IMPLEMENTED CODE

1.LOGIN

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import styled from 'styled-components';

const Login = ({ setIsAuthenticated }) => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const navigate = useNavigate();

  // Handle form submission for login
  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(''); // Reset error message

    // Ensure that email and password are not empty
    if (!email || !password) {
      setError('Email and password are required!');
      return;
    }

    try {
      const response = await fetch('http://localhost:5000/api/auth/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
        },
```

```
      body: JSON.stringify({ email, password }),
    });

    const result = await response.json();

    if (response.status === 200) {
      // Store the JWT token in localStorage
      localStorage.setItem('authToken', result.token);
      localStorage.setItem('token', result.token);

      // Update parent state (App.js)
      setIsAuthenticated(true);

      alert('Login successful!');
      navigate('/dashboard'); // Redirect to dashboard on success
    } else {
      setError(result.message || 'Invalid email or password');
    }
  } catch (err) {
    console.error('Error during login:', err);
    setError('Failed to login. Please try again.');
  }
};

return (
  <Container>
    <Card>
      <Title>Login</Title>
      {error && <ErrorMessage>{error}</ErrorMessage>} {/* Display error message */}
      <Form onSubmit={handleSubmit}>
        <InputContainer>
```

```
          <Label htmlFor="email">Email</Label>
          <Input
            type="email"
            id="email"
            placeholder="Email"
            value={email}
            onChange={(e) => setEmail(e.target.value)}
            required
          />
        </InputContainer>
        <InputContainer>
          <Label htmlFor="password">Password</Label>
          <Input
            type="password"
            id="password"
            placeholder="Password"
            value={password}
            onChange={(e) => setPassword(e.target.value)}
            required
          />
        </InputContainer>
        <Button type="submit">Login</Button>
      </Form>
      <TextCenter>
        Don't have an account? <StyledLink href="/">Register here</StyledLink>
      </TextCenter>
    </Card>
  </Container>
 );
};
```

```
// Styled components (same as before)
const Container = styled.div`
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
  background-color: #f4f7fa;
`;

const Card = styled.div`
  background-color: #fff;
  padding: 20px;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  width: 100%;
  max-width: 400px;
`;

const Title = styled.h2`
  text-align: center;
  color: #007bff;
  font-weight: bold;
`;

const ErrorMessage = styled.div`
  color: #ff4d4d;
  font-size: 14px;
  margin-bottom: 15px;
  text-align: center;
`;
```

```
const Form = styled.form`
  display: flex;
  flex-direction: column;
`;

const InputContainer = styled.div`
  margin-bottom: 15px;
`;

const Label = styled.label`
  display: block;
  margin-bottom: 5px;
  color: #333;
`;

const Input = styled.input`
  width: 100%;
  padding: 10px;
  border-radius: 4px;
  border: 1px solid #ddd;
  font-size: 14px;
  color: #333;
  margin-top: 5px;

  &:focus {
    border-color: #007bff;
  }
`;

const Button = styled.button`
  width: 100%;
```

```
  padding: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;

  &:hover {
    background-color: darkblue;
  }
`;

const TextCenter = styled.p`
  text-align: center;
  margin-top: 15px;
`;

const StyledLink = styled.a`
  color: #007bff;
  text-decoration: none;

  &:hover {
    color: #ff4d4d;
  }
`;

export default Login;
```

2.REGISTER

```
import { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import styled from 'styled-components'; // Import styled-components

const Register = () => {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [error, setError] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);
  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError('');
    setIsSubmitting(true);

    // Basic password validation
    const passwordRegex = /^(?=.\d)(?=.[a-z])(?=.*[A-Z]).{6,20}$/;
    if (!passwordRegex.test(password)) {
      setError('Password must be between 6-20 characters and contain at least one number, one lowercase, and one uppercase letter.');
      setIsSubmitting(false);
      return;
    }

    try {
      const response = await fetch('http://localhost:5000/api/auth/register', {
        method: 'POST',
        headers: {
```

