

AGILE TRACK SYSTEM

IIHT

Time To Complete: 2 hr

CONTENTS

1	Project Abstract	3
2	Problem Statement	3
3	Proposed Agile Track System Application Wireframe	4
3.1	Welcome Page	4
3.2	User Login	5
3.2	Admin Login	7
4	Business-Requirement	11
5	Validations	22
6	Constraints	22
7	Mandatory Assessment Guidelines	22

1 PROJECT ABSTRACT

In the rapidly evolving landscape of project management, the need for streamlined and accessible platforms to manage agile workflows has become increasingly important. With the vision of creating an innovative and user-friendly Agile Track System, the CEO of a growing software development company, Mr. X, assigns a team of developers to build an application using React.

This application aims to provide a user-friendly and efficient platform for both team members and administrators, enhancing the overall task management and team collaboration experience.

Your task is to develop a comprehensive digital solution that enables users to log in, view the tasks assigned to them, and track their progress within their respective scrum teams. Allows administrators to monitor and manage users, scrum teams, and tasks efficiently, fostering a collaborative and productive work environment. Provides detailed insights into task statuses, user assignments, and team performance, facilitating better decision-making and agile project management.

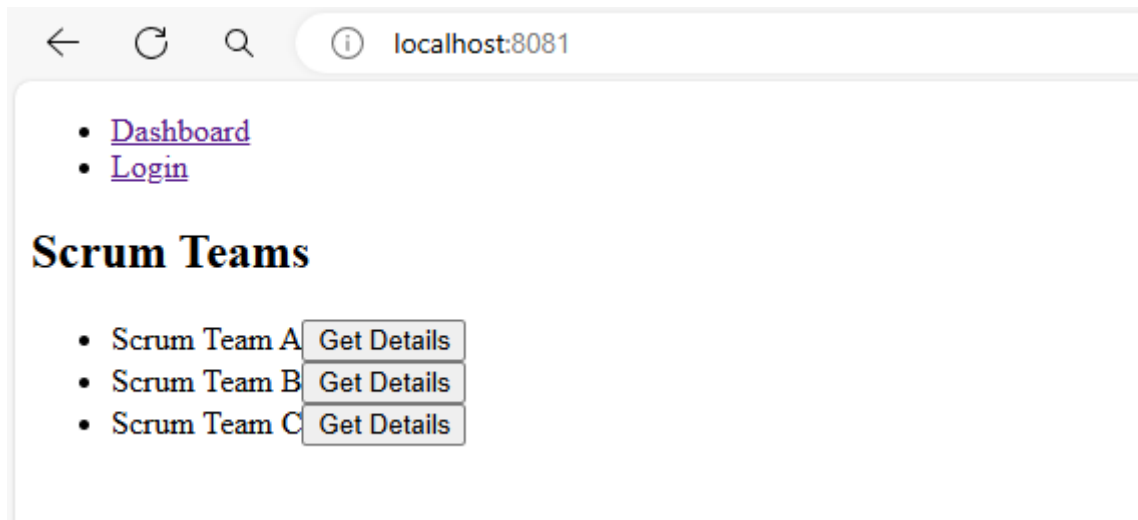
2 PROBLEM STATEMENT

The "**Agile Track System**" is a Single Page Application (SPA) designed to streamline task management and team collaboration within agile frameworks. This platform allows users and administrators to log in, view tasks assigned to users, and track progress across various scrum teams.

3 PROPOSED AGILE TRACK SYSTEM WIREFRAME

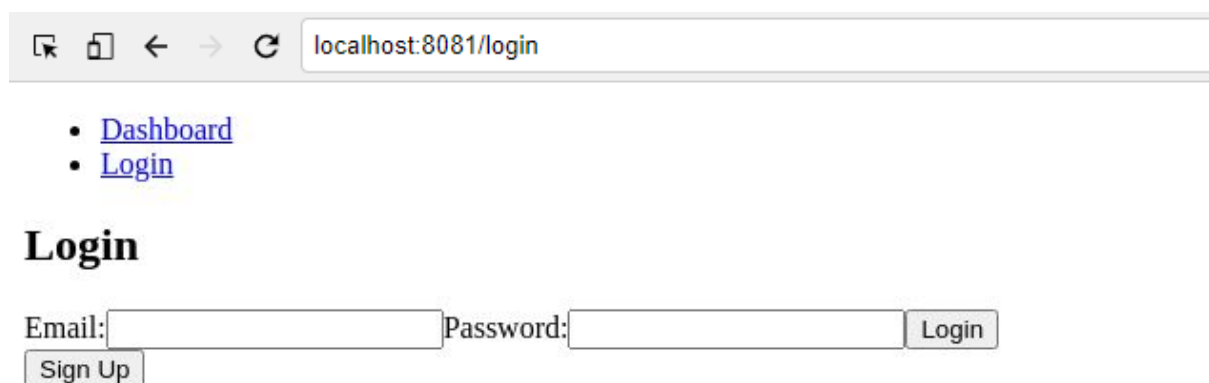
UI needs improvisation and modification as per given use case and to make test cases passed.

