Project Documentation: Dashboard System

Project Overview

The Dashboard System is a web-based application designed to provide role-based dashboards for Super Admin, Sub Admin, User, and Customer roles. It supports user authentication, password recovery, and role-specific functionalities such as user management, customer management, and performance tracking. The application features a responsive UI with animations (via Framer Motion), data visualization (via Recharts), and a modular architecture for scalability.

Technology Stack

The project utilizes a modern web development stack to ensure performance, scalability, and developer productivity:

- **Frontend**:
  - **React**: JavaScript library for building dynamic user interfaces.
  - **React Router**: For client-side routing and navigation.
  - **Tailwind CSS**: Utility-first CSS framework for responsive and customizable styling.
  - **Framer Motion**: For animations and transitions in UI components.
  - **Recharts**: For rendering charts (AreaChart, BarChart) in dashboards.
  - **React Icons**: For iconography (e.g., FaEnvelope, FaLock).
- **Build Tools**:
  - **Vite**: Development server and bundler (assumed, based on modern React setup).
  - **ESLint/Prettier** (assumed): For code linting and formatting.
- **Other**:
  - **LocalStorage**: For storing authentication tokens and user data.
  - **CDN**: React and dependencies are assumed to be hosted via cdn.jsdelivr.net for production.

Folder Structure

The project follows a modular, MVC-inspired folder structure for the frontend, promoting maintainability and scalability. Below is the folder structure with a textual tree representation: login screen1/

```
 ├── public
|   ├── index.html
 ├── src
|   ├── components
|   |   ├── Login
|   |   |   ├── Login.css
|   |   |   ├── Login.jsx
|   |   |   ├── LoginForm.jsx
|   |   ├── AuthContext.jsx
|   |   ├── Dashboard.jsx
|   |   ├── Navbar.jsx
|   ├── pages
|   |   ├── Dashboard
|   |   |   ├── SubAdminDashboard.jsx
|   |   |   ├── SuperAdminDashboard.jsx
|   |   ├── CustomerDashboard.jsx
|   |   ├── ForgotPassword.jsx
|   |   ├── UserDashboard.jsx
|   |   ├── CampaignManagement.jsx
|   ├── services
|   |   ├── Authservices.js
|   ├── App.css
|   ├── App.jsx
|   ├── index.css
|   ├── main.jsx
 ├── tailwind.config.js
 ├── package.json
 ├── package-lock.json
```

Folder Structure Screenshot:

- To visualize the folder structure, use the tree command in your terminal (tree -a -I 'node_modules' to exclude node_modules) or capture a screenshot of your IDE's file explorer showing the above structure.

Feature-wise Component & API Flow

The application is organized into key features, each involving specific components and API interactions. Below is the flow for major features:

1. Authentication (Login/Forgot Password)

- **Components**:
  - Login.jsx: Renders the login page layout with logo and form.
  - LoginForm.jsx: Manages login form inputs, validation, and submission. o

    ForgotPassword.jsx: Handles password reset email requests.
  - AuthContext.jsx: Manages authentication state across the app.
- **API Flow**:
  - **Login**:
    - User submits credentials in LoginForm.jsx.
    - handleSubmit uses mock authentication (checks hardcoded credentials).
    - In a real implementation, calls loginUser from authservices.js to send POST request to /api/auth/login.
    - Stores token and user data in localStorage and updates AuthContext.
    - Redirects to role-specific dashboard (/super-admin-dashboard, /subadmin-dashboard, /user-dashboard, /customer-dashboard).

- **Forgot Password**:
  - User submits email in ForgotPassword.jsx.
  - Simulates API call (2-second delay) or calls forgotPassword from authservices.js to send POST to /api/auth/forgot-password.
  - Displays success or error message.

