**Billsight Frontend Architecture & Work Structure**

The Billsight frontend is a modern, responsive Single-Page Application (SPA) built with React. It provides a dynamic and user-friendly interface for interacting with the Billsight backend API. The application allows users to manage their accounts, upload and view receipts, analyze their spending habits, and export their data. The architecture is component-based, promoting reusability and a clean separation of concerns.

**Core Technologies**

* **Framework**: React (for building the user interface)
* **Routing**: React Router (react-router-dom) for client-side navigation between pages.
* **API Communication**: Axios (for making HTTP requests to the backend API).
* **State Management**: React Context API (for global state like authentication) and React Hooks (useState, useEffect) for local component state.
* **Charting**: Recharts (for data visualization on the Analytics page).
* **File Uploads**: React Dropzone (for a user-friendly drag-and-drop upload experience).
* **Styling**: Standard CSS with dedicated stylesheets for each component or page.

**Architecture**

The frontend follows a standard component-based architecture. The application is structured as a tree of components, starting from the root App component.

**Data Flow:**

1. **API Layer (api/)**: An Axios instance is configured to communicate with the backend. An interceptor automatically attaches the user's JWT token to every outgoing request.
2. **State Management (context/)**: A global AuthContext provides authentication status (user and token) to all components, preventing the need to pass props down through many levels.
3. **Pages (pages/)**: These are top-level components that represent a full view or "page" of the application (e.g., Dashboard, ReceiptsPage). They are responsible for fetching the data they need from the API and managing the page-level state.
4. **Components (components/)**: These are smaller, reusable UI pieces (e.g., Navbar, GoalEditor, ReceiptViewModal) that receive data and functions as props from their parent pages.

**Folder and File Structure Breakdown**

The billsight-frontend/src/ directory is organized to maintain a clean and logical structure:

* **index.js**: The entry point of the React application. It renders the main App component.
* **App.js**: The root component. It sets up the application's routing using React Router and wraps the entire application in the AuthProvider to provide global state.
* **api/**: Contains the API communication logic.
  + **axios.js**: Configures a global Axios instance with the backend's base URL and an interceptor to automatically add the JWT Authorization header to all requests.
* **context/**: Manages the global state of the application.
  + **AuthContext.js**: Provides authentication state (user, token) and functions (login, logout) to any component that needs it. This is the central hub for managing user sessions.
* **pages/**: Each file represents a major view or page in the application.
  + **Dashboard.js**: The main landing page after login. Displays summary KPIs, recent receipts, and quick actions.
  + **ReceiptsPage.js**: The core page for managing receipts. It includes the file uploader, filter controls, and the main receipts table with inline editing.
  + **AnalyticsPage.js**: Displays charts and graphs visualizing the user's spending data.
  + **ExportPage.js**: Allows users to configure and generate data exports in CSV or JSON format.
  + **LoginPage.js & RegisterPage.js**: Handle user authentication forms.
* **components/**: Contains reusable UI components used across different pages.
  + **Navbar.js**: The top navigation bar, visible on all authenticated pages.
  + **GoalEditor.js**: A modal component for setting the monthly spending goal.
  + **ReceiptViewModal.js**: A modal for viewing the details and image of a single receipt.
  + **ProtectedRoute.js**: A wrapper component that prevents unauthenticated users from accessing protected pages and redirects them to the login page.

**Data Flow Example: User Login**

1. The user visits the application and is shown the **LoginPage.js** component.
2. The user enters their credentials and submits the form.
3. An onSubmit handler in LoginPage.js calls the api.post('/token', ...) function from **api/axios.js**.
4. The backend authenticates the user and returns a JWT token.
5. The onSubmit handler then calls the login() function from the **AuthContext.js**.
6. The login() function saves the received token to the browser's localStorage and updates the global user and token state.
7. Because the authentication state has changed, the **ProtectedRoute.js** component now allows access, and the user is redirected to the **Dashboard.js** page.
8. The Dashboard.js component mounts, and its useEffect hook makes an API call (e.g., /dashboard/summary) to fetch the necessary data. The Axios interceptor automatically attaches the token from localStorage to this request.