Front End Code Explaination

Login Page

File: Login.tsx

Overview

The Login.tsx component serves as the login page for the application. It allows users to input their email and password and then attempts to log them in by submitting the credentials to a backend API. Upon successful login, the component navigates users to the dashboard, passing along relevant user information such as userType to display different dashboard versions based on the user type.

Code Breakdown

Imports and Dependencies

1. **React and React Router DOM**:
   - useState, useEffect: Hooks to manage component state and lifecycle events.
   - useNavigate: A hook from react-router-dom to programmatically navigate users after successful login.
2. **Chakra UI Components**:
   - Various UI components such as Box, Button, FormControl, FormLabel, etc., are used to structure the page with consistent styling.
3. **Axios**:
   - Handles API requests to the backend.
4. **Custom CSS**:
   - Login.css is assumed to contain styles specific to this component, such as the gradient background.

Component Structure and Functionality

State Variables

1. **email and password**:
   - Stores the user's input for login credentials.
2. **errorMessage**:
   - Stores any error messages related to login failure, which displays below the login form.
3. **currentTime**:
   - Displays the current date and time on the page, updated every second.

useEffect Hook

- **Purpose**: Sets up an interval to update currentTime every second, providing real-time clock functionality.
- **Cleanup**: Clears the interval upon component unmount to avoid memory leaks.

handleLogin Function

- **Purpose**: Handles form submission when the user attempts to log in.
- **Process**:
  - preventDefault() is called on the event object to prevent the page from reloading on form submission.
  - A POST request is sent to the backend API at http://localhost:5000/api/Students/Login, with emailand password as the payload.
  - **Success Response**: If the response status is 200, the API returns user data, including fullName, userId, and userType. navigate is used to redirect the user to the dashboard, passing these details along in the route’s state.
  - **Error Handling**: If the request fails (e.g., incorrect credentials), an error message is displayed to the user.

Return JSX (HTML Structure)

Layout

The main layout uses a Flex container with Box elements to create a two-panel layout for the login form and an illustration.

Left Side (Box Element)

1. **Welcome Message and Instructions**:
   - A centered Heading and Text provide a greeting and instruction to the user.
2. **Date and Time Display**:
   - Displays the current date and time using the currentTime state, formatted using toLocaleDateString() and toLocaleTimeString().
3. **Form**:
   - **Email Input**: Collects the user’s email with a FormControl element. The onChange handler updates the email state.
   - **Password Input**: Collects the user’s password with similar behavior, updating the password state on changes.
   - **Submit Button**: A Chakra UI Button that triggers handleLogin when clicked.
   - **Error Message**: Conditionally renders a Text component in red if errorMessage has any text, providing feedback for login failures.

Right Side (Box Element)

1. **Illustration Image**:
   - Displays an image related to the login, enhancing visual appeal. The image file is imported from a relative path (./photos/login-image.png).

Code Flow Summary

1. **Initial Render**:
    - useState initializes the variables. useEffect sets up the currentTime update interval, starting the real-time clock.
2. **User Interaction**:
    - User inputs are stored in the email and password states.
    - On form submission, handleLogin sends credentials to the backend. If valid, the user is redirected to the dashboard with their userType and other info; if invalid, an error message appears.
3. **Page Update**:
    - currentTime is updated every second through the useEffect hook, creating a live clock effect.

Dashboard.tsx

Dashboard.tsx

Overview

The Dashboard.tsx component is the main dashboard for both Admin and Student users. It displays different data based on the user's role, such as a list of students, courses, and enrollment details for admins, and active/completed classes and available classes for students. This component interacts with various backend endpoints to retrieve and manage data and has modals for additional user interactions like adding courses or viewing course and student details.

Code Breakdown

Imports and Dependencies

1. **React and Chakra UI Components**:
    - Imports React hooks like useState, useEffect, and various Chakra UI components to structure the page and implement styling and interactivity.
2. **React Router DOM**:
    - useLocation and useNavigate from react-router-dom are used for navigation and retrieving user data passed from the login page.
3. **Axios**:
    - Axios manages HTTP requests to the backend for data retrieval and manipulation.

Interfaces

- **Enrollment**: Defines an enrollment with courseName, completionStatus, and optional progress.

- **Course**: Defines a course with details like courseId, courseName, description, and seat information.
- **Student**: Defines a student with properties like studentId, fullName, and email.