- **Flow Diagram**:
- User -> LoginForm.jsx -> authservices.js -> (/api/auth/login) -> AuthContext > Dashboard
- User -> ForgotPassword.jsx -> authservices.js -> (/api/auth/forgot-password) -> Success/Error UI

2. Dashboards (Super Admin, Sub Admin, User, Customer)

- **Components**:
  - Dashboard.jsx: Core dashboard with role-based metrics and charts.
  - SuperAdminDashboard.jsx: Wraps Dashboard.jsx with role="superadmin". ○ SubAdminDashboard.jsx: Wraps Dashboard.jsx with role="subadmin". ○ UserDashboard.jsx: Manages user data with filtering and actions. ○ CustomerDashboard.jsx: Manages customer data with filtering and actions.
  - Navbar.jsx: Provides navigation across dashboards.
- **API Flow**:
  - Uses mock data (superAdminMetrics, subAdminMetrics, initialUsers, initialCustomers).
- **Flow Diagram**:
- User -> App.jsx -> Routes -> Dashboard (SuperAdmin/SubAdmin) -> Fetch Metrics -> Render Charts/Cards
- User -> UserDashboard.jsx -> Filter Users -> Render Table/Cards
- User -> CustomerDashboard.jsx -> Filter Customers -> Render Cards

3. Navigation and Logout

- **Components**:
  - Navbar.jsx: Renders role-based navigation and logout button.

    - AuthContext.jsx: Handles logout logic by clearing localStorage.

- **Flow Diagram**:
- User -> Navbar.jsx -> handleLogout -> AuthContext.logout -> Clear localStorage -> Redirect to /login

Component Descriptions

Below is a detailed explanation of each file's purpose and functionality:

1. **Login.css**

- **Purpose**: Defines styles for the login page, including animations and responsive design.
- **Functionality**:
    - fadeIn animation for login container entrance. ○ shake animation for error messages. ○ Input focus effects with blue shadow.
    - Button hover (translate) and active states.
- **Usage**: Applied to Login.jsx and LoginForm.jsx via className.

2. **Login.jsx**

- **Purpose**: Renders the login page layout with a logo, title, and LoginForm.
- **Functionality**:
    - Displays a gradient logo and centered title. ○ Renders LoginForm within a shadowed card. ○ Includes a footer with copyright text.
    - Uses Tailwind CSS for responsive styling.
- **Dependencies**: LoginForm.jsx, Login.css.

3. **LoginForm.jsx**

- **Purpose**: Manages the login form, user input, and authentication logic.
- **Functionality**:
    - Handles email, password, "Remember Me," and show/hide password state. ○ Validates email against hardcoded list (superadmin@example.com, etc.).
    - Checks credentials for Super Admin (SuperPass123), Sub Admin (SubPass123), User (user123), and Customer (customer123).

    - Stores email in localStorage if "Remember Me" is checked. ○ Redirects to role-specific dashboards using useNavigate.
    - Displays error messages and loading spinner.
    - Uses icons (FaEnvelope, FaLock, FaEye, FaEyeSlash) for UI.
- **Dependencies**: react, react-router-dom, react-icons/fa.

4. **AuthContext.jsx**

- **Purpose**: Provides a context for authentication state management.
- **Functionality**:
    - Creates AuthContext and useAuth hook. ○ Manages user, isAuthenticated, and loading states. ○ Checks localStorage for authToken and userData on mount. ○ Provides login, logout, getToken, and updateUser methods.
    - Wraps app with AuthProvider to share context.
- **Dependencies**: react.

5. **Dashboard.jsx**

- **Purpose**: Renders role-specific dashboard content with metrics and charts.
- **Functionality**:
    - Displays superAdminMetrics (Total Users, Customers, Revenue, Campaigns, Active Sub Admins) or subAdminMetrics (Assigned Tasks, Reviewed Screenshots, Wallet Actions) based on role.
    - Uses Recharts for Super Admin charts (AreaChart for user growth, BarChart for campaign performance).
    - Uses Framer Motion for animated counters and card transitions.

      Includes Sub Admin quick action buttons (Screenshot Verification, Task Assignment).

6. **Dependencies**: react, recharts, framer-motion.
   Navbar.jsx

- **Purpose**: Renders a responsive navigation bar with role-based branding.
- **Functionality**:

    - Displays logo with role-specific initials (e.g., "SA" for admins).
    - Shows role-specific title and color (purple-600 for Super Admin, green-600 for Sub Admin).
    - Includes notification button (hidden on mobile) and user avatar.
    - Provides logout button to clear localStorage and redirect to /login.
- **Dependencies**: react, react-router-dom.

7. **SubAdminDashboard.jsx**
    - **Purpose**: Entry point for Sub Admin dashboard.
    - **Functionality**:
        - Renders Navbar with role="subadmin". o Renders Dashboard with role="subadmin".
        - Sets gray background for the page.
    - **Dependencies**: Navbar.jsx, Dashboard.jsx.