```
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ username, email, password }),
    });

    const result = await response.json();

    if (response.status === 201) {
      alert('User registered successfully');
      navigate('/login');
    } else {
      setError(result.message || 'Error during registration');
    }
  } catch (err) {
    console.error('Error during registration:', err);
    setError('Failed to register. Please try again.');
  } finally {
    setIsSubmitting(false);
  }
};

return (
  <Container>
    <Card>
      <Title>Register</Title>
      {error && <ErrorMessage>{error}</ErrorMessage>}
      <Form onSubmit={handleSubmit}>
        <InputContainer>
          <Label htmlFor="username">Username</Label>
          <Input
            type="text"
```

```
    id="username"
    placeholder="Username"
    value={username}
    onChange={(e) => setUsername(e.target.value)}
    required
    aria-label="Username"
   />
  </InputContainer>
  <InputContainer>
   <Label htmlFor="email">Email</Label>
   <Input
    type="email"
    id="email"
    placeholder="Email"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    required
    aria-label="Email"
   />
  </InputContainer>
  <InputContainer>
   <Label htmlFor="password">Password</Label>
   <Input
    type="password"
    id="password"
    placeholder="Password"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    required
    aria-label="Password"
   />
```

```
        </InputContainer>
        <Button type="submit" disabled={isSubmitting}>
          {isSubmitting ? 'Registering...' : 'Register'}
        </Button>
      </Form>
      <TextCenter>
        Already have an account? <StyledLink href="/login">Login here</StyledLink>
      </TextCenter>
    </Card>
  </Container>
 );
};

// Styled components

const Container = styled.div`
 display: flex;
 justify-content: center;
 align-items: center;
 height: 100vh;
 background-color: #f4f7fa;
`;

const Card = styled.div`
 background-color: #fff;
 padding: 20px;
 border-radius: 8px;
 box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
 width: 100%;
 max-width: 400px;
`;
```

```
const Title = styled.h2`
  text-align: center;
  color: #007bff;
  font-weight: bold;
`;

const ErrorMessage = styled.div`
  color: #ff4d4d;
  font-size: 14px;
  margin-bottom: 15px;
  text-align: center;
`;

const Form = styled.form`
  display: flex;
  flex-direction: column;
`;

const InputContainer = styled.div`
  margin-bottom: 15px;
`;

const Label = styled.label`
  display: block;
  margin-bottom: 5px;
  color: #333;
`;

const Input = styled.input`
  width: 100%;
```

```
  padding: 10px;
  border-radius: 4px;
  border: 1px solid #ddd;
  font-size: 14px;
  color: #333;
  margin-top: 5px;

  &:focus {
    border-color: #007bff;
  }

  &:hover {
    border-color: #007bff;
  }
`;

const Button = styled.button`
  width: 100%;
  padding: 10px;
  background-color: #007bff;
  color: #fff;
  border: none;
  border-radius: 4px;
  font-size: 16px;
  cursor: pointer;

  &:hover {
    background-color: darkblue;
  }

  &:disabled {
```

```
    background-color: #cccccc;
    cursor: not-allowed;
  }
`;

const TextCenter = styled.p`
  text-align: center;
  margin-top: 15px;
`;

const StyledLink = styled.a`
  color: #007bff;
  text-decoration: none;

  &:hover {
    color: #ff4d4d;
  }
`;

export default Register;
```

3.DASHBOARD

```
import { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { Helmet } from 'react-helmet';
import styled from 'styled-components';
import { jwtDecode } from 'jwt-decode'; // Import jwt-decode for token expiration check

const Dashboard = () => {
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(false);
  const [loadingTasks, setLoadingTasks] = useState(true);
  const [userName, setUserName] = useState(''); // State for storing the username part of email
  const navigate = useNavigate();

  // Load tasks from the backend
  const loadTasks = async () => {
    setLoadingTasks(true);

    // Retrieve token from localStorage
    const token = localStorage.getItem('authToken');
    console.log('Token retrieved:', token); // Debugging line

    if (!token) {
      // No token found, redirect to login
      console.log('No token found, redirecting to login'); // Debugging line
      navigate('/login');
      return;
    }

    try {
      // Decode token and check if it's expired
```

```
const decoded = jwtDecode(token);
const currentTime = Date.now() / 1000; // Get current time in seconds

if (decoded.exp < currentTime) {
  localStorage.removeItem('authToken');
  console.log('Token expired, redirecting to login');
  navigate('/login'); // Redirect to login if token is expired
  return;
}

