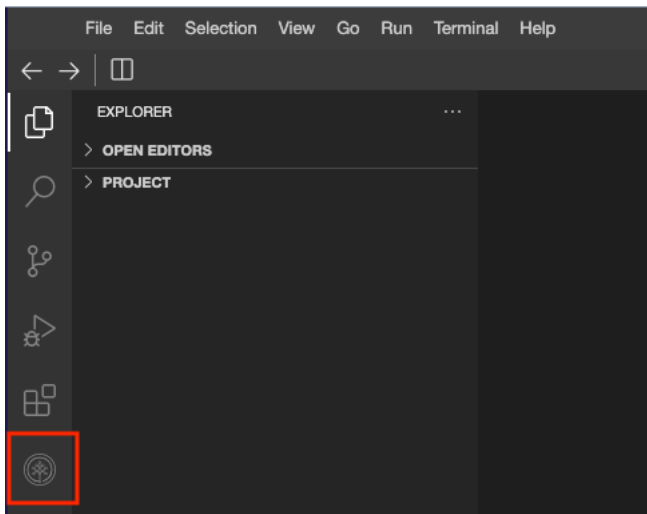
Skills Network Cloud IDE (based on Theia and Docker) provides an environment for hands on labs for course and project related labs. Theia is an open source IDE (Integrated Development Environment).

## Important Notice about this Lab Environment

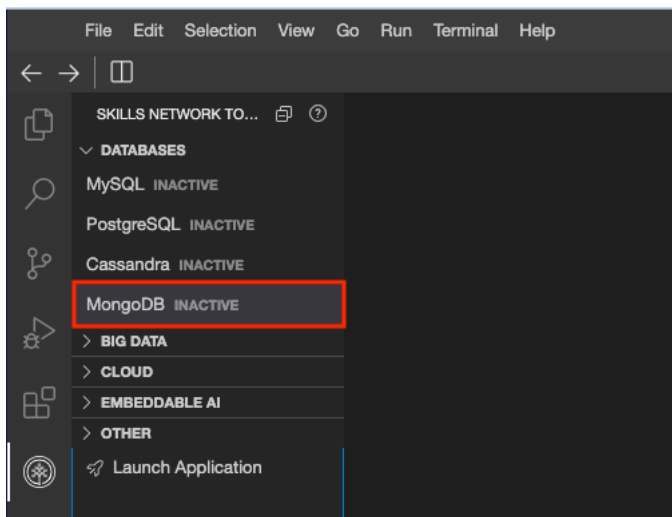
Please be aware that sessions for this lab environment are not persisted. Every time you connect to this lab, a new environment is created for you. Any data you may have saved in the earlier session would get lost. Plan to complete these labs in a single session, to avoid losing your data.

## Exercise 1 - Start MongoDB Server

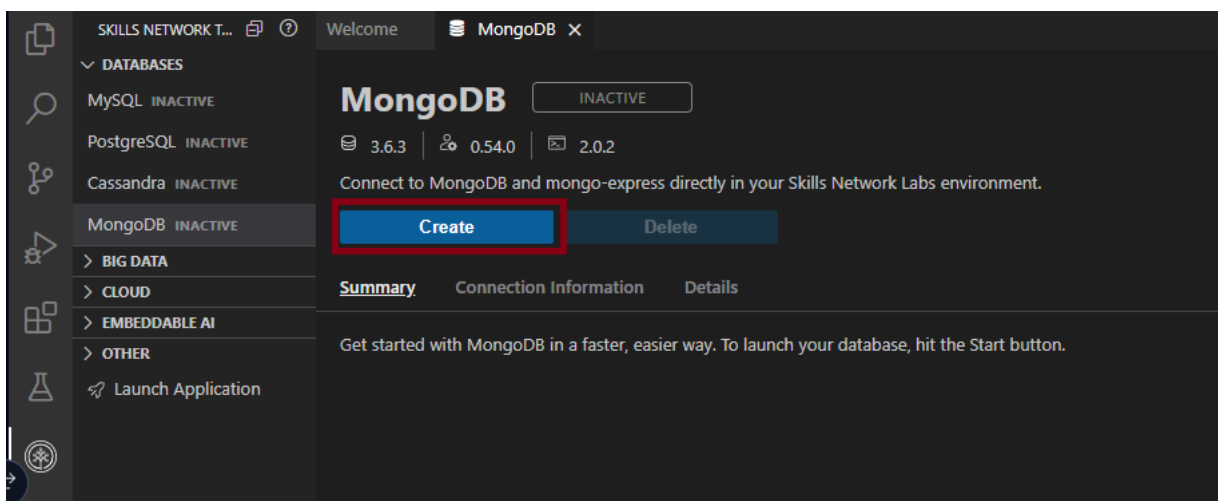
1. Click on the SkillsNetwork Toolbox Icon.



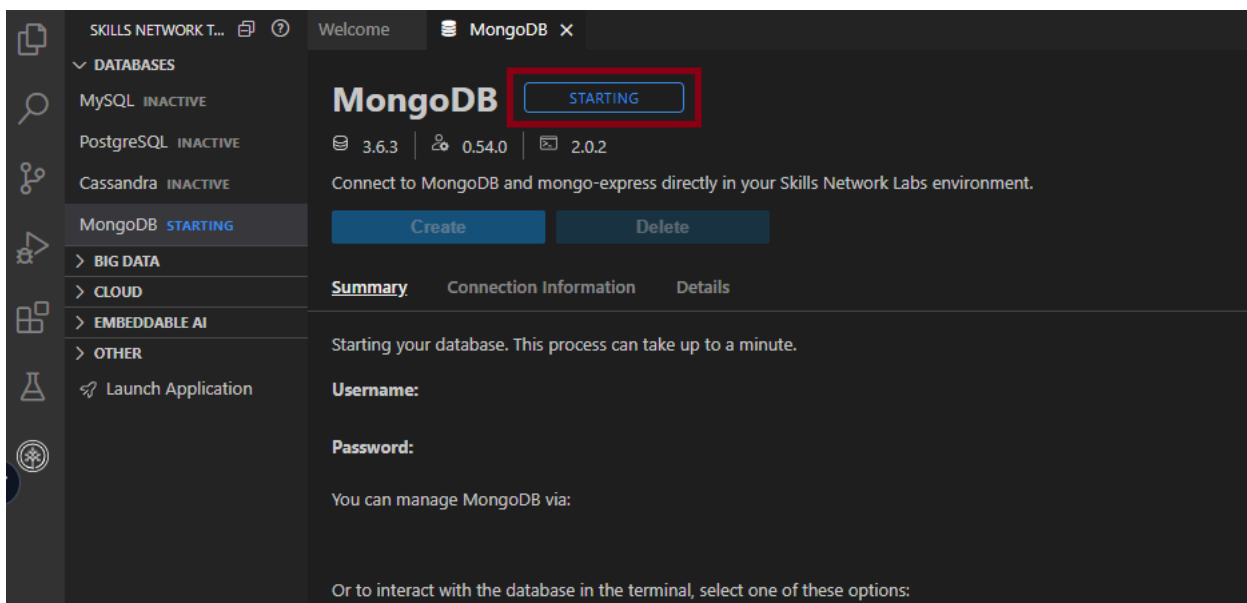
2. You will notice MongoDB listed, under DATABASES, but inactive. Which means the database is not available to use.



