Login Page

Start.jsx:

The start component serves as the landing page for the application. It handles clearing local storage, session storage, and cookies, and provides navigation options for logging in as an employee or admin.

Clearing data and logging out:

- useEffect: Ensures that data clearing and logout requests are executed once when the component mounts.
- Clearing storage: Clears local storage and session storage to remove any sensitive data.
- Removing cookies: Removes all cookies by setting their expiry data to a past data.
- Logout request: Sends a logout request to the server to ensure the user is logged out from the backend as well.

Navigation handler:

- handleNavigation: A function to handle navigation to different routes using navigate.

Key Considerations:

- User experience: Clears any existing session data to ensure a fresh start for the user.
- Security: Ensures sensitive data is removed from local storage, session storage, and cookies.
- Navigation: Provides clear options for logging in as an employee or admin.

Software Engineering principles:

- Separation of concerns: Data clearing and logout logic are separated from navigation logic.
- Maintainability: The component is straightforward and easy to maintain.
- Security: Properly clears sensitive data to ensure user privacy.

Logic.jsx:

The Login component handles the admin login process.

State management:

- UseState: Used to manage form input values, error messages and success messages.

Handle Form Submission:

- HandSubmit: Handles the form submission, sends a login request to the server, and manages navigation on success.
- Axios.post: Sends a POST request to the server with the login credentials.
- Conditional navigation: Redirects to the dashboard on successful login, and displays appropriate messages on success or failure.

Render Method:

- Bootstrap classes: Used for styling and layout.
- Form: Contains fields for email and password, and a submit button.

Key Considerations:

- User Feedback: Provides clear success and error messages.
- Form Handling: Manages form state and submission effectively.
- Navigation: Redirects users to either the employee dashboard or admin dashboard on successful login.

Software Engineering Principles:

- User Experience: Ensures users receive immediate feedback on their actions.
- Security: Handles login securely with proper error handling.
- Separation of concerns: Separates state management, form handling, and navigation login.

EmployeeLogin.jsx:

The EmployeeLogin component handles the employee login process.

State management:

- useState: Used to manage form input values and error messages.
- Axios.defaults.withCredentials: Ensures that credentials are included with requests.

Handle Form Submission:

- HandleSubmit: Handles the form submission, sends a login request to the server, and manages navigation on success.
- Axios.post: Sends a POST request to the server with the login credentials.
- Conditional navigation: Redirects to the employee detail page on successful login, and displays appropriate messages on failure.

Render Method:

- Bootstrap classes: Used for styling and layout.
- Form: Contains fields for email and password, and a submit button.

Key Considerations:

- User Feedback: Provides clear error messages.
- Form Handling: Manages form state and submission effectively.
- Navigation: Redirects users to the appropriate page on successful login.

Software Engineering Principles:

- User Experience: Ensures users receive immediate feedback on their actions.
- Security: Handles login securely with proper error handling.
- Separation of Concerns: Separates state management, form handling, and navigation login.

Dashboard:

AddAdmin.jsx

The AddAdmin.jsx component is a React form based component designed to handle the addition of new admin users to the system. It interacts with a backend API to submit new admin details and provides feedback to the user upon success or failure. The component is structured to follow best practices in software engineering, ensuring maintainability, security, and usability.

State Management:

- useState: Using the useState hook allows for the management of form data within a functional component. This approach is preferred in modern React applications for its simplicity and ease of use compared to class based components.
- Form Data Initialization: The initial state includes email. Password, and a default role set to admin. This ensures that the form is pre configured for adding an admin without requiring additional user input for the role.

Form Handling:

- Dynamic State Uploads: This approach ensures that any form field changes are immediately reflected in the component state. It enhances user experience by providing real time feedback and allows for flexible form structures without hardcoding field names.
- Form Submission: The use of async/await ensures that the form submission is handle asynchronously, allowing for non-blocking operations and improved user experience. Error handling is integrated to provide feedback in case of failures.