// Extract email and split before @ to get the username part
const email = decoded.email;  // Assuming the email is stored in 'email' field in the token
if (email) {
  const username = email.split('@')[0];  // Extract the part before '@'
  setUserName(username); // Store the username in state
}

// If token is valid, proceed to fetch tasks
const response = await fetch('http://localhost:5000/api/tasks', {
  method: 'GET',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': Bearer ${token},
  },
});

if (!response.ok) {
  // If the response is not ok (e.g., 404, 500), throw an error
  throw new Error(Error: ${response.status});
}
```

```
    const result = await response.json(); // Parse JSON response
    setLoadingTasks(false);
    setTasks(result);
  } catch (error) {
    setLoadingTasks(false);
    if (error.message === 'Error: 401') {
      localStorage.removeItem('authToken');
      navigate('/login'); // Redirect to login if token is invalid
    } else {
      alert(Error loading tasks: ${error.message}); // Show the error message
    }
  }
};

// Mark a task as completed
const markAsCompleted = async (taskId) => {
  const token = localStorage.getItem('authToken');
  if (!token) {
    navigate('/login'); // Redirect to login if no token
    return;
  }

  setLoading(true);

  const response = await fetch(http://localhost:5000/api/tasks/complete/${taskId}, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${token},
    },
  });
```

```
  const result = await response.json();
  setLoading(false);

  if (response.status === 200) {
    setTasks(tasks.map((task) =>
      task._id === taskId ? { ...task, completed: true } : task
    ));
  } else {
    alert(result.message || 'Error marking task as completed');
  }
};

// Delete a task from the backend with a confirmation prompt
const deleteTask = async (taskId) => {
  const token = localStorage.getItem('authToken');
  if (!token) {
    navigate('/login'); // Redirect to login if no token
    return;
  }

  const confirmDelete = window.confirm('Are you sure you want to delete this task?');
  if (!confirmDelete) return;

  setLoading(true);

  const response = await fetch(http://localhost:5000/api/tasks/${taskId}, {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${token},
```

```
    },
  });

  const result = await response.json();
  setLoading(false);

  if (response.status === 200) {
    setTasks(tasks.filter((task) => task._id !== taskId));
  } else {
    alert(result.message || 'Error deleting task');
  }
};

// Edit a task (navigate to the edit page)
const editTask = (taskId) => {
  navigate(/edit-task/${taskId});
};

// Fetch tasks when the component is mounted
useEffect(() => {
  loadTasks();
}, []);

// Logout function
const handleLogout = () => {
  localStorage.removeItem('authToken'); // Clear the token from localStorage
  navigate('/login'); // Redirect to login page
};

return (
  <div>
```

```
    <Helmet>
      <style>
        {`
          body {
            background-color: #f5f5f5; /* Override background for Dashboard */
            color: #333;
          }
        `}
      </style>
    </Helmet>

    <Header>
      <h1>Welcome, {userName ? userName : 'User'}!</h1> {/* Display the username part of the email */}
      <LogoutButton onClick={handleLogout}>Logout</LogoutButton>
    </Header>

    <Container>
      <h3>Your Tasks</h3>
      <AddTaskButton to="/add-task">Add New Task</AddTaskButton>

      {/* Email Logs Button */}
      <EmailLogsButton to="/email-logs">View Email Logs</EmailLogsButton>

      {loadingTasks ? (
        <LoadingMessage>Loading tasks...</LoadingMessage>
      ) : (
        <TaskTable>
          <thead>
            <tr>
              <th>Task</th>
```

```
    <th>Due Date</th>
    <th>Reminder</th>
    <th>Action</th>
  </tr>
</thead>
<tbody>
  {tasks.length > 0 ? (
    tasks.map((task) => (
      <tr key={task._id}>
        <td>{task.title}</td>
        <td>{new Date(task.due_date).toLocaleString()}</td>
        <td>{task.reminder ? new Date(task.reminder).toLocaleString() : 'No reminder'}</td>
        <td>
          {task.completed ? (
            <CompletedLabel>Completed</CompletedLabel>
          ) : (
            <CompletedButton onClick={() => markAsCompleted(task._id)}>
              Mark as Completed
            </CompletedButton>
          )}
          <DeleteButton onClick={() => deleteTask(task._id)}>Delete Task</DeleteButton>
          <EditButton onClick={() => editTask(task._id)}>Edit Task</EditButton>
        </td>
      </tr>
    ))
  ) : (
    <tr>
      <td colSpan="4">No tasks available</td>
    </tr>
  )}
</tbody>
```