3.1 WELCOME PAGE

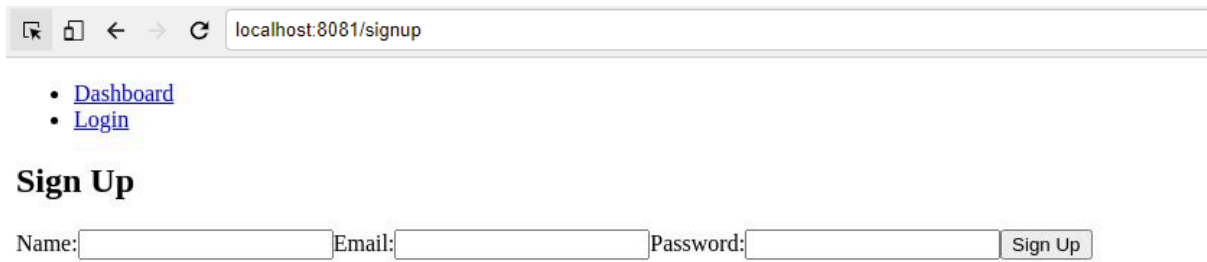


Already added users and admin with their details

*** Login Page***



*** Sign Up Page***



The screenshot shows a web browser window with the address bar displaying 'localhost:8081/signup'. Below the address bar, there are two links: 'Dashboard' and 'Login'. The main heading is 'Sign Up'. Below the heading, there are three input fields labeled 'Name:', 'Email:', and 'Password:', followed by a 'Sign Up' button.

• [Dashboard](#)

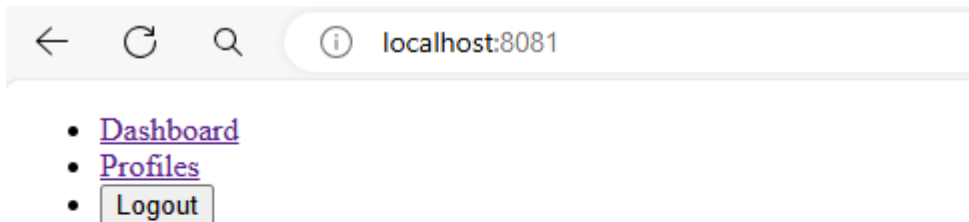
• [Login](#)

Sign Up

Name: Email: Password:

3.2 USER LOGIN

*** User Home Page***



The screenshot shows a web browser window with the address bar displaying 'localhost:8081'. Below the address bar, there are three links: 'Dashboard', 'Profiles', and 'Logout'.

• [Dashboard](#)

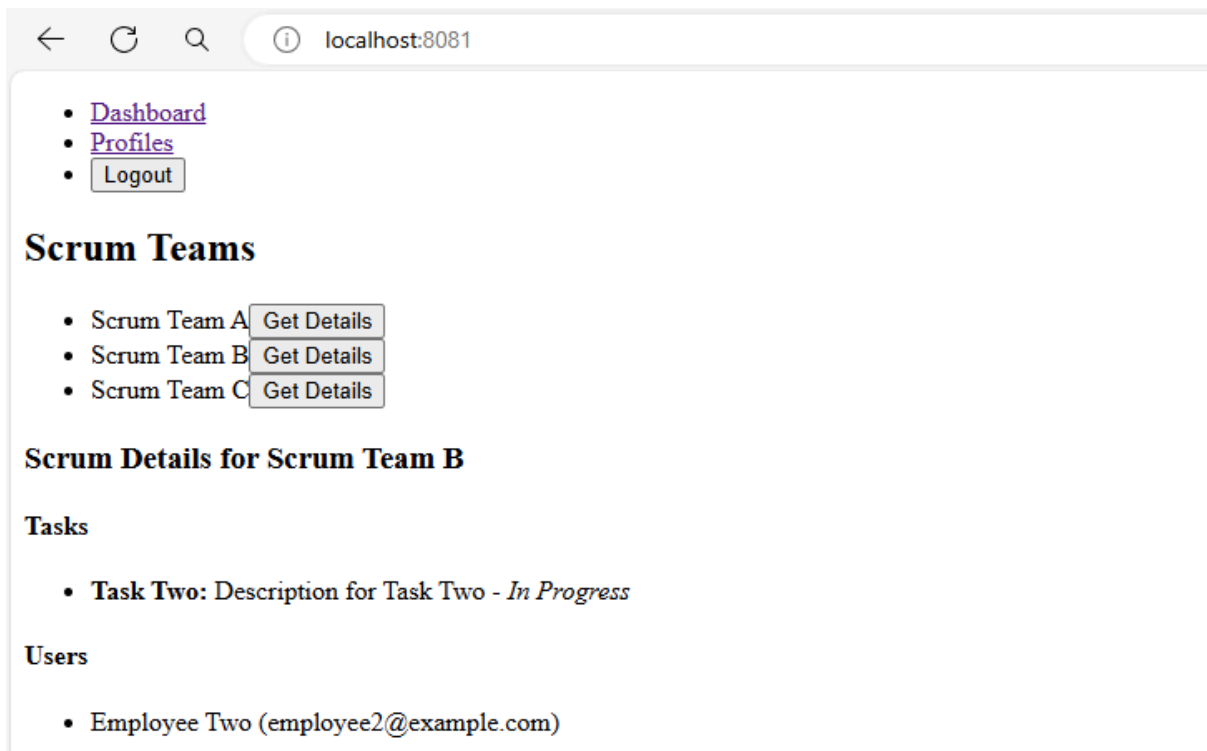
• [Profiles](#)

•

Scrum Teams

- Scrum Team A
- Scrum Team B
- Scrum Team C

*** Team Details***



• [Dashboard](#)

• [Profiles](#)

• [Logout](#)

Scrum Teams

- Scrum Team A [Get Details](#)
- Scrum Team B [Get Details](#)
- Scrum Team C [Get Details](#)

Scrum Details for Scrum Team B