API Interaction:

- Axios is a HTTP client for making requests to the backend. It offers a clean API, supports Promises, and is easy to integrate with React. Using Axios over other methods like fetch provides better error handling and response manipulation.

Component Structure:

- The form is structured to provide a user friendly experience with clearly labeled fields and necessary validation (e.g., required fields). This structure adheres to standard form design principles, ensuring usability and accessibility.

Software Engineering Guidelines:

- Separation of concerns: The component is focused on a single responsibility - adding an admin. This makes the component easier to maintain and test.
- Reusability: The onAdminAdded callback allows for integration with parent components, making the AddAdmin component reusable in different contexts.
- Error handling: Proper error handling ensures that admins are informed of any issues during the signup process, improving reliability and user trust.

Dashboard.jsx

The Dashboard component serves as the main layout for the admin dashboard. It includes a sidebar with navigation links and handles user authentication and authorization.

Imports:

```
import React, { useEffect } from 'react';
import 'bootstrap-icons/font/bootstrap-icons.css';
import { Link, Outlet, useNavigate } from 'react-router-dom';
import axios from 'axios';
import Performance from './Performance';
```

- Link, Outlet, UseNavigate: Components and hooks from react-router-dom for navigation and rendering nested routes

Navigation and Authentication:

```
const navigate = useNavigate();
axios.defaults.withCredentials = true;
```

- useNavigate: This hook is used for programmatic navigation.
- axios.defaults.withCredentials: Ensures that cross-site-Access-Control requests are made using credentials such as cookies, authorization headers, or TLS client certificates.

Authentication Check:

```
useEffect(() => {
  axios.get('http://localhost:8081/dashboard', { withCredentials: true })
    .then(res => {
      if (res.data.Status === 'Success') {
        if (res.data.role === 'admin') {
          console.log('Admin Dashboard accessed successfully');
        } else {
          const id = res.data.id;
          navigate('/employeedetail/' + id);
        }
      } else {
        navigate('/dashboard');
      }
    })
    .catch(err => {
      console.error(err);
      navigate('/dashboard');
    });
}, [navigate]);
```

- useEffectL Ensures the authentication check runs after the component mounts.
- Axios.get: Sends a GET request to check the authentication status.
- Conditional navigation:

- If the user is an admin, they stay on the dashboard.
- If the user is not an admin, they are redirected to their employee detail page
- If authentication fails, the user is redirected to the login page.

Logout Handler:

```
const handleLogout = () => {
  axios.get('http://localhost:8081/logout', { withCredentials: true })
    .then(res => {
      console.log('Logged out');
      navigate('/start');
    }).catch(err => console.log(err));
};
```

- handleLogout: A function to handle user logout.
- Axios.get: Sends a GET request to log the user out.
- navigate(‘/start’): redirects the user to the start page after logout.

Layout and Styling:

- Sidebar: The sidebar contains navigation links for different sections of the dashboard. Each link uses ‘Link’ from ‘react-router-dom’ to enable client side routing.
- Logout button: The logout button is styled and placed within the sidebar, ensuring it is easily accessible.

Key Considerations:

- Authentication and Authorization: The ‘useEffect’ hook ensures that only authenticated users can access the dashboard. If the user is not authenticated or is not an admin, they are redirected appropriately.
- State Management: While this component does not manage a complex state, it handles navigation and authentication checks efficiently.
- User Experience: The layout is user friendly with a sidebar for easy navigation and a clean, responsive design.
- Security: Using ‘axios.defaults.withCredentials’ ensures that cookies are sent with requests, which is important for maintaining sessions securely.

Software Engineering Principles:

- Separation of Concerns: The component handles the layout and authentication logic separately from other business logic, ensuring a clear separation of concerns.
- Reusability: The sidebar structure and navigation links are designed to be easily extendable if new sections are added in the future.
- Maintainability: The use of hooks and organized structure makes the component easy to maintain and update.
- Scalability: The component is designed to scale, with a sidebar that can accommodate additional links and a layout that can support more content with the Outlet component.