3. Once you click on it, you will see more details and a button to start.



4. Clicking on the start button will run a background process to configure and start your MongoDB server.



5. Shortly after that, your server is ready for use. This deployment has access control enabled and MongoDB enforces authentication. So, Copy the MONGO\_HOST, MONGO\_PORT, MONGO\_PASSWORD which is generated under connection information tab. You will need this information in the next step of this lab. Since the lab environment is temporary, the password and Host Id might be different the next time you start MongoDB database service.

MongoDB

ACTIVE

3.6.3 | 0.54.0 | 2.0.2

Connect to MongoDB and mongo-express directly in your Skills Network Labs environment.

CreateDelete

SummaryConnection InformationDetails

MONGO\_USERNAME:root

MONGO\_HOST:172.21.41.238

MONGO\_PORT:27017

URL:https://labs-mongo-thundering-most-fall.mongo.databases.labs.skills.network

MONGO\_URL:https://labs-mongo-thundering-most-fall.mongo.databases.labs.skills.network

MongoDB CLI Command:mongosh mongodb://root:aPmca0TKv1vxBt1ANcooACHH@172.21.41.238:27017

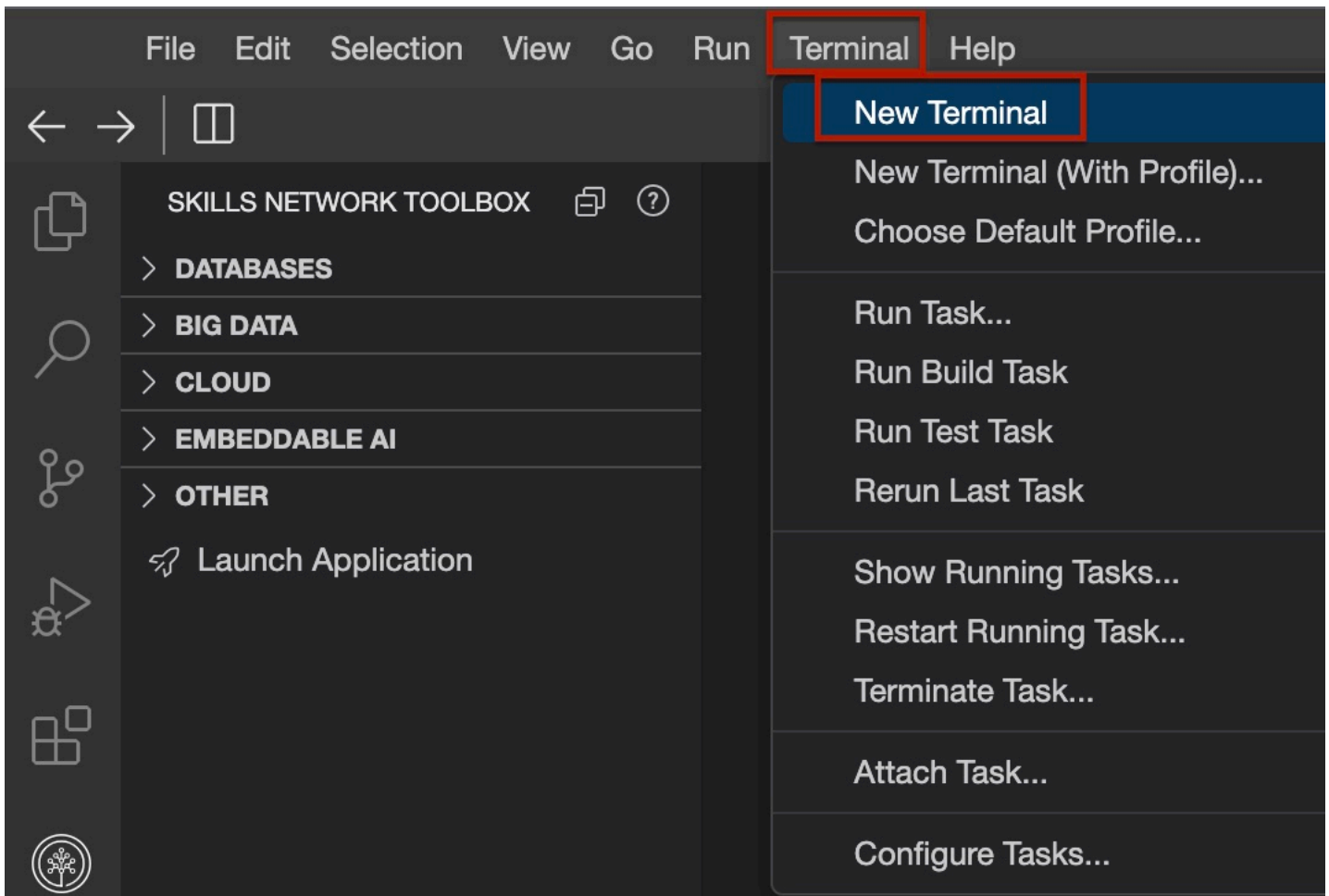
MONGO\_COMMAND:mongosh mongodb://root:aPmca0TKv1vxBt1ANcooACHH@172.21.41.238:27017

MONGO\_PASSWORD:aPmca0TKv1vxBt1ANcooACHH

MONGO\_TITLE:MongoDB Database

MONGO\_ID:labs-mongo-thundering-most-fall

## Exercise 2 - Secure JWT and Middleware Function for Authentication



1. In the terminal, clone the git repository which has the starter code.

```
git clone https://github.com/ibm-developer-skills-network/rofg-backend-nodejs-finalproject.git
```

2. Change to the new directory that you just cloned.

```
cd rofg-backend-nodejs-finalproject
```

3. To set up the necessary Node.js packages, execute the following command in the terminal. This will install the required packages specified in the packages.json file, including mongoose for connecting to MongoDB server, express for creating API endpoints for user registration and login, and body-parser for parsing JSON input sent to the API endpoints.

```
npm install
```

4. Launch your editor and examine the contents of the app.js file. Replace the MongoDB password & MongoDB host with the password you copied in the previous exercise.
5. Paste the below code for authenticateJWT Function to authenticate users based on a JSON Web Token (JWT) stored in the session. You will write code create this token when the user logs in.

```
function authenticateJWT(req, res, next) {
  const token = req.session.token;
  if (!token) return res.status(401).json({ message: 'Unauthorized' });
  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(401).json({ message: 'Invalid token' });
  }
}
```

This code is an Express.js middleware (authenticateJWT) that checks for a JWT in the session of an incoming request. If the token is present, it verifies it using a secret key (SECRET\_KEY). If verification succeeds, the decoded user information is attached to the request (req.user), and the middleware passes control to the next function. If there's no token or verification fails, it sends a 401 Unauthorized response with an appropriate message.