8. **SuperAdminDashboard.jsx**
    - **Purpose**: Entry point for Super Admin dashboard.
    - **Functionality**:
        - Renders Navbar with role="superadmin". o Renders Dashboard with role="superadmin".
        - Sets gray background for the page.
    - **Dependencies**: Navbar.jsx, Dashboard.jsx.

9. **CustomerDashboard.jsx**
    - **Purpose**: Manages customer data with filtering and actions.
    - **Functionality**:

        Displays mock customer data (initialCustomers) with name, email, plan, campaigns, wallet, and stats.
        - Supports filtering by name/email and plan (Bronze, Silver, Gold). o Renders expandable cards with action buttons (Approve Campaigns, Adjust Wallet, Change Plan, View Stats).

    - Uses dynamic badge colors for plans (e.g., yellow-500 for Gold).
- **Dependencies**: react.

Campaign Management.jsx

- **Stats Table:** Likes, Shares, Views (platform-wise)
- **Actions:** Approve / Reject, Assign Tasks, Pause / Edit
- **UX Enhancements:** Progress bars (performance), Campaign thumbnails (quick glance)

-

10. ForgotPassword.jsx

- **Purpose**: Handles password reset requests.
- **Functionality**:
    - Renders email input form with a simulated API call (2-second delay). ○ Displays success message with email confirmation or error message. ○ Provides "Try again" and "Back to Sign in" options.
    - Includes logo, title, and footer.
- **Dependencies**: react, react-router-dom, react-icons/fa.

11. UserDashboard.jsx

- **Purpose**: Manages user data with filtering and actions.
- **Functionality**:
    - Displays mock user data (initialUsers) with name, email, location, age, gender, referral, and wallet.
    - Supports filtering by search, location, age, gender, and referral. ○ Renders table (desktop) or cards (mobile) with action buttons (View Profile, Assign Task, Verify User, Ban User, Adjust Wallet).
    - Updates user state for verification, banning, and wallet adjustments.
- **Dependencies**: react, react-icons/fa.

12. Authservices.js

- **Purpose**: Provides API service functions for authentication.

- **Functionality**:
    - Defines loginUser to send POST to /api/auth/login.

      Defines forgotPassword to send POST to /api/auth/forgot-password.
    - Uses environment-based API_URL (/api/auth for development).
    - Handles errors with meaningful messages.
- **Dependencies**: axios.

13. App.jsx

- **Purpose**: Configures client-side routing.
- **Functionality**:
    - Uses react-router-dom to define routes for /login, /super-admin-dashboard, /sub-admin-dashboard, /user-dashboard, /customer-dashboard, and /forgotpassword.
    - Renders components based on URL path.
- **Dependencies**: react-router-dom, all page components.

14. App.css

- **Purpose**: Placeholder for application-specific styles.
- **Functionality**: Not used in the provided code; can be used for custom styles outside index.css.
- **Dependencies**: None.

15. index.css

- **Purpose**: Defines global styles for the application.
- **Functionality**:
    - Imports Google Fonts (Inter) and Tailwind CSS. o Sets default font and background color (#f6f3f3).
    - Customizes input focus styles with blue ring.
- **Dependencies**: Tailwind CSS, Google Fonts.

16. main.jsx

- **Purpose**: Entry point for rendering the React application.