Home.jsx:

The Home component displays the counts of admins and employees, the total salary, and a list of admins. It also includes a form to add new admins via the AddAdmin component.

State Management:

- UseState: This hook is used to manage various pieces of state including admin count, employee count, total salary, and the list of admins.
- Initial states: The initial states for adminCount, employeeCount, and salary are undefined, while admins is initialized as an empty array.

Effect Hook for Fetching Data:

- UseEffect: Ensures that data fetching happens after the component mounts.
- Empty dependency array: Indicated the effect runs only once, when the component mounts.

Data Fetching Function:

- fetchCountsAndAdmins: A function to fetch admin count, employee count, total salary, and the list of admins from the backend.
- Axios.get: Makes GET requests to the backend to fetch the required data.
- State updates: The responses are used to update the state variables (setAdminCount, setEmployeeCount, setSalary, and setAdmins).

Layout and Styling:

- Data display: Admin count, employee count, and tidal salary are displayed in separate cards for clarity.
- Table for admins: The list of admins is displayed in a table format for easy readability.
- AddAdmin component: Integrated to allow adding new admins and updating the admin list dynamically.

Key Considerations:

- State Management: Multiple pieces of state are managed using useState ensuring that each piece of data has its own state.
- Data fetching: Data fetching is centralized in the fetchCountsAndAdmins function, making it easy to manage and reuse when needed (e.g. after adding a new admin).
- Component Integration: The AddAdmin component is integrated to allow dynamic updates to the list of admins.
- User Experience: The layout is user friendly with clear sections for admin count, employee count, salary, and list of admins. The use of Bootstrap ensures a clean and response design.

Software Engineering Principles

- Separation of Concerns: Data fetching logic is separated into the fetchCountsAndAdmins function, and state management is handled through React hooks.
- Reusability: The fetchCountsAndAdmins function and the AddAdmin component can be reused and extended easily.

- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with sections that can accommodate additional data points if needed.

Employee.jsx

The Employee component display a list of employees and allows the admin to add, edit, or delete employees.

StateManagement:
- useState: Manages the state of the employee data and the loading status.
- Data: Stores the list of employees.
- Loading: Indicates whether the data is still being loaded.

Fetching Employee Data:
- useEffect: Fetches the employee data when the component mounts.
- Axios.get: Sends a GET request to the server to fetch the employee data.
- setData: Updates the state with the fetched data.
- setLoading: Sets the loading state to false once the data is fetched.

Handle Employee Deletion:
- handleDelete: Handles the deletion of an employee.
- Axios.delete: Sends a Delete Request to the server to delete the employee.
- window.location.reload(true): Reloads the page to refresh the employee list.

Conditional Rendering:
- Loading state: Displays a loading message while the data is being fetched.

Render Method:
- Bootstrap classes: Used for styling and layout.
- Table: Displays the list of employees with options to edit or delete each employee.

Key Considerations:
- State Management: Manages the state of the employee data and loading status effectively using useState.
- Data Fetching: Retrieves employee data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching or deletion.

- User Experience: Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:

- **Separation of Concerns**: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- **Reusability**: The useEffect hook and axios request can be reused in other components for similar data fetching operations.
- **Maintainability**: The component structure is clear and organized, making it easy to maintain and update.
- **Scalability**: The component is designed to scale, with the ability to add more employee actions or functionalities if needed.

Why This Approach

- **useState for State Management**: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- **useEffect for Data Fetching**: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- **axios for HTTP Requests**: A popular and well-supported HTTP client, axios simplifies making requests and handling responses.
- **Bootstrap for Styling**: Provides a responsive and modern UI without needing to write extensive custom CSS.
- **Conditional Rendering**: Enhances user experience by providing feedback during the data fetching process.

AddEmployee.jsx:

The Add Employee component is a functional component that allows an admin to add a new employee by filling out a form. Upon submission, it sends the data to a backend server using axios.