6. Paste the below code for requireAuth Function to check whether a user is authenticated. If not, it redirects them to the login page.

```
function requireAuth(req, res, next) {
  const token = req.session.token;
  if (!token) return res.redirect('/login');
  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    req.user = decoded;
    next();
  } catch (error) {
    return res.redirect('/login');
  }
}
```

This Express.js middleware (requireAuth) checks for a JWT in the session of an incoming request. If the token is present, it verifies it using a secret key (SECRET\_KEY). If verification succeeds, the decoded user information is attached to the request (req.user), and it passes control to the next function. If there's no token or verification fails, it redirects the user to the '/login' page.

## Exercise 3 - Implement API Endpoints for User Management and Social Media Posts

1. Insert the following code into the designated placeholder in the app.js file for routing HTML files.

```
app.get('/', (req, res) => res.sendFile(path.join(__dirname, 'public', 'index.html')));
app.get('/register', (req, res) => res.sendFile(path.join(__dirname, 'public', 'register.html')));
app.get('/login', (req, res) => res.sendFile(path.join(__dirname, 'public', 'login.html')));
app.get('/post', requireAuth, (req, res) => res.sendFile(path.join(__dirname, 'public', 'post.html')));
app.get('/index', requireAuth, (req, res) => res.sendFile(path.join(__dirname, 'public', 'index.html'), { username: req.user.username }));
```

- These routes handle GET requests to specific paths and send corresponding HTML files in response.
  - For example, when a user accesses /register, it sends the 'register.html' file in response.
2. Place the provided code into the assigned location in the app.js file to enable user registration with applied JWT authentication.

```
app.post('/register', async (req, res) => {
  const { username, email, password } = req.body;
  try {
    const existingUser = await User.findOne({ $or: [{ username }, { email }] });
    if (existingUser) return res.status(400).json({ message: 'User already exists' });
    const newUser = new User({ username, email, password });
    await newUser.save();
    const token = jwt.sign({ userId: newUser._id, username: newUser.username }, SECRET_KEY, { expiresIn: '1h' });
    req.session.token = token;
    res.send({ "message": `The user ${username} has been added` });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
});
```

- Handles POST requests to the '/register' endpoint.
  - Checks if the user already exists in the database.
  - If not, creates a new user, generates a JWT token, stores it in the session, and redirects to the '/index' route.
3. Insert the given code into the specified area in the app.js file to implement user login with applied JWT authentication.

```
app.post('/login', async (req, res) => {
  const { username, password } = req.body;
  try {
    const user = await User.findOne({ username, password });
    if (!user) return res.status(401).json({ message: 'Invalid credentials' });
    const token = jwt.sign({ userId: user._id, username: user.username }, SECRET_KEY, { expiresIn: '1h' });
    req.session.token = token;
    res.redirect({ "message": `${user.username} has logged in` });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Internal Server Error' });
  }
});
```

- Handles POST requests to the '/login' endpoint.
  - Checks if the provided username and password match a user in the database.
  - If successful, creates a JWT token, stores it in the session, and redirects to the '/index' route.
4. Integrate the supplied code into the specified section of the app.js file to implement post creation with JWT authentication.

```
app.post('/posts', authenticateJWT, (req, res) => {
  const { text } = req.body;
  if (!text || typeof text !== 'string') return res.status(400).json({ message: 'Please provide valid post content' });
  const newPost = { userId: req.user.userId, text };
  posts.push(newPost);
  res.status(201).json({ message: 'Post created successfully' });
});
```

- Handles POST requests to the '/posts' endpoint, requiring authentication.
  - Validates and creates a new post, pushing it to a 'posts' array.
5. Insert the provided code into the assigned section of the app.js file to enable post updation with implemented JWT authentication.

```
app.put('/posts/:postId', authenticateJWT, (req, res) => {
  const postId = parseInt(req.params.postId);
  const { text } = req.body;
  const postIndex = posts.findIndex((post) => post.id === postId && post.userId === req.user.userId);
  if (postIndex === -1) return res.status(404).json({ message: 'Post not found' });
  posts[postIndex].text = text;
  res.json({ message: 'Post updated successfully', updatedPost: posts[postIndex] });
});
```

- Handles PUT requests to the '/posts/:postId' endpoint, requiring authentication.
  - Finds and updates the text of a specific post in the 'posts' array.
6. Place the provided code into the designated section of the app.js file to implement post deletion with applied JWT authentication.

```
app.delete('/posts/:postId', authenticateJWT, (req, res) => {
  const postId = parseInt(req.params.postId);
  const postIndex = posts.findIndex((post) => post.id === postId && post.userId === req.user.userId);
  if (postIndex === -1) return res.status(404).json({ message: 'Post not found' });
  const deletedPost = posts.splice(postIndex, 1)[0];
  res.json({ message: 'Post deleted successfully', deletedPost });
});
```

```
});
```

- Handles DELETE requests to the '/posts/:postId' endpoint, requiring authentication.
- Finds and deletes a specific post from the 'posts' array.

7. Insert the provided code into the specified portion of the app.js file to implement user logout with applied JWT authentication.

```
app.get('/logout', (req, res) => {
  req.session.destroy((err) => {
    if (err) console.error(err);
    res.redirect('/login');
  });
});
```

- Handles GET requests to the '/logout' endpoint.
- Destroys the user session and redirects to the '/login' route.

► [Click here for the full solution](#)

8. Test the /register and /login with cURL commands for the following cases. Make a note of the response messages on notepad or any other editor.

- Register with a new username, password, email  
eg:

```
curl -X POST -H 'Content-Type: application/json' -d '{
  "username": "coopercoote",
  "email": "ccoote@gmail.com",
  "password": "cg123"}' -k 'http://localhost:3000/register'
```

- Register with the same username, email and password again. You should get appropriate error message.
- Login with username password you just registered  
eg:

```
curl -X POST -H 'Content-Type: application/json' -d '{
  "username": "coopercoote",
  "password": ""}' -k 'http://localhost:3000/login'
```

- Login with correct username and wrong password. You should get appropriate error message. Make a note of the response message on notepad or any other editor.