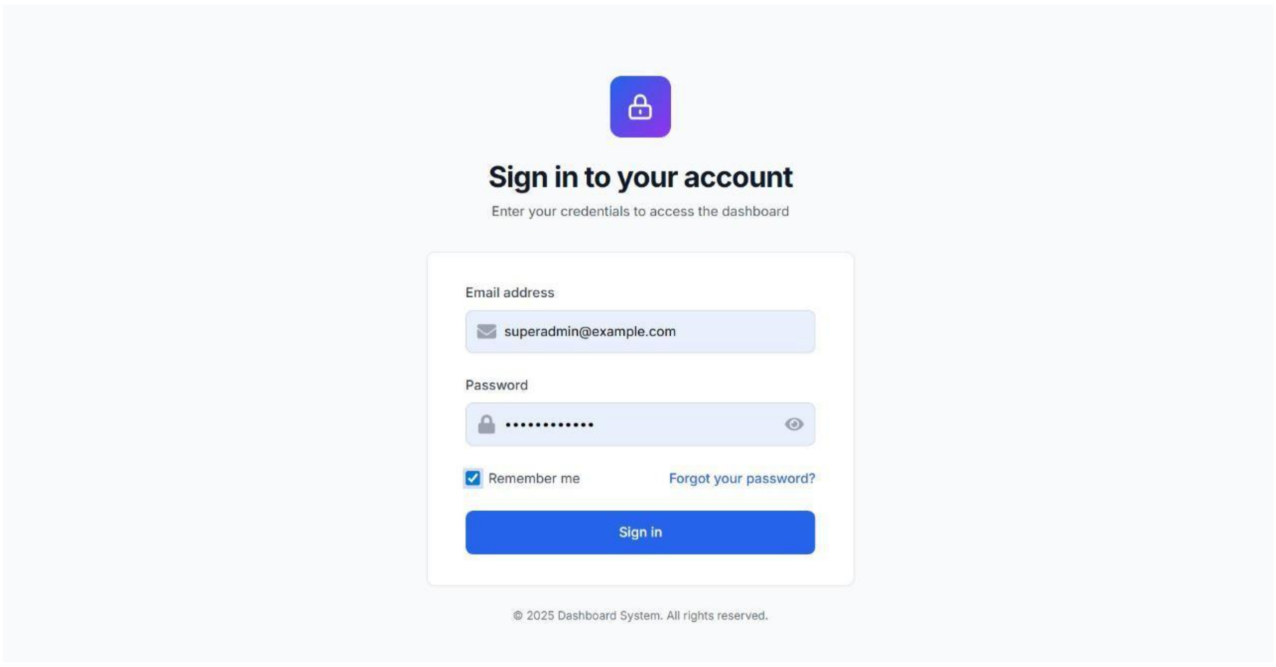
- **Functionality**:
    - Renders App.jsx into the DOM at #root.
    - Wraps App with AuthProvider for authentication context.

- **Dependencies**: react, react-dom, AuthContext.jsx, App.jsx.

17. tailwind.config.js

- **Purpose**: Configures Tailwind CSS.
- **Functionality**:
    - Specifies content files for Tailwind to scan (index.html, src/**/*).
    - Extends theme with custom colors (primary, secondary, accent, background) and font family (Inter).
- **Dependencies**: Tailwind CSS.
-

Screenshots



Sign in to your account

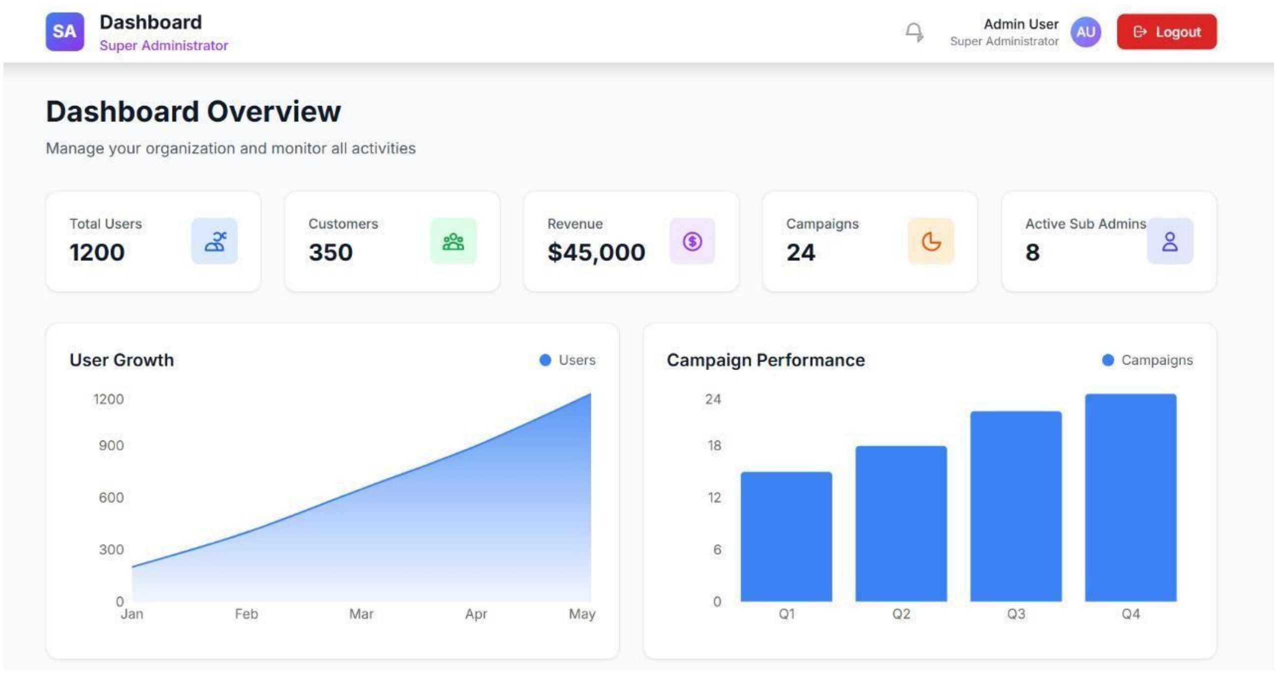
Enter your credentials to access the dashboard