Imports:

- Axios: A HTTP client for making requests to the server.
- UseState: A react hook for managing state within the component.
- UseNavigate: A hook from react-router-dom for navigation after form submission.

State Management:

```
const [data, setData] = useState({
   name: '',
   email: '',
   password: '',
   address: '',
```

```
  salary: '',
  image: ''
});
```

- UseState: This hook is used to manage the form data. Each field in the form has a corresponding state property.
- Object State: Using an object to manage the state of all form fields ensures that the state is organized and updates are straightforward with a single setData call.

Navigation:

```
const navigate = useNavigate();
```

- UseNavigate: This hook is used for programmatic navigation. After a successful form submission, it redirects the admin to the admin dashboard.

Handle Form Submission:

```
const handleSubmit = (event) => {
  event.preventDefault();
  const formdata = new FormData();
  formdata.append("name", data.name);
  formdata.append("email", data.email);
  formdata.append("password", data.password);
  formdata.append("address", data.address);
  formdata.append("salary", data.salary);
  formdata.append("image", data.image);
  axios.post('http://localhost:8081/create', formdata)
  .then(res => {
    navigate('/dashboard/employee');
  })
  .catch(err => console.log(err));
};
```

- event.preventDefault(): Prevents the default form submission behavior, allowing for custom handling.
- FormData, An instance of FormData is used to handle form submissions, especially when dealing with file uploads.
- Axios.post: Sends a POST request to the backend to create a new employee. The endpoint http://localhost:8081/create.
- Navigate on success: On successful form submission, the user is navigated back to the Admin Dashboard.

Form Layout and Fields:

- Bootstrap classes: This form and its elements use Bootstrap classes for styling. This ensures a responsive and modern UI without writing custom CSS.
- Labels and Inputs: Each input field has a corresponding label, ensuring accessibility and clarity.

- onChange handlers: Each input field updates the state using the setData function. This ensures that the component state is always in sync with the form inputs.

Key Considerations:

- State Management: The state is managed using a single useState hook with an object. This approach simplifies state updates and ensures all form fields are kept together.
- Form Handling: Using FormData for handling the form submission, particularly to handle file uploads, follows best practices for multipart form data.
- Error Handling: The catch block in the axios.post method ensures that any errors during the submission are logged. This can be extended to provide user feedback.
- Navigation: Using useNavigate for programmatic navigation enhances user experience by redirecting users after successful operations.
- Component Structure: The component is well structured and easy to understand. Each part of the form is clearly defined, making the code maintainable and extensible.

Software Engineering principles:

- Separation of Concerns: The component strictly handles the form logic and submission, keeping concerns separated.
- Reusability: The form structure is reusable. If needed, it can be refactored into smaller components for even more modularity.
- Maintainability: The use of hooks and organized state management makes the component easy to maintain and update.
- User Experience: Using Bootstrap for styling ensures a consistent and responsive design, enhancing the user experience.

EditEmployee.jsx:

The EditEmployee component allows an admin to edit an existing employee's details. The component fetches the current details of the employee, displays them in a form, and allows the admin to update them.

State Management:

- UseState: Manages the state of the form data. The initial state is an object with empty strings for name, email, address, and salary.

Navigation and URL Parameters:

- useNavigate: Used for Programmatic navigation.
- useParams: Retrieves the id parameter from the URL, which is used to fetch and update the specific employee's details.

Fetching Employee Data:

- useEffect: Fetches the employee's current data when the component mounts.
- Axios.get: Makes a GET request to fetch the employee's details based on the id.
- SetData. Updates the state with the fetched data.

Handle Form Submission:

- HandleSubmit: Handles the form submission, sends a PUT request to update the employee's details, and navigates to the employee dashboard on success.
- event.preventDefault(): Prevents the default form submission behavior.
- Axios.put: Sends a PUT request to update the employee's details on the server.

Render Method:
- Bootstrap classes: Used for styling and layout, ensuring a responsive and modern UI.
- Form fields: Rendered with current values and onChange handlers to update the state.