```
        </TaskTable>
      )}
    </Container>
  </div>
 );
};

const Header = styled.header`
 background-color: #007BFF;
 color: white;
 padding: 20px;
 text-align: center;
`;

const LogoutButton = styled.button`
 background-color: #FF5733;
 color: white;
 padding: 10px 20px;
 border: none;
 cursor: pointer;
 border-radius: 4px;
 margin-left: 20px;
`;

const Container = styled.div`
 max-width: 900px;
 margin: 20px auto;
 padding: 20px;
 background: white;
 border-radius: 8px;
 box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
```

```
`;

const AddTaskButton = styled(Link)`
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  display: inline-block;
  margin-bottom: 20px;
  text-decoration: none;
`;

const EmailLogsButton = styled(Link)`
  background-color: #007BFF;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  display: inline-block;
  margin-bottom: 20px;
  text-decoration: none;
`;

const LoadingMessage = styled.div`
  text-align: center;
  padding: 20px;
  font-size: 16px;
`;
```

```
const TaskTable = styled.table`
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
`;

const CompletedLabel = styled.span`
  color: green;
  font-weight: bold;
`;

const CompletedButton = styled.button`
  background-color: #4CAF50;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
`;

const DeleteButton = styled.button`
  background-color: #FF5733;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  margin-left: 10px;
`;
```

```
const EditButton = styled.button`
  background-color: #007BFF;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  margin-left: 10px;
`;

export default Dashboard;
```

4.ADD TASK

```
import { useEffect, useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { Helmet } from 'react-helmet';
import styled from 'styled-components';
import { jwtDecode } from 'jwt-decode'; // Import jwt-decode for token expiration check

const Dashboard = () => {
  const [tasks, setTasks] = useState([]);
  const [loading, setLoading] = useState(false);
  const [loadingTasks, setLoadingTasks] = useState(true);
  const [userName, setUserName] = useState(''); // State for storing the username part of email
  const navigate = useNavigate();

  // Load tasks from the backend
  const loadTasks = async () => {
    setLoadingTasks(true);

    // Retrieve token from localStorage
    const token = localStorage.getItem('authToken');
    console.log('Token retrieved:', token); // Debugging line

    if (!token) {
      // No token found, redirect to login
      console.log('No token found, redirecting to login'); // Debugging line
      navigate('/login');
      return;
    }

    try {
      // Decode token and check if it's expired
```

```
const decoded = jwtDecode(token);
const currentTime = Date.now() / 1000; // Get current time in seconds

if (decoded.exp < currentTime) {
  localStorage.removeItem('authToken');
  console.log('Token expired, redirecting to login');
  navigate('/login'); // Redirect to login if token is expired
  return;
}

// Extract email and split before @ to get the username part
const email = decoded.email;  // Assuming the email is stored in 'email' field in the token
if (email) {
  const username = email.split('@')[0];  // Extract the part before '@'
  setUserName(username); // Store the username in state
}

// If token is valid, proceed to fetch tasks
const response = await fetch('http://localhost:5000/api/tasks', {
  method: 'GET',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': Bearer ${token},
  },
});

if (!response.ok) {
  // If the response is not ok (e.g., 404, 500), throw an error
  throw new Error(Error: ${response.status});
}
```

```
    const result = await response.json(); // Parse JSON response
    setLoadingTasks(false);
    setTasks(result);
  } catch (error) {
    setLoadingTasks(false);
    if (error.message === 'Error: 401') {
      localStorage.removeItem('authToken');
      navigate('/login'); // Redirect to login if token is invalid
    } else {
      alert(Error loading tasks: ${error.message}); // Show the error message
    }
  }
};

// Mark a task as completed
const markAsCompleted = async (taskId) => {
  const token = localStorage.getItem('authToken');
  if (!token) {
    navigate('/login'); // Redirect to login if no token
    return;
  }

  setLoading(true);

  const response = await fetch(http://localhost:5000/api/tasks/complete/${taskId}, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${token},
    },
  });
```

```
  const result = await response.json();
  setLoading(false);

  if (response.status === 200) {
    setTasks(tasks.map((task) =>
      task._id === taskId ? { ...task, completed: true } : task
    ));
  } else {
    alert(result.message || 'Error marking task as completed');
  }
};