Email address: superadmin@example.com

Password: ••••••••••••

[✓] Remember me

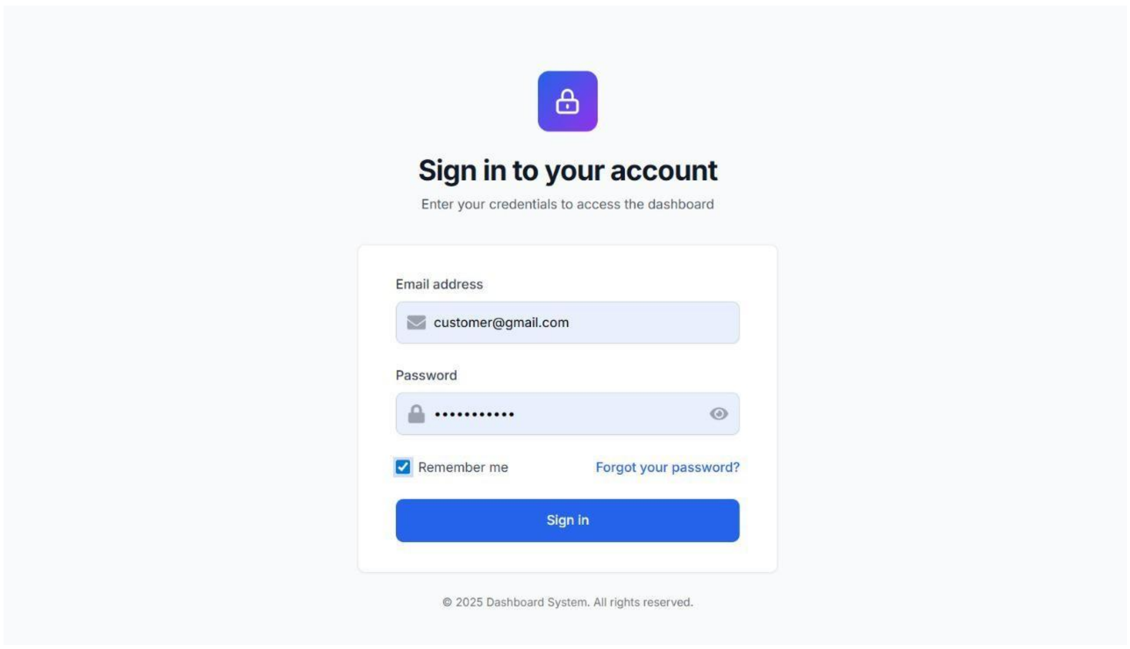
© 2025 Dashboard System. All rights reserved.