## Exercise 4 - Developing the frontend component for a Social Media Dashboard

1. Paste the below code in public/index.html.

This HTML code is the main page of a Social Media Dashboard App.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Social Media Dashboard App</title>
  <style>
    body {
      font-family: 'Roboto', sans-serif;
      background-color: #f5f5f5;
      text-align: center;
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    header {
      background-color: #0077cc;
      color: #fff;
      padding: 20px;
      text-align: right;
    }
    h1 {
      color: #333;
      font-size: 2.5em;
      margin-bottom: 20px;
      text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
    }
    p {
      color: #555;
      font-size: 1.2em;
    }
    .app-names {
      display: flex;
      justify-content: center;
      margin-top: 30px;
    }
    .app-names div {
      padding: 10px 20px;
      margin: 0 10px;
      background-color: #0099ff;
      color: #fff;
      border-radius: 5px;
      font-weight: bold;
      text-transform: uppercase;
    }
    .button-link {
      margin-top: 30px;
      padding: 15px 30px;
      font-size: 1.2em;
```

```

border: 2px solid #0099ff;
border-radius: 8px;
background-color: #fff;
color: #0099ff;
cursor: pointer;
text-decoration: none;
display: inline-block;
}
</style>
</head>
<body>
<header>
<button id="registerBtn" class="button-link" style="display:none" onclick="window.location.href='/register'">Register</button>
<button id="loginBtn" class="button-link" style="display:none" onclick="window.location.href='/login'">Login</button>
<button id="logoutBtn" class="button-link" style="display:none" onclick="logout()">Logout</button>
</header>
<h1>Welcome to the Social Media Dashboard App!</h1>
<p id="welcomeMessage">Explore and manage your social media accounts in one place.</p>
<div class="app-names">
<div>FaceSnap</div>
<div>InstaBook</div>
<div>TweetChat</div>
</div>
<button id="createPostBtn" class="button-link" style="display:none" onclick="redirectToPostCreation()">Create Post</button>
<script>
function getUsernameFromURL() {
return new URLSearchParams(window.location.search).get('username');
}
function handleAuthentication() {
const username = getUsernameFromURL();
const registerBtn = document.getElementById('registerBtn');
const loginBtn = document.getElementById('loginBtn');
const logoutBtn = document.getElementById('logoutBtn');
const createPostBtn = document.getElementById('createPostBtn');
if (username) {
registerBtn.style.display = loginBtn.style.display = 'none';
logoutBtn.style.display = createPostBtn.style.display = 'inline-block';
welcomeMessage.innerText = `Welcome, ${username}!`;
} else {
registerBtn.style.display = loginBtn.style.display = 'inline-block';
logoutBtn.style.display = createPostBtn.style.display = 'none';
}
}
function logout() {
window.location.href = '/logout';
}
function redirectToPostCreation() {
window.location.href = '/post';
}
window.onload = function () {
getUsernameFromURL();
handleAuthentication();
};
</script>
</body>
</html>

```

The web app has a header with Register, Login, and Logout buttons. It displays a welcome message and stylized app names. The Create Post button visibility depends on authentication. JavaScript functions handle authentication, retrieve usernames, and control redirections. These functions run on page load for dynamic user interaction.

2. Paste the below code in public/login.html.

```

<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Login</title>
<style>
body{font-family:Arial,sans-serif;background-color:#f8f8f8;margin:0}
.container{max-width:400px;margin:50px auto;padding:20px;background-color:#fff;box-shadow:0 0 10px rgba(0,0,0,.1);border-radius:8px}
h1{color:#333;font-size:2em;margin-bottom:20px}
.auth-form label{display:block;margin-bottom:8px;font-weight:bold}
.auth-form input{width:100%;padding:10px;margin-bottom:15px;border:1px solid #ccc;border-radius:5px;box-sizing:border-box}
.auth-form button{background-color:#0077cc;color:#fff;padding:10px 20px;border:none;border-radius:5px;cursor:pointer}
.auth-form button:hover{background-color:#005faa}
p{margin-top:20px;font-size:1.2em}
p a{color:#0077cc;font-weight:bold;text-decoration:none}
p a:hover{text-decoration:underline}
</style>
</head>
<body>
<div class="container">
<h1>Login</h1>
<form action="/login" method="post" class="auth-form">
<label for="username">Username:</label>
<input type="text" id="username" name="username" required>
<br>
<label for="password">Password:</label>
<input type="password" id="password" name="password" required>
<br>
<button type="submit">Login</button>
</form>
<p>Don't have an account? <a href="/register">Register here</a>.</p>
</div>
</body>
</html>

```

This HTML code represents the login page for a Social Media Dashboard app. It includes a form for users to enter their username and password. The form action is set to "/login" for handling login requests. The page encourages users to register if they don't have an account, providing a link to the registration page ("/register"). The styling ensures a clean and responsive layout.

3. Paste the below code in public/register.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Register</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background-color: #f8f8f8;
      margin: 0;
    }
    .container {
      max-width: 400px;
      margin: 50px auto;
      padding: 20px;
      background-color: #fff;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
      border-radius: 8px;
    }
    h1 {
      color: #333;
      font-size: 2em;
      margin-bottom: 20px;
    }
    .auth-form label {
      display: block;
      margin-bottom: 8px;
      font-weight: bold;
    }
    .auth-form input {
      width: 100%;
      padding: 10px;
      margin-bottom: 15px;
      border: 1px solid #ccc;
      border-radius: 5px;
      box-sizing: border-box;
    }
    .auth-form button {
      background-color: #0077cc;
      color: #fff;
      padding: 10px 20px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
    }
    .auth-form button:hover {
      background-color: #005faa;
    }
    p {
      margin-top: 20px;
      font-size: 1.2em;
    }
    p a {
      color: #0077cc;
      font-weight: bold;
      text-decoration: none;
    }
    p a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>Register</h1>
    <form action="/register" method="post" class="auth-form">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" required>
      <br>
      <label for="email">Email:</label>
      <input type="email" id="email" name="email" required>
      <br>
      <label for="password">Password:</label>
      <input type="password" id="password" name="password" required>
      <br>
      <button type="submit">Register</button>
    </form>
    <p>Already have an account? <a href="/login">Login here</a>.</p>
  </div>
</body>
</html>
```

This HTML code represents the registration page for a Social Media Dashboard app. It includes a form for users to provide their username, email, and password. The form action is set to "/register" for handling registration requests. The page encourages users who already have an account to log in by providing a link to the login page ("/login"). The styling ensures a clean and responsive layout.

4. Paste the below code in public/post.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Create and Manage Posts</title>
  <style>
    body {
      font-family: 'Roboto', sans-serif;
      background-color: #f5f5f5;
      text-align: center;
      margin: 0;
    }
```



```

padding: 0;
box-sizing: border-box;
}
header {
background-color: #0077cc;
color: #fff;
padding: 20px;
text-align: right;
}
h1 {
color: #333;
font-size: 2.5em;
margin-bottom: 20px;
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}
.post-form, .post-list {
width: 60%;
margin: 50px auto;
padding: 20px;
background-color: #fff;
border-radius: 8px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
label, textarea {
display: block;
margin-bottom: 10px;
font-size: 1.2em;
color: #333;
}
textarea {
width: 100%;
padding: 10px;
margin-bottom: 20px;
border: 1px solid #ccc;
border-radius: 4px;
box-sizing: border-box;
}
button {
background-color: #0099ff;
color: #fff;
padding: 10px 20px;
font-size: 1.2em;
border: none;
border-radius: 8px;
cursor: pointer;
}
.post {
border-bottom: 1px solid #ccc;
padding: 10px;
margin-bottom: 10px;
display: flex;
justify-content: space-between;
align-items: center;
}
.post-text {
flex-grow: 1;
margin-right: 10px;
font-size: 1.1em;
color: #333;
}
.edit-post-btn, .delete-post-btn {
padding: 8px;
margin-right: 5px;
cursor: pointer;
background-color: #0099ff;
color: #fff;
border: none;
border-radius: 4px;
}
.pagination {
margin-top: 20px;
}
.pagination button {
background-color: #0099ff;
color: #fff;
border: none;
padding: 10px;
margin: 0 5px;
cursor: pointer;
border-radius: 4px;
transition: background-color 0.3s;
}
.pagination button:hover {
background-color: #0077cc;
}
</style>
</head>
<body>
<header>
<button id="logoutBtn" onclick="logout()">Logout</button>
</header>
<h1>Create and Manage Posts</h1>
<div class="post-form">
<label for="text">Create a new post:</label>
<textarea id="text" name="text" rows="4" required></textarea>
<button onclick="createPost()">Create Post</button>
</div>
<div class="post-list" id="postList"></div>
<div class="pagination" id="pagination"></div>
<script>
const posts = [];
const postsPerPage = 3;
let currentPage = 1;
function createPost() {

```