// Delete a task from the backend with a confirmation prompt
const deleteTask = async (taskId) => {
  const token = localStorage.getItem('authToken');
  if (!token) {
    navigate('/login'); // Redirect to login if no token
    return;
  }

  const confirmDelete = window.confirm('Are you sure you want to delete this task?');
  if (!confirmDelete) return;

  setLoading(true);

  const response = await fetch(http://localhost:5000/api/tasks/${taskId}, {
    method: 'DELETE',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': Bearer ${token},
```

```
    },
  });

  const result = await response.json();
  setLoading(false);

  if (response.status === 200) {
    setTasks(tasks.filter((task) => task._id !== taskId));
  } else {
    alert(result.message || 'Error deleting task');
  }
};

// Edit a task (navigate to the edit page)
const editTask = (taskId) => {
  navigate(/edit-task/${taskId});
};

// Fetch tasks when the component is mounted
useEffect(() => {
  loadTasks();
}, []);

// Logout function
const handleLogout = () => {
  localStorage.removeItem('authToken'); // Clear the token from localStorage
  navigate('/login'); // Redirect to login page
};

return (
  <div>
```

```
    <Helmet>
      <style>
        {`
          body {
            background-color: #f5f5f5; /* Override background for Dashboard */
            color: #333;
          }
        `}
      </style>
    </Helmet>

    <Header>
      <h1>Welcome, {userName ? userName : 'User'}!</h1> {/* Display the username part of the email */}
      <LogoutButton onClick={handleLogout}>Logout</LogoutButton>
    </Header>

    <Container>
      <h3>Your Tasks</h3>
      <AddTaskButton to="/add-task">Add New Task</AddTaskButton>

      {/* Email Logs Button */}
      <EmailLogsButton to="/email-logs">View Email Logs</EmailLogsButton>

      {loadingTasks ? (
        <LoadingMessage>Loading tasks...</LoadingMessage>
      ) : (
        <TaskTable>
          <thead>
            <tr>
              <th>Task</th>
```

```
    <th>Due Date</th>
    <th>Reminder</th>
    <th>Action</th>
  </tr>
</thead>
<tbody>
  {tasks.length > 0 ? (
    tasks.map((task) => (
      <tr key={task._id}>
        <td>{task.title}</td>
        <td>{new Date(task.due_date).toLocaleString()}</td>
        <td>{task.reminder ? new Date(task.reminder).toLocaleString() : 'No reminder'}</td>
        <td>
          {task.completed ? (
            <CompletedLabel>Completed</CompletedLabel>
          ) : (
            <CompletedButton onClick={() => markAsCompleted(task._id)}>
              Mark as Completed
            </CompletedButton>
          )}
          <DeleteButton onClick={() => deleteTask(task._id)}>Delete Task</DeleteButton>
          <EditButton onClick={() => editTask(task._id)}>Edit Task</EditButton>
        </td>
      </tr>
    ))
  ) : (
    <tr>
      <td colSpan="4">No tasks available</td>
    </tr>
  )}
</tbody>
```

```
        </TaskTable>
      )}
    </Container>
  </div>
 );
};

const Header = styled.header`
 background-color: #007BFF;
 color: white;
 padding: 20px;
 text-align: center;
`;

const LogoutButton = styled.button`
 background-color: #FF5733;
 color: white;
 padding: 10px 20px;
 border: none;
 cursor: pointer;
 border-radius: 4px;
 margin-left: 20px;
`;

const Container = styled.div`
 max-width: 900px;
 margin: 20px auto;
 padding: 20px;
 background: white;
 border-radius: 8px;
 box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
```

```
`;

const AddTaskButton = styled(Link)`
  background-color: #4CAF50;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  display: inline-block;
  margin-bottom: 20px;
  text-decoration: none;
`;

const EmailLogsButton = styled(Link)`
  background-color: #007BFF;
  color: white;
  padding: 10px 20px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  display: inline-block;
  margin-bottom: 20px;
  text-decoration: none;
`;

const LoadingMessage = styled.div`
  text-align: center;
  padding: 20px;
  font-size: 16px;
`;
```