Key Considerations:
- State Management: Manages the form state effectively using useState.
- Data Fetching: Retrieves current employee data when the component mounts, ensuring the form is prefilled with existing values.
- Form Handling: Submits the form data to update the employee's details.
- UserExperience: Provides immediate feedback by updating the form fields with the current values and navigating to the admin dashboard or employee dashboard on successful update.

Software Engineering Principles:
- Separation of Concerns: Data fetching, state management, and form handling are separated into appropriate hooks and functions.
- Reusability: The useEffect hook and handleSubmit function are designed to be reusable for similar components.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more fields or functionalities if needed.

Why this approach?
- UseState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- UseEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Axios for HTTP Requests: A popular and well supported HTTP client, axios simplifies making requests and handling responses.
- useNavigate for Navigation: Facilitates programmatic navigation, improving user experience by redirecting after successful operations.
- Bootstrap for Styling: Provides a responsive and modern UI without needing to write extensive custom CSS.

Profile.jsx:

The Profile component fetches and displays the profile information of the logged in user/

State Management:
- UseState: Manages the state of the profile data and the loading status.

- ProfileData: Stores the fetched profile data.
- Loading: Indicates whether the data is still being loaded

Fetching Profile Data:
- UseEffect: Fetches the profile data when the component mounts.
- Axios.get: Sends a GET request to the server to fetch the profile data.
- setProfileData: Updates the state with the fetched profile data.
- setLoading: Sets the loading state to false once the data is fetched/

Conditional Rendering:
- Loading state: Displays a loading message while the data is being fetched/
- No data: Displays a message if no profile data is found.

Render Method:
- Bootstrap classes: Used for styling and layout, ensuring a responsive modern UI.
- Profile Data: Displays the email and role of the user.

Key Considerations:
- State Management: Manages the state of the profile data and the loading status effectively using useState.
- Data Fetching: Retrieves profile data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:
- Separation of Concerns: Data fetching and state management are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios request can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more profile fields or functionalities if needed.

Why This Approach:
- useState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures the data fetching logic run

AdminMessages.jsx:

- The AdminMessages component handles messaging between an admin and employees. It allows the admin to select an employee, view messages, and send new messages.

State Management:

- **useState**: Manages the state of the admin ID, messages, new message content, employees, selected employee, and error messages.

Fetching Profile and Employee Data:

- **useEffect**: Fetches the admin profile and employee data when the component mounts or when selectedEmployee changes.
- **axios.get**: Sends GET requests to fetch the profile and employee data.
- **setAdminId, setEmployees**: Updates the state with the fetched data.
- **localStorage**: Stores the selected employee ID in local storage to persist the selection across sessions.

Fetching Messages:

- **useEffect**: Fetches messages whenever selectedEmployee or adminId changes.
- **axios.get**: Sends a GET request to fetch messages between the admin and the selected employee.
- **setMessages**: Updates the state with the fetched messages.

Handle Send Message:

- **handleSendMessage**: Handles sending a new message.
- **axios.post**: Sends a POST request to send a new message.
- **setMessages**: Updates the state with the new message.

Handle Employee Change:

- **handleEmployeeChange**: Handles changing the selected employee.
- **setSelectedEmployee**: Updates the state and local storage with the new selected employee ID.
- **fetchMessages**: Fetches messages for the new selected employee.

Render Method:

- **Bootstrap classes**: Used for styling and layout.
- **Select dropdown**: Allows selecting an employee to view messages.
- **Messages list**: Displays the list of messages between the admin and the selected employee.
- **Textarea and button**: Provides a form to send new messages.

Key Considerations:

- **State Management**: Manages the state of the admin ID, messages, new message content, employees, selected employee, and error messages effectively using useState.
- **DataFetching:** Retrieves profile, employee, and message data when the component mounts, ensuring the component is always up-to-date with the latest data.
- **Error Handling:** Provides user feedback in case of errors during data fetching or sending messages.