Component Structure and Functionality

State Variables

1. **User and Time Information**:
   - currentTime: Stores the current time and updates every second.
   - fullName, userId, userType: Derived from location.state, representing the logged-in user's information.
2. **Data Management**:
   - **Enrollment and Courses**:
     - enrollments: Stores the list of courses a student is enrolled in.
     - availableCourses: Stores the list of available courses.
     - students: Stores the list of students (admin-specific).
   - **Modals and Course Details**:
     - selectedStudentEnrollments, selectedStudentName: Store specific enrollment details for a selected student in a modal.
     - selectedCourseDetails: Stores details of a selected course.
     - isCourseModalOpen, isStudentsOpen, isCoursesOpen, etc.: Controls modal visibility and collapsible sections.
3. **Form Data**:
   - courseName, description, maxSeats, currentSeats, startDate, endDate, etc.: Hold data for creating a new course.

useEffect Hook for Initial Data Fetching

- Based on userType, this hook:
  - **Admin**: Fetches students and courses.
  - **Student**: Fetches enrollments and courses.

Fetching Logic:

- fetchCourses: Retrieves the list of available courses.
- fetchStudents: Retrieves the list of students.
- fetchEnrollments: Retrieves all enrollments and filters by studentName for student users.

Event Handlers

1. **openAddCourseModal and closeAddCourseModal**:
   - **Purpose**: Toggles the course modal for adding new courses and clears form values.
2. **handleAddCourse**:

   - **Purpose**: Adds a new course by sending a POST request to the backend with course details.
   - **Error Handling**: Displays an error if required fields are missing or if the backend request fails.
   - **Success Handling**: Shows a success message in a modal and refreshes the course list.
3. **handleEnroll**:
   - **Purpose**: Enrolls a student in a course by sending a POST request to the backend.
   - **Success Handling**: Shows an enrollment confirmation message in a modal and refreshes enrollment data for the student.
4. **handleStudentClick**:
   - **Purpose**: Fetches enrollment details for a selected student and displays them in a modal.
5. **handleCourseClick**:
   - **Purpose**: Displays details of a selected course by finding the course in availableCourses and setting selectedCourseDetails.
6. **handleLogout**:
   - **Purpose**: Logs the user out by navigating to the login page.
7. **Toggle Functions** (togglePreviousClasses, toggleStudents, toggleCourses):
   - **Purpose**: Toggle the visibility of sections like students, courses, and previous classes.

Conditional UI Rendering

1. **User Type-based UI**:
   - **Admin (userType === 1)**: Displays a student list, course list, and an “Add New Course” button.
   - **Student (userType === 2)**: Displays completed and active courses, and a table of available classes for enrollment.
2. **Progress Calculation for Student Courses**:
   - Progress is visually represented in colored boxes based on completion status: green for complete, red for <60%, yellow for <90%, and teal for <100%.
3. **Pagination Logic**:
   - **Students and Courses Pagination**: Uses currentPage and currentCoursePage to manage student and course lists' pagination.

Modal Structures

1. **Main Enrollment Status Modal**:
   - Displays a message when a course is added or an enrollment action

completes.

2. **Student Enrollment Details Modal**:
   - Shows all courses a student is enrolled in, with a scrollable carousel-style list to browse enrollments.
3. **Course Details Modal**:
   - Displays details about a selected course, including description, maxSeats, currentSeats, startDate, and endDate.

Code Flow Summary

1. **Initial Render**:
   - The useEffect hook fetches the required data based on the user's role and initializes the real-time clock.
2. **Admin Interactions**:
   - Admins can view students, available courses, and add new courses through a modal.
   - Toggling functions control the visibility of sections, while modals allow detailed views.
3. **Student Interactions**:
   - Students see active and completed courses in a card layout and can enroll in available classes.
   - Enrollments and courses update upon successful enrollment or course addition.
4. **Real-time Updates**:
   - The current time updates every second, displaying it dynamically on the page.

How the project (front end) works in react

In the Login.tsx and Dashboard.tsx components, we utilized a range of React concepts and techniques that form the foundation of a dynamic, interactive user interface. Here's an overview of the primary React concepts used, how we applied them, and how the components interact within this project.

1. React Functional Components

- **Concept**: Functional components are simpler and more readable, enabling us to use hooks directly.
- **Application**: Both Login.tsx and Dashboard.tsx are built as functional components, allowing us to easily manage state, side effects, and event handling with React Hooks.