```
const TaskTable = styled.table`
  width: 100%;
  border-collapse: collapse;
  margin-top: 20px;
`;

const CompletedLabel = styled.span`
  color: green;
  font-weight: bold;
`;

const CompletedButton = styled.button`
  background-color: #4CAF50;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
`;

const DeleteButton = styled.button`
  background-color: #FF5733;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  margin-left: 10px;
`;
```

```
const EditButton = styled.button`
  background-color: #007BFF;
  color: white;
  padding: 5px 10px;
  border: none;
  cursor: pointer;
  border-radius: 4px;
  margin-left: 10px;
`;

export default Dashboard;
```

5.EDIT TASK

```
import { useState, useEffect } from 'react';
import { useNavigate, useParams } from 'react-router-dom';
import styled from 'styled-components';

const EditTask = () => {
  const { taskId } = useParams();  // Get taskId from URL
  const [task, setTask] = useState({
    title: '',
    due_date: '',
    reminder: '',
    completed: false,  // Added 'completed' state
  });
  const [error, setError] = useState('');
  const [loading, setLoading] = useState(true);
  const navigate = useNavigate();

  // Fetch task data when the component is mounted
  useEffect(() => {
    const loadTask = async () => {
      const token = localStorage.getItem('authToken');
      if (!token) {
        window.location.href = '/login';  // Redirect to login if no token
        return;
      }

      try {
        const response = await fetch(http://localhost:5000/api/tasks/${taskId}, {
          method: 'GET',
          headers: {
            'Content-Type': 'application/json',
```

```
          'Authorization': Bearer ${token},  // Pass token for authorization
        },
      });

      if (!response.ok) {
        throw new Error('Task not found');
      }

      const result = await response.json();
      setTask(result);  // Set the task data to state
      setLoading(false);
    } catch (error) {
      setError('Error fetching task: ' + error.message);
      setLoading(false);
    }
  };

  loadTask();
}, [taskId]);

// Handle form field changes
const handleChange = (e) => {
  const { name, value, type, checked } = e.target;
  if (type === 'checkbox') {
    setTask((prevTask) => ({ ...prevTask, [name]: checked }));
  } else {
    setTask((prevTask) => ({ ...prevTask, [name]: value }));
  }
};

// Handle task update
```

```
const handleUpdate = async (e) => {
 e.preventDefault();

 const token = localStorage.getItem('authToken');
 if (!token) {
  window.location.href = '/login';  // Redirect to login if no token
  return;
 }

 setLoading(true);

 try {
  const response = await fetch(http://localhost:5000/api/tasks/${taskId}, {
   method: 'PUT',
   headers: {
    'Content-Type': 'application/json',
    'Authorization': Bearer ${token},  // Pass token for authorization
   },
   body: JSON.stringify(task),  // Send updated task data
  });

  const result = await response.json();

  if (response.status === 200) {
   alert('Task updated successfully');
   navigate('/dashboard');  // Redirect to dashboard after updating
  } else {
   setError(result.message || 'Error updating task');
  }
 } catch (error) {
  setError('Error updating task: ' + error.message);
```

```
  } finally {
    setLoading(false);
  }
};

if (loading) return <div>Loading...</div>;  // Loading state

return (
  <Container>
    <Heading>Edit Task</Heading>
    {error && <Error>{error}</Error>}
    <Form onSubmit={handleUpdate}>
      <InputContainer>
        <Label htmlFor="title">Task Title</Label>
        <Input
          type="text"
          id="title"
          name="title"
          placeholder="Enter task title"
          value={task.title}
          onChange={handleChange}
          required
        />
      </InputContainer>

      <InputContainer>
        <Label htmlFor="due_date">Due Date</Label>
        <Input
          type="datetime-local"
          id="due_date"
          name="due_date"
```

```
    value={task.due_date ? task.due_date.slice(0, 16) : ''} // Format date for input
    onChange={handleChange}
    required
   />
  </InputContainer>

  <InputContainer>
   <Label htmlFor="reminder">Reminder</Label>
   <Input
    type="datetime-local"
    id="reminder"
    name="reminder"
    value={task.reminder ? task.reminder.slice(0, 16) : ''}  // Format date for input
    onChange={handleChange}
   />
  </InputContainer>