- **User Experience:** Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:

- **Separation of Concerns:** Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- **Reusability:** The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- **Maintainability:** The component structure is clear and organized, making it easy to maintain and update.
- **Scalability:** The component is designed to scale, with the ability to add more message actions or functionalities if needed.

Why This Approach:

- **useState for State Management**: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- **useEffect for Data Fetching**: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- **axios for HTTP Requests**: A popular and well-supported HTTP client, axios simplifies making requests and handling responses.
- **Bootstrap for Styling**: Provides a responsive and modern UI without needing to write extensive custom CSS.
- **Conditional Rendering**: Enhances user experience by providing feedback during the data fetching process.

AdminCalendar.jsx:

The AdminCalendar component displays a calendar with events and allows the admin to add new events. It also shows event details in a modal.

Imports:

- **Calendar, momentLocalizer**: Components from react-big-calendar for displaying a calendar.
- **moment**: A library for handling dates and times.
- **Modal**: A component for displaying modal dialogs.
- **AdminCalendar.css**: Custom CSS for styling the component.

State Management:

- **useState**: Manages the state of the events, new event details, selected event, and admin ID.

Fetching Profile and Events Data:

- UseEffect: Fetches the admin profile and events data when the component mounts.
- Axios.get: Sends GET requests to fetch the profile and events data.
- setAdminId, setEvents: Updates the state with the fetched data.

Handle Add Event:

- handleAddEvent: Handles Adding a new event:
- Axios.post: Sends a POST request to add a new event.
- setNewEvent: Resets the new event form fields after adding the event.

Handle Event Select:

- HandleEventSelect: Hand;es selecting an event to display its details.
- Axios.get: Sends a GET request to fetch the email of the event creator.
- SetSelectedEvent: Updates the state with the selected event details.

Close Modal:

- closeModal: Closes the modal by setting selectedEvent to null.

Render Method:

- Bootstrap classes: Used for styling and layout.
- Event form: Provides input fields to add new events.
- Calendar: Displays events on a calendar.
- Modal: Displays selected event details in a modal.

Key Considerations:

- State Management: Manages the state of the events, new event details, selected event, and admin ID effectively using useState.
- Data Fetching: Retrieves profile and event data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching or adding events.
- User Experience. Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:

- Separation of Concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability. The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more event actions or functionalities if needed.

Why This Approach:

- UseState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.

- Axios for HTTP Requests: A popular and well supported HTTP client, axios simplifies making requests and handling responses.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.

Performance.jsx:

The performance component displays and manages the performance data of employees. It allows adding new performance entries and viewing existing performance records.

State Management:

- UseState: Manages the state of performance data, loading status, error messages, form data, and employee list.

Fetching Data:

- UseEffect: Fetches performance data and employee list when the component mounts.
- Axios.get: Sends GET requests to fetch the performance data and employee list.
- setPerformanceData, setEmployees: Updates the state with the fetched data.
- setLoading, setError: Manages the loading and error states.

Handle Form Change and Submission:

- handleChange: Updates the form data state when form inputs change.
- handleSubmit: Handles form submission to add new performance data.
- Axios.post: Sends a POST request to add new performance data.
- fetchPerformanceData: Refreshes the performance data after adding a new entry.
- setFormData: Resets the form data after successful submission.

Conditional Rendering:

- Loading state: Displays a loading message while the data is being fetched.
- Error state: Displays an error message if there is an error fetching data.

Render Method:

- Form: Allows adding new performance data.
- Table: Displays existing performance data.

Key Considerations:

- State Management: Manages the state of performance data, loading status, error messages, form data, and employee list effectively using useState.
- Data Fetching: Retrieves performance and employee data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching or adding performance data.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:

- Separation of Concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and the axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more performance metrics or functionalities if needed.

Why This Approach:

- useState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Axios for HTTP requests: A popular and well supported HTTP client, axios simplifies making requests and handling responses.
- Bootstrap for Styling: Provides a responsive and modern UI without needing to write extensive custom CSS.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.

Reports.jsx:

The Report component displays performance reports of employees and provides an option to send messages to employees.