```

const text = document.getElementById('text').value;
if (text) {
  posts.push({ id: posts.length + 1, text });
  updatePostList();
  document.getElementById('text').value = '';
}
}
function deletePost(postId) {
  const index = posts.findIndex(post => post.id === postId);
  if (index !== -1) {
    posts.splice(index, 1);
    updatePostList();
  }
}
function editPost(postId) {
  const newText = prompt('Edit the post:', posts.find(post => post.id === postId).text);
  if (newText !== null) {
    const index = posts.findIndex(post => post.id === postId);
    if (index !== -1) {
      posts[index].text = newText;
      updatePostList();
    }
  }
}
function updatePostList() {
  const postList = document.getElementById('postList');
  const pagination = document.getElementById('pagination');
  const startIndex = (currentPage - 1) * postsPerPage;
  const endIndex = startIndex + postsPerPage;
  postList.innerHTML = '';
  if (posts.length === 0) {
    postList.innerHTML = '<p>No posts available.</p>';
    pagination.innerHTML = '';
    return;
  }
  posts.slice(startIndex, endIndex).forEach(post => {
    const postElement = document.createElement('div');
    postElement.classList.add('post');
    const postText = document.createElement('div');
    postText.classList.add('post-text');
    postText.textContent = post.text;
    const editButton = document.createElement('span');
    editButton.classList.add('edit-post-btn');
    editButton.textContent = 'Edit';
    editButton.onclick = () => editPost(post.id);
    const deleteButton = document.createElement('span');
    deleteButton.classList.add('delete-post-btn');
    deleteButton.textContent = 'Delete';
    deleteButton.onclick = () => deletePost(post.id);
    postElement.appendChild(postText);
    postElement.appendChild(editButton);
    postElement.appendChild(deleteButton);
    postList.appendChild(postElement);
  });
  const totalPages = Math.ceil(posts.length / postsPerPage);
  pagination.innerHTML = '';
  const maxPages = Math.min(5, totalPages);
  for (let i = 1; i <= maxPages; i++) {
    const pageButton = document.createElement('button');
    pageButton.textContent = i;
    pageButton.onclick = () => {
      currentPage = i;
      updatePostList();
    };
    pagination.appendChild(pageButton);
  }
}
function logout() {
  window.location.href = '/logout';
}
window.onload = () => updatePostList();
</script>
</body>
</html>

```

This HTML code represents a Social Media Dashboard for creating and managing posts. Users can create posts with a text input, and the "Create Post" button adds them to a list. The page displays posts with options to edit and delete, and implement pagination for multiple pages. Users can also log out. JavaScript handles dynamic functionalities like post creation, deletion, and pagination without page reloads.

5. In `app.js` replace the JSON response on successful registration, with the following.

```
res.redirect(`/index?username=${newUser.username}`);
```

This will redirect the response to `/index` with the username parameter set.

6. In `app.js` replace the JSON response on successful login, with the following.

```
res.redirect(`/index?username=${user.username}`);
```

This will redirect the response to `/index` with the username parameter set.

## Exercise 5 - Deploy the Social Media Dashboard app on Docker

1. Replace the mongo URL and connect in `app.js` with the following.

```
const uri = "mongodb://mongodb:27017";
mongoose.connect(uri, { dbName: 'SocialDB' });
```

2. Run the following command to build the Docker app

```
docker build . -t socialapp
```

```
theia@theiadocker-pallavir:/home/project/abltc-backend-nodejs-customerportal$ d
ocker build . -t customerapp
[+] Building 20.5s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 436B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18.12.1-bullseye-slim
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load build context
=> => transferring context: 15.90MB
=> [1/5] FROM docker.io/library/node:18.12.1-bullseye-slim@sha256:70bf84739156657c85440e6a55a3d7
=> => resolve docker.io/library/node:18.12.1-bullseye-slim@sha256:70bf84739156657c85440e6a55a3d7
=> => sha256:3f4ca61aafcd4fc07267a105067db35c0f0ac630e1970f3cd0c7bf552780e985 31.40MB / 31.40MB
=> => sha256:00fde01815c92cc90586fcf531723ab210577a0f1cb1600f08d9f8e12c18f108 4.18kB / 4.18kB
=> => sha256:723367e6ef2d175cd363a4ca96a65ac6fa3f388506de7c0489318e8d8ed59dd7 45.15MB / 45.15MB
=> => sha256:70bf84739156657c85440e6a55a3d77a7cac668f9c4c3c44005bc29bdc529db7 1.21kB / 1.21kB
=> => sha256:0c3ea57b6c560f83120801e222691d9bd187c605605185810752a19225b5e4d9 1.37kB / 1.37kB
```

3. The docker-compose.yml has been created to run two containers, one for Mongo and the other for the Node app. Run the following command to run the server:

```
docker-compose up
```

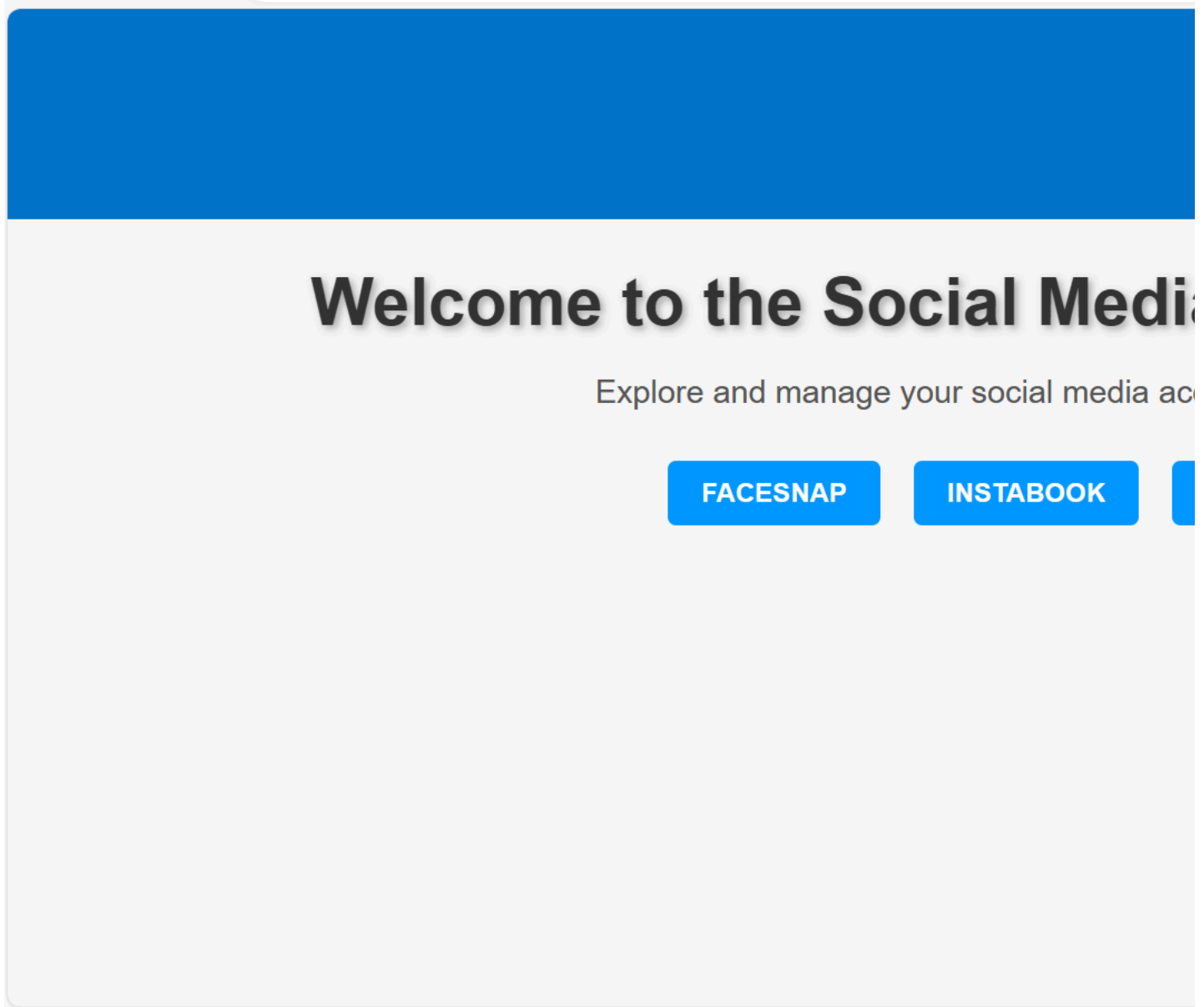
```
theia@theiadocker-pallavir:/home/project/SocialMediaDashboard$ docker-compose up
[+] Running 8/10
 : mongodb 9 layers [#####] 31.08MB/224.4MB Pulling
  ✓ 29202e855b20 Download complete
  ✓ 7513301b17d7 Download complete
  ✓ 8584f3ef3048 Download complete
  ✓ 5b7464f50635 Download complete
  ✓ c6ff633f781c Download complete
  ✓ 5644f6e5c0e6 Download complete
  ✓ d930da07d87d Download complete
  .: 06fc900f7e64 Downloading 31.08MB/224.4MB
  ✓ 17a4f29a303b Download complete
```

4. Click on the button below to launch the application.

[Social Media Dashboard](#)

► Click here if the button above doesn't work

4. Verify the dashboard:



## Exercise 6 - Working of Social Media Dashboard app

Task 1: Take a screenshot of the launched application to verify the successful creation of the social media dashboard. Save the screenshot as social\_media\_app.png.

# Welcome to the Social Media

Explore and manage your social media account

FACESNAP

INSTABOOK

TWITTER

Sample

Task 2: Append "/post" to the end of the URL and verify that you are redirected to the login page, indicating that posting is restricted without authentication. Take a screenshot and save it as post\_authentication.png.

## Login

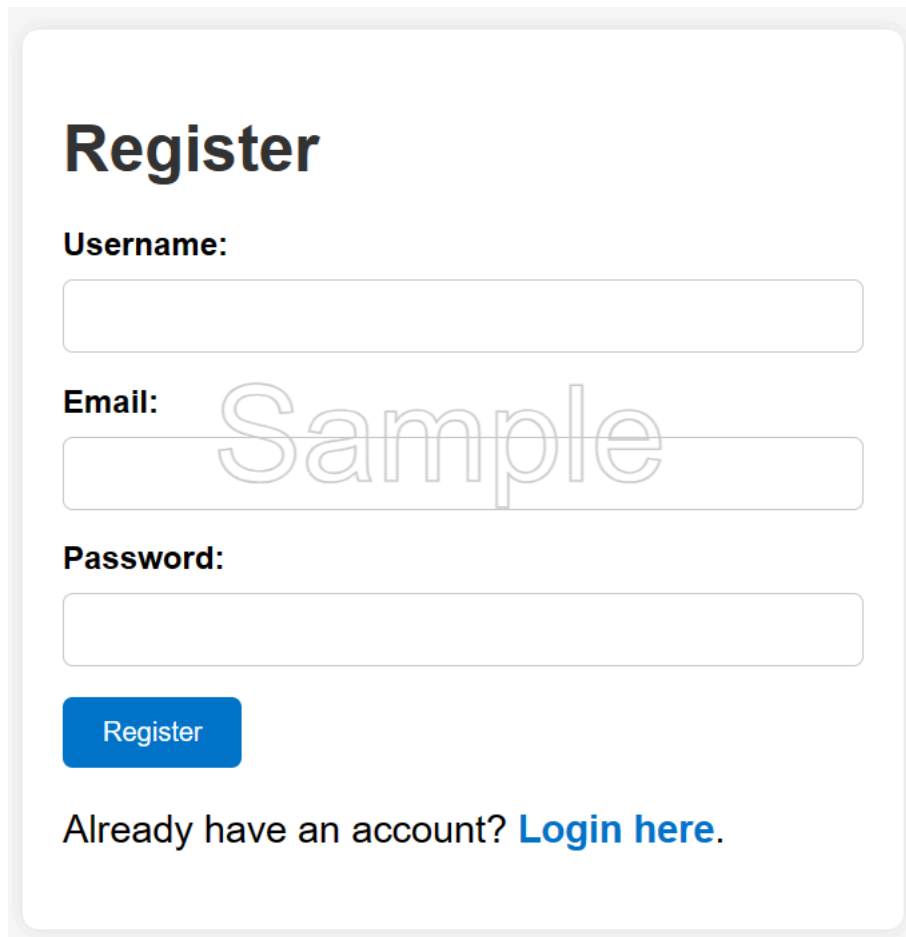
Username:

Password:

Login