Dashboard Overview

Manage your organization and monitor all activities

Total Users	Customers	Revenue	Campaigns	Active Sub Admins
1200	350	$45,000	24	8



Sign in to your account

Enter your credentials to access the dashboard

Email address

customer@gmail.com

Password

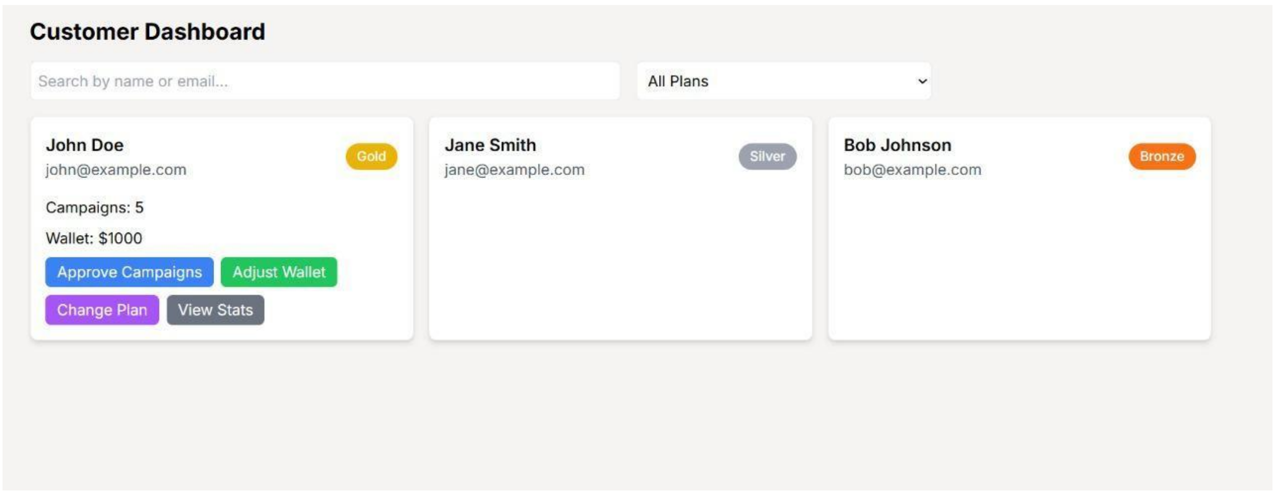
•••••••••••

[✓] Remember me

Forgot your password?

Sign in

© 2025 Dashboard System. All rights reserved.



Customer Dashboard

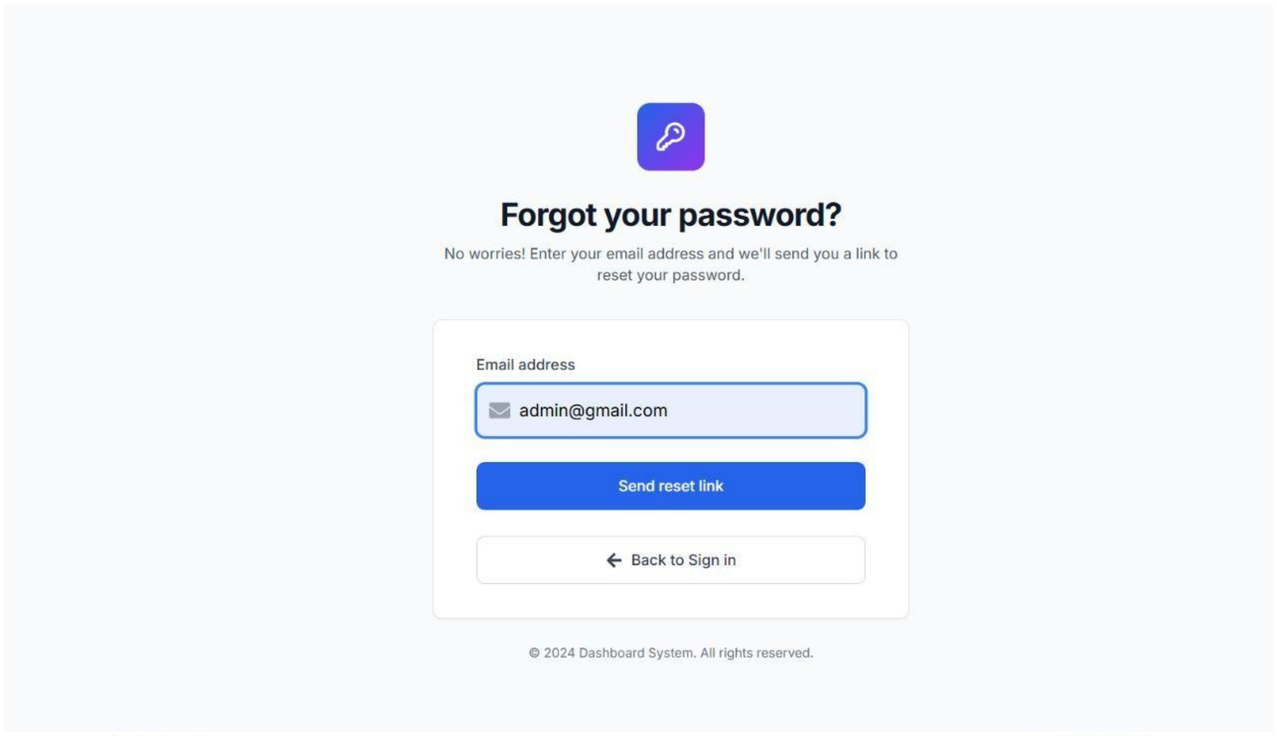
Search by name or email...