State Management:

- UseState: Manages the state of performance data, loading status, and error messages.
- UseNavigate: Used for programmatic navigation.

Fetching Performance Data:

- UseEffect: Fetches performance data when the component mounts.
- Axios.get: Sends a GET request to fetch the performance report data.
- setPerformanceData: Updates the state with the fetched data.
- setLoading, setError: Manages the loading and error states.

Handle Message Click:

- HandleMessageClick: Navigates to the messaging component with the selected employee's ID.

Conditional Rendering:

- Loading state: Displays a loading message while the data is being fetched.
- Error state: Displays an error message if there is an error fetching data.

Render Method:

- Table: Displays the performance report data with an option to send messages.

Key Considerations:

- State Management: Manages the state of performance data, loading status, and error messages effectively using useState.
- Data Fetching: Retrieves performance report data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:
- Separation of concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more actions or functionalities if needed.

Why This Approach:
- useState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Axios for HTTP Requests: A popular and well supported HTTP client, axios simplifies making requests and handling responses.
- Bootstrap for styling: Provides a responsive and modern UI without needed to write extensive custom CSS.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.
-

Attendance.jsx:

The Attendance component displays and manages the attendance records of employees. It allows viewing detailed attendance for specific dates and adding new attendance entries.

State Management:
- UseState: Manages the state of attendance data, loading status, error messages, status messages, modal visibility, form data, employee emails, selected data, and selected attendance.

Fetching Data:
- UseEffect: Fetches attendance data and employee emails when the component mounts.
- Axios.get: Sends GET requests to fetch attendance data and employee emails.
- setAttendanceData, setEmployeeEmails: Updates the state with the fetched data.
- setLoading, setError: Manages the loading and error states.

Handle Form Change and Submission:
- handleInputChange: Updates the form data state when form inputs change.
- handleFormSubmit: Handles form submission to add new attendance data.

- Axios.post: Sends a POST request to add new attendance data.
- fetchAttendanceData: Refreshes the attendance data after adding a new entry.
- setFormData: Resets the form data after successful submission.

Handle Date Click:
- handleDateClick: Handles clicking on a date to view detailed attendance.
- setSelectedDate, setSelectedAttendance: Updates the state with the selected date and attendance data.
- setIsDetailModalOpen: Opens the detail modal.

Close Modals:
- closeAddModal, closeDetailModal: Closes the respective modals.

Render Method:
- Modals: Used for adding new attendance entries and viewing detailed attendance.
- Agenda: Displays a list of dates with clickable headers to view attendance details.

Key Considerations:
- State Management: Manifest the state of attendance data, loading status, error messages, status messages, modal visibility, form data, employee emails, selected date, and selected attendance effectively using useState.
- Data Fetching: Retrieves attendance data and employee emails when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching or adding attendance data.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process. Provides modals for a better user interface to add and view attendance data.

Software Engineering principles;
- Separation of Concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more attendance actions or functionalities if needed.

Why This Approach:
- UseState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.

- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.

AdminDocuments.jsx:

The AdminDocuments component manages the sharing and viewing of documents with employees. It allows admins to upload new documents and view existing ones.

State Management:
- UseState: Manages the state of employees, documents, selected employee, modal visibility, form data, and selected document.

Fetching Data:
- UseEffect: Fetches employees and documents data when the component mounts.
- Axios.get: Sends GET requests to fetch the employees and documents data.
- SetEmployees, SetDocuments: Updates the state with the fetched data.

Handle Add and View Document:
- handleAddDocument: Opens the modal to add a new document.
- handleViewDocument: Opens the modal to view the selected document.

Handle Form Change and Submission:
- handleInputChange: Updates the form data state when form inputs change.
- handleFileChange: Updates the form data state when a file is selected.
- handleFormSubmit: Handles form submission to add a new document.
- Axios.post: Sends a POST request to add a new document.
- fetchDocuments: Refreshes the document data after adding a new entry.