  {/* Completed Checkbox */}
  <InputContainer>
   <Label htmlFor="completed">Completed</Label>
   <Checkbox
    type="checkbox"
    id="completed"
    name="completed"
    checked={task.completed}
    onChange={handleChange}
   />
  </InputContainer>

  <Button type="submit" disabled={loading}>Update Task</Button>
 </Form>
```

```
    </Container>
  );
};

// Styled Components
const Container = styled.div`
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  background: #f9f9f9;
  border-radius: 8px;
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
`;

const Heading = styled.h1`
  text-align: center;
  margin-bottom: 20px;
`;

const Form = styled.form`
  display: flex;
  flex-direction: column;
`;

const InputContainer = styled.div`
  margin-bottom: 15px;
`;

const Label = styled.label`
  font-weight: bold;
  margin-bottom: 5px;
```

```
  display: block;
`;

const Input = styled.input`
  width: 100%;
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ddd;
`;

const Checkbox = styled.input`
  width: auto;
  margin-top: 5px;
  margin-left: 10px;
`;

const Button = styled.button`
  width: 100%;
  padding: 10px;
  background-color: #4caf50;
  color: white;
  border: none;
  border-radius: 4px;
  cursor: pointer;
  font-size: 16px;
`;

const Error = styled.div`
  color: red;
  margin-bottom: 15px;
```

```
  font-weight: bold;
  text-align: center;
`;

export default EditTask;
```

6.EMAIL LOGS

```
import { useEffect, useState } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';  // Import useNavigate

const EmailLogs = () => {
  const [emailLogs, setEmailLogs] = useState([]);  // State to hold email logs
  const [loading, setLoading] = useState(true);    // State for loading
  const [error, setError] = useState('');          // State for errors
  const navigate = useNavigate();  // Hook to navigate programmatically

  useEffect(() => {
    const fetchEmailLogs = async () => {
      const token = localStorage.getItem('authToken');  // Get the token from localStorage

      // If no token, redirect to login page
      if (!token) {
        navigate('/login');
        return;
      }

      try {
        const response = await axios.get('http://localhost:5000/api/email-logs', {
          headers: {
            Authorization: Bearer ${token},  // Send token in Authorization header
          },
        });

        setEmailLogs(response.data);  // Set email logs state
        setLoading(false);  // Set loading to false when data is fetched
      } catch (error) {
```

```
      setLoading(false);  // Set loading to false in case of error
      if (error.response && error.response.status === 401) {
        // If token is invalid/expired, redirect to login
        localStorage.removeItem('authToken');  // Optional: Clear invalid token from localStorage
        navigate('/login');
      } else {
        setError('Error fetching email logs. Please try again later.');
      }
    }
  };

  fetchEmailLogs();
}, [navigate]);  // Add navigate as a dependency

return (
  <div>
    <h2>Email Logs</h2>

    {loading ? (
      <div>Loading...</div>  // Show loading message or spinner
    ) : error ? (
      <div style={{ color: 'red' }}>{error}</div>  // Display error message if fetching fails
    ) : (
      <div>
        {emailLogs.length === 0 ? (
          <p>No email logs found.</p>  // Message when no logs are found
        ) : (
          <ul>
            {emailLogs.map((log) => (
              <li key={log._id}>
                <strong>{log.subject}</strong> - {log.to} - {new Date(log.sentAt).toLocaleString()}
```

```
              </li>
            ))}
          </ul>
        )}
      </div>
    )}
  </div>
 );
};

export default EmailLogs;
```

7.EMAIL SENDING CODE

```
const nodemailer = require('nodemailer');

// Configure the email transporter (this example uses Gmail's SMTP server)
const transporter = nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'studyplanner575@gmail.com',  // Replace with your email address
    pass: 'qdxl wkbq gvmh kalc',  // Replace with your email password or app-specific password
  },
  tls: {
    rejectUnauthorized: false  // Disable certificate validation (only for development)
  }
});

// Function to send email
const sendEmail = async (to, subject, text) => {
  const mailOptions = {
    from: 'studyplanner575@gmail.com',  // Sender email
    to: to,                        // Recipient email
    subject: subject,              // Email subject
    text: text,                    // Email body
  };

  try {
    const info = await transporter.sendMail(mailOptions);
    console.log('Email sent: ' + info.response);
  } catch (error) {
    console.error('Error sending email: ', error);
  }
};
```

```
module.exports = sendEmail;
```