Don't have an account? [Register here.](#)

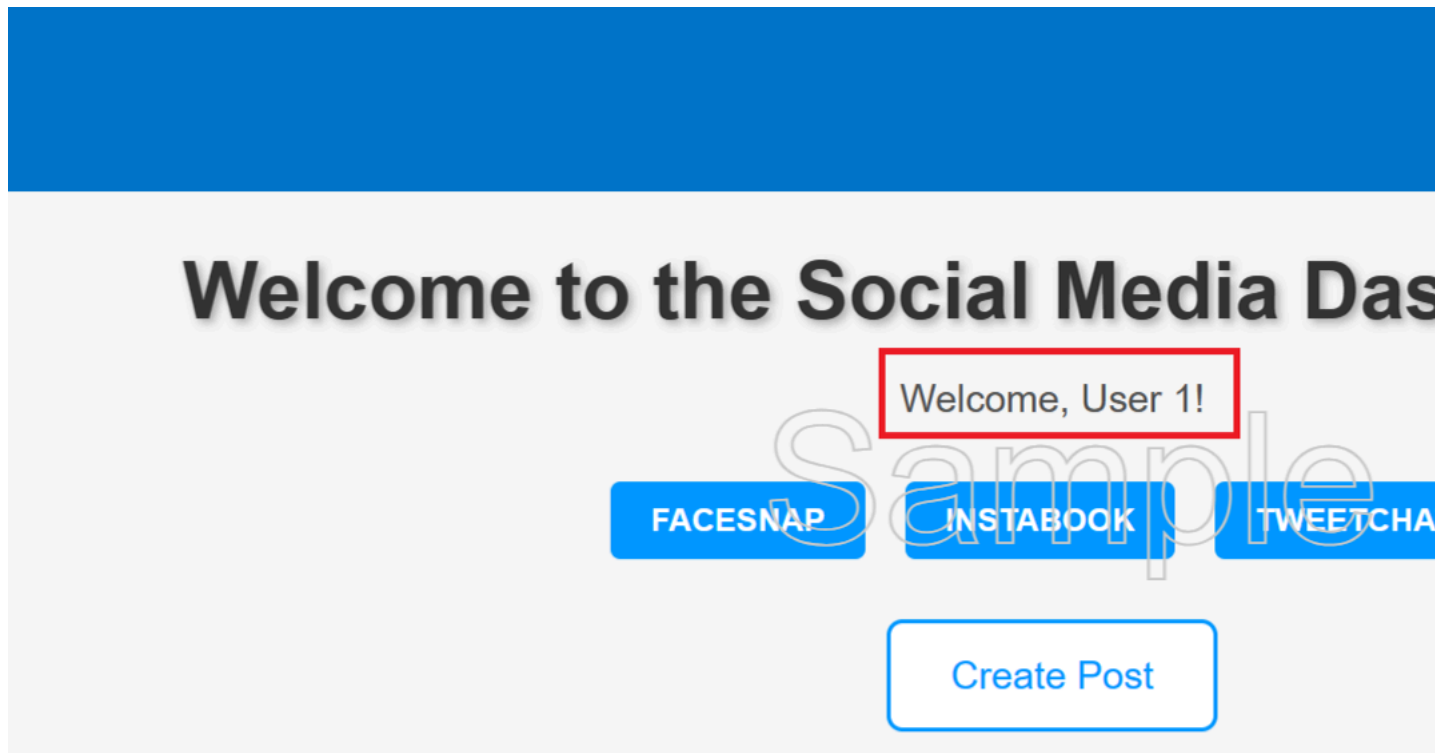
Task 3: Click on the register button, provide the required details, and complete the registration process to gain access to the application. Take a screenshot and save it as register.png.



The screenshot shows a registration form with the following elements:

- Register**: A large heading at the top.
- Username:** A label followed by a text input field.
- Email:** A label followed by a text input field. A large, faint "Sample" watermark is visible over the email field.
- Password:** A label followed by a text input field.
- Register**: A blue button with white text.
- Already have an account? [Login here.](#)**: A link to the login page.

Task 4: Upon successful registration, observe the welcome message containing your username on the app interface. Take a screenshot and save it as register\_success.png.



Task 5: Confirm the visibility of the "Create post" button on the application page post-registration. Take a screenshot and save it as create\_postbutton.png.

# Welcome to the Social Media Das

Welcome, User 1!

FACESNAP

INSTABOOK

TWEETCHAT

Create Post

Task 6: Verify your ability to successfully create a post, ensuring that the content is accurately reflected. Take a screenshot and save it as create\_post.png.

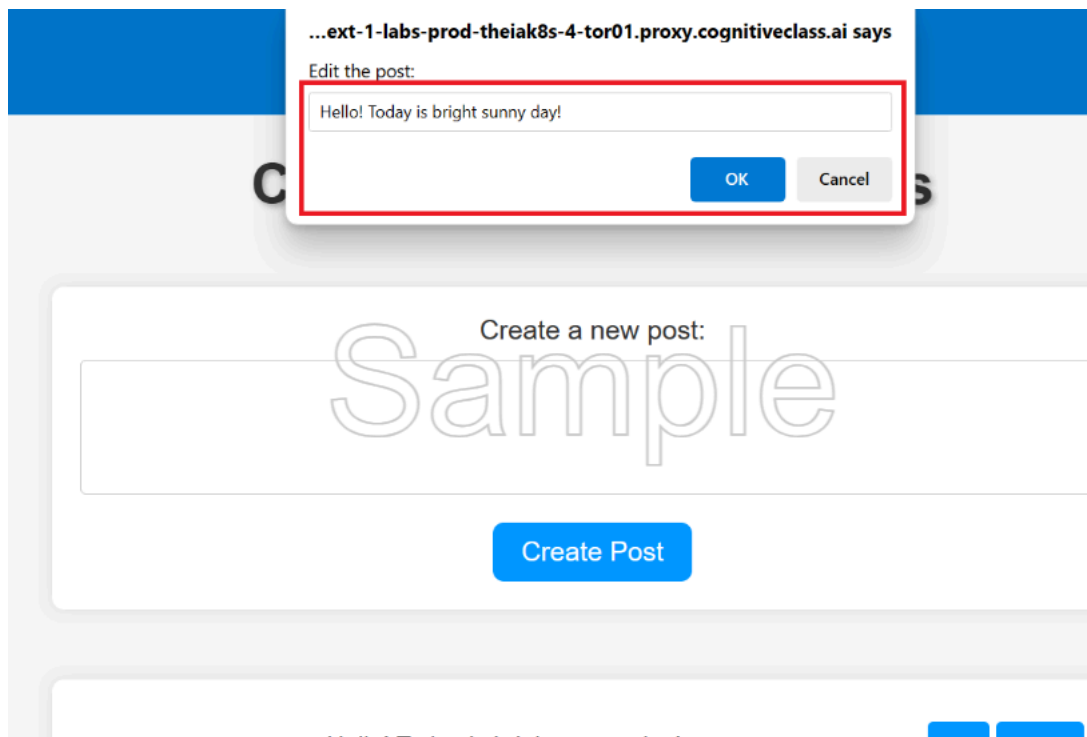