Close Modals:
- closeAddModal, closeViewModal: Closes the respective modals.

Render Method:
- Form: Allows adding new documents.
- Table: Displays existing documents with an option to view each document.
- Modals: Used for adding and viewing documents.

Key Considerations:
- State Management: Manages the state of employees, documents, selected employee, modal visibility, form data, and selected document effectively using useState.
- Data Fetching: Retrieves employees and documents data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching or adding documents.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process. Provides modals for a better user interface to add and view documents.

Software Engineering Principles:

- Separation of concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more document actions or functionalities if needed.

Why This Approach:

- UseState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.

Announcements.jsx:

The Announcements component displays and manages announcements. It allows admins to filter announcements, add new ones, and view comments on existing announcements.

State Management:

- UseState: Manages the state of announcements, filters, dates, new announcement data, comments, new comment data, selected announcement, modal visibility, and user email.

Fetching Data:

- UseEffect: Fetches announcements, comments, and user profile data when the component mounts.
- Axios.get: Sends GET requests to fetch the announcements, comments, and user profile data.
- setAnnouncements, setComments, setUserEmail: Updates the state with the fetched data.

Handle Filter and Input Changes:

- handleFilterChange: Updates the filter state when the filter input changes.
- handleInputChange: Updates the new announcement state when form inputs change.

Handle Add Announcement and Comment:

- handleAddAnnouncement: Handles adding a new announcement.
- Axios.post: Sends a POST request to add a new announcement.
- handleAddComment: Handles adding a new comment to an announcement.
- setAnnouncements, setNewAnnouncement, setComments: Updates the state with the new data.

Handle Comment Modal:

- openCommentModal, closeCommentModal: Opens and closes the comment modal.

Filtered Announcements:

- FilteredAnnouncements: Filters announcements based on the selected filer and date range.

Render Method:

- Filter Form: Allows filtering announcements by importance and date.
- Form: Allows adding new announcements.
- Table: Displays existing announcements with options to view comments.
- Modals: Used for viewing and adding comments.

Key Considerations:

- State Management: Manages the state of announcements, filters, dates, new announcement data, comments, new comment data, selected announcement, modal visibility, and user email effectively using useState.
- Data Fetching: Retrieves announcements, comments, and user profile data when the component mounts, ensuring the component is always up to date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching, adding announcements, or adding comments.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process. Provides modals for a better user interface to view and add comments.

Software Engineering Principles:

- Separation of Concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more announcement actions or functionalities if needed.

Why This Approach:

- useState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.

AdminFeedback.jsx:

The AdminFeedback component displays feedback submitted by employees. It fetches and displays the feedback data in a table format.

State Management:

- useState: Manages the state of feedbacks and error messages.

Fetching Feedbacks:

- useEffect: Fetches feedback data when the component mounts.
- setFeedbacks, setError: Updates the state with the fetched data or error message.

Render Method:

- Table: Displays feedback data in a table format.

Key Considerations:

- State Management: Manages the state of feedbacks and error messages effectively using useState.
- Data Fetching: Retrieves feedback data when the component mounts, ensuring the component is always up-to-date with the latest data.
- Error Handling: Provides user feedback in case of errors during data fetching.
- User Experience: Displays loading and error messages to keep the user informed about the data fetching process.

Software Engineering Principles:

- Separation of Concerns: Data fetching, state management, and user interactions are handled separately, making the component more maintainable.
- Reusability: The useEffect hook and axios requests can be reused in other components for similar data fetching operations.
- Maintainability: The component structure is clear and organized, making it easy to maintain and update.
- Scalability: The component is designed to scale, with the ability to add more feedback actions or functionalities if needed.

Why This Approach:

- useState for State Management: Using useState allows for simple and effective state management, which is easy to understand and maintain.
- useEffect for Data Fetching: Ensures that data fetching logic runs only once when the component mounts, avoiding unnecessary re-renders.
- Conditional Rendering: Enhances user experience by providing feedback during the data fetching process.