All Plans

John Doe
john@example.com
Gold
Campaigns: 5
Wallet: $1000

Jane Smith
jane@example.com
Silver

Bob Johnson
bob@example.com
Bronze



Forgot your password?

No worries! Enter your email address and we'll send you a link to reset your password.

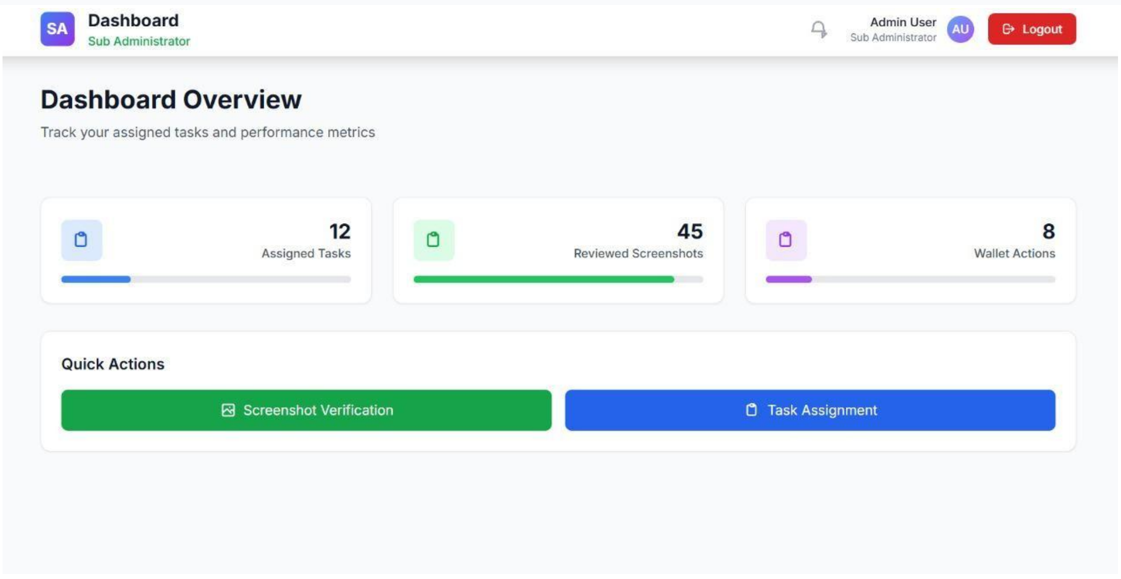
Email address

admin@gmail.com

Send reset link

Back to Sign in

© 2024 Dashboard System. All rights reserved.



SA Dashboard
Sub Administrator

Admin User
Sub Administrator AU
Logout

Dashboard Overview

Track your assigned tasks and performance metrics

12 Assigned Tasks

45 Reviewed Screenshots

8 Wallet Actions

Quick Actions

Screenshot Verification

Task Assignment

How to Run the Project

Follow these steps to set up and run the project locally.

Steps

1. **Clone the Repository** (if applicable):
2. git clone <repository-url>
3. cd login-screen1
4. **Install Frontend Dependencies**:
5. npm install

Installs dependencies: react, react-router-dom, recharts, framer-motion, react-icons, axios, tailwindcss.

8. npm run dev o Starts the Vite development server at

   http://localhost:5173.

   - If using Create React App, use npm start (port http://localhost:3000).

7. **Access the Application**:

   - Open http://localhost:5173 in a browser.
   - Use credentials:
     - Super Admin: superadmin@example.com / SuperPass123
     - Sub Admin: subadmin@example.com / SubPass123
     - User: user@gmail.com / user123
     - Customer: customer@gmail.com / customer123

Conclusion

Challenges Faced

- **Mock Data**: Reliance on hardcoded data (initialUsers, initialCustomers, mock credentials) limits dynamic functionality.
- **Authentication**: LoginForm.jsx uses mock authentication, requiring integration with authservices.js for real API calls.
- **State Management**: AuthContext is sufficient for small apps for complex state handling.