## Create and Manage Posts

Create a new post:

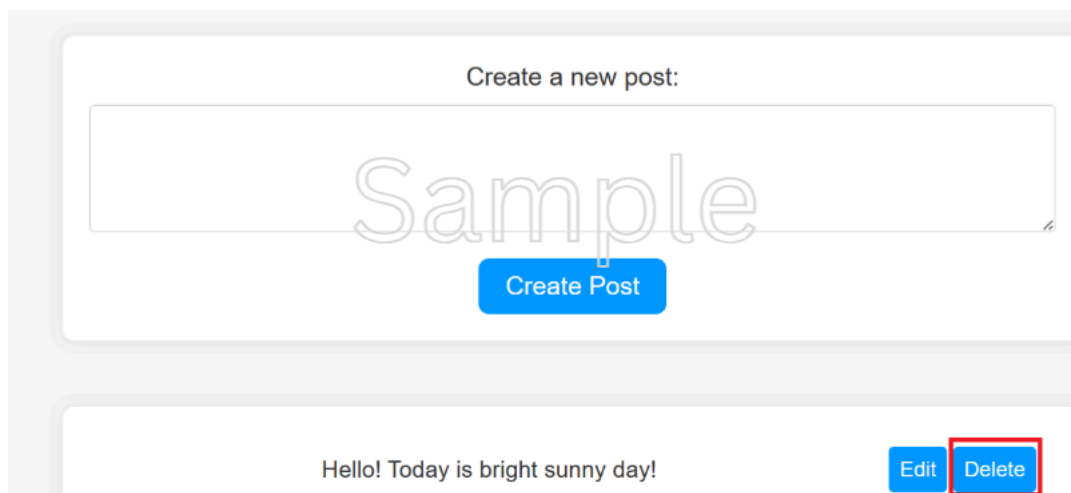
Hello! Today is bright sunny day!

Create Post

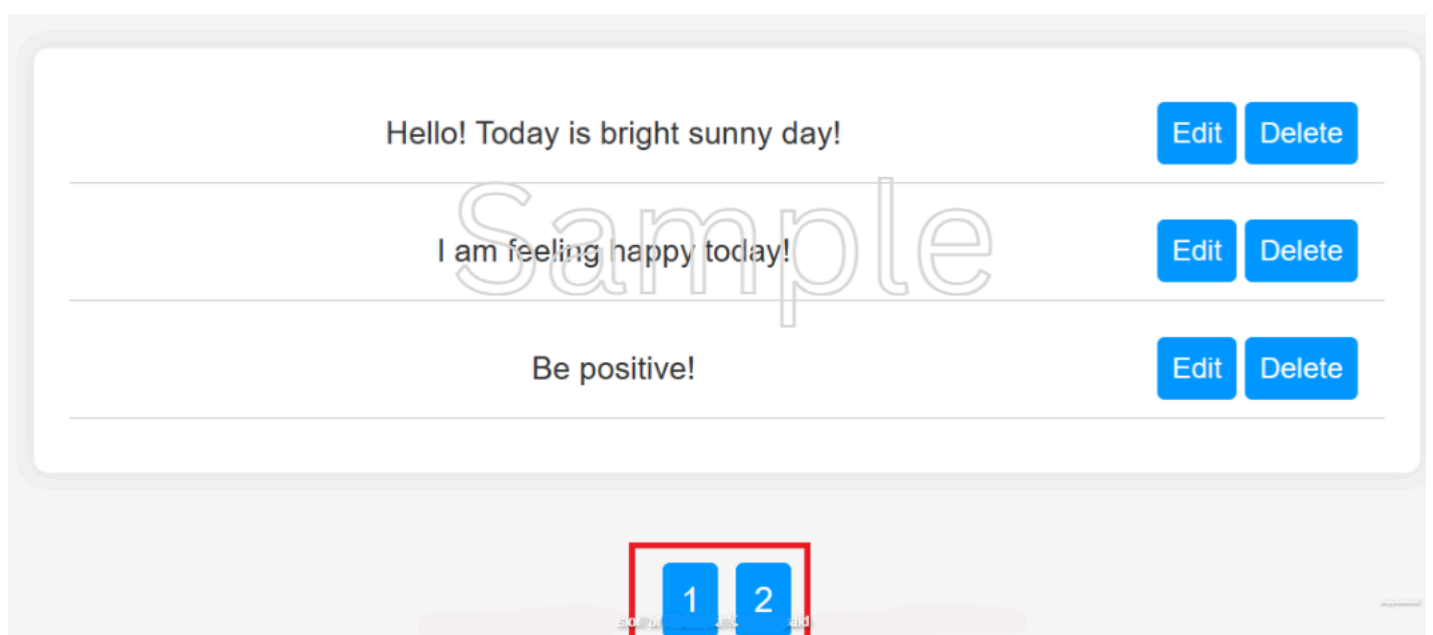
Task 7: Confirm your capability to edit the posts you've created without encountering issues. Take a screenshot and save it as edit\_post.png.



Task 8: Test your ability to delete the posts you've created with the app, ensuring the process works correctly. Take a screenshot and save it as delete\_post.png.

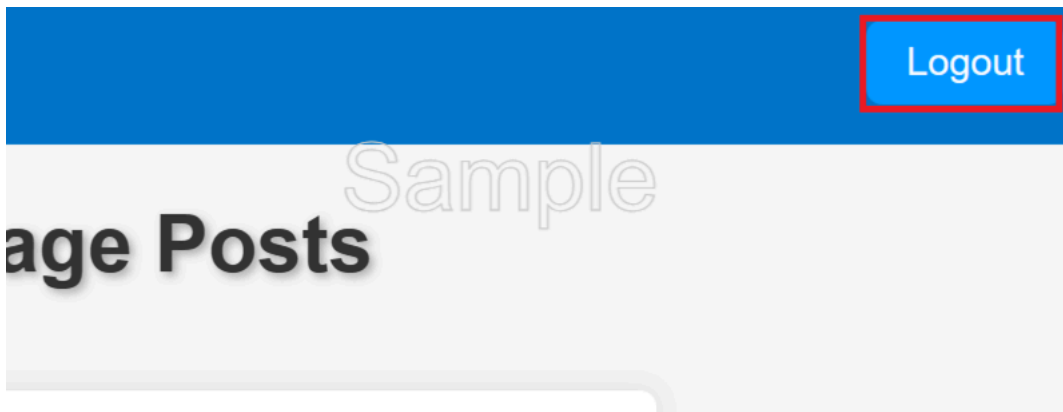


Task 9: Observe the pagination feature by adding three posts and confirming that the fourth post is displayed on a new page. Take a screenshot and save it as pagination.png.



Task 10: Click on the logout button and ensure successful logout from the application. Take a screenshot and save it as Logout.png.





## Conclusion

Congratulations! You have successfully finished this Final Project lab and acquired the skills to develop a Social Media dashboard application.

In this lab, you learned how to:

- Create a Social Media Dashboard web app with user authentication, JWT-based security, and post management.
- Develop a web application with Node.js and Express on the backend, and HTML/CSS/JS on the frontend. Include features like user registration, login, post creation, and pagination.

## Authors

Pallavi Rai

© IBM Corporation. All rights reserved.