2. React Hooks

- **Concept**: Hooks let us access React features without needing class components, enabling state and lifecycle functionalities within functional components.
- **Application in Project**:
  - **useState**: Manages component-specific states like input values, modal visibility, and data lists.
    - **Login.tsx**: Uses useState for email, password, errorMessage, and currentTime.
    - **Dashboard.tsx**: Extensively uses useState to handle data fetched from APIs, modal visibility, and form input values for creating courses.
  - **useEffect**: Manages side effects, such as data fetching and time-based updates.
    - **Login Page**: Updates currentTime every second.
    - **Dashboard Page**: Fetches data from the backend on component mount based on userType, which controls whether the user sees student or admin features.

3. Conditional Rendering

- **Concept**: React conditionally renders elements based on component state or props, allowing different UIs to be displayed based on certain conditions.
- **Application**:
  - **Login.tsx**: After a successful login, navigates the user to Dashboard with user information.
  - **Dashboard.tsx**: Displays different layouts for students and admins based on userType. For instance, userType === 1 (Admin) shows a list of students and courses with the ability to add a course, while userType === 2 (Student) shows enrolled courses and available courses for enrollment.

4. Props and State Management Across Components

- **Concept**: props allow passing data from parent to child components, while state maintains local component data.
- **Application**:
  - **Data Passing via State in navigate**: On successful login, Login.tsx uses navigate to pass fullName, userId, and userType to Dashboard.tsx, determining the dashboard view.
  - **State and Props**: Dashboard.tsx uses state to hold data for display and pass it to modal components, like showing student enrollment details or course descriptions in modals.

5. React Router for Navigation

- **Concept**: React Router provides navigation capabilities within single-page applications (SPAs).
- **Application**:
  - **Login Page**: Uses navigate from react-router-dom to programmatically route to the dashboard after a successful login.
  - **Dashboard Page**: Uses useLocation to receive the user information (passed from Login.tsx) to control which components display based on userType.

6. Handling User Input and Events

- **Concept**: React provides event handling directly within JSX, allowing interactions with form inputs, buttons, and other UI elements.
- **Application**:
  - **Login Page**: handleLogin function intercepts form submission to send user credentials to the backend, preventing page reload with e.preventDefault().
  - **Dashboard Page**: Functions like handleAddCourse, handleEnroll, and handleStudentClick respond to button clicks for adding courses, enrolling in classes, or viewing specific student/course details, triggering modals and data refreshes.

7. API Calls with Axios

- **Concept**: Axios is a popular library for making HTTP requests.
- **Application**:
  - **Data Fetching**: Both pages use axios to fetch or send data to a backend. For instance, Login.tsx sends login credentials and Dashboard.tsx fetches courses, enrollments, and students based on user role. We wrapped these requests in try-catch blocks for error handling and to provide feedback to the user (e.g., setting error messages).

8. Modals, Collapse, and Component-Specific UI Elements (Chakra UI)

- **Concept**: Chakra UI provides ready-to-use components like Modal and Collapse, enhancing React's capability for complex UIs.
- **Application**:
  - **Dashboard Page**:
    - Admin users can toggle views of the student and course lists using Chakra's Collapse.
    - Modals allow admins to add new courses and view specific student or course details.
    - A detailed enrollment modal shows students' enrolled courses with scrollable cards, using Chakra's Modal and Button

components to create smooth user experiences.

9. Error Handling and Feedback Mechanisms

- **Concept**: Providing real-time user feedback ensures smooth UX, especially during asynchronous actions.
- **Application**:
  - **Login Page**: Displays error messages when login fails due to incorrect credentials or network issues.
  - **Dashboard Page**: Shows error messages when API requests fail (e.g., handleAddCourse or handleEnrollerrors), keeping users informed.

How Components Interact

1. **Login and Dashboard Interaction**:
   - Login.tsx checks credentials and, upon success, uses navigate to transfer control to Dashboard.tsx, passing userType, userId, and fullName as state. This state controls the initial setup in Dashboard.tsx and determines whether the user views an admin or student dashboard.
2. **Data Fetching and Updates**:
   - Dashboard.tsx manages most backend interactions, fetching data upon mount with useEffect and updating data as user interactions require, ensuring a real-time display of students, courses, and enrollment statuses.
3. **UI Updates and Feedback**:
   - Both components use local state (useState) to display success or error messages, modal visibility, and real-time information like currentTime, making the UI responsive to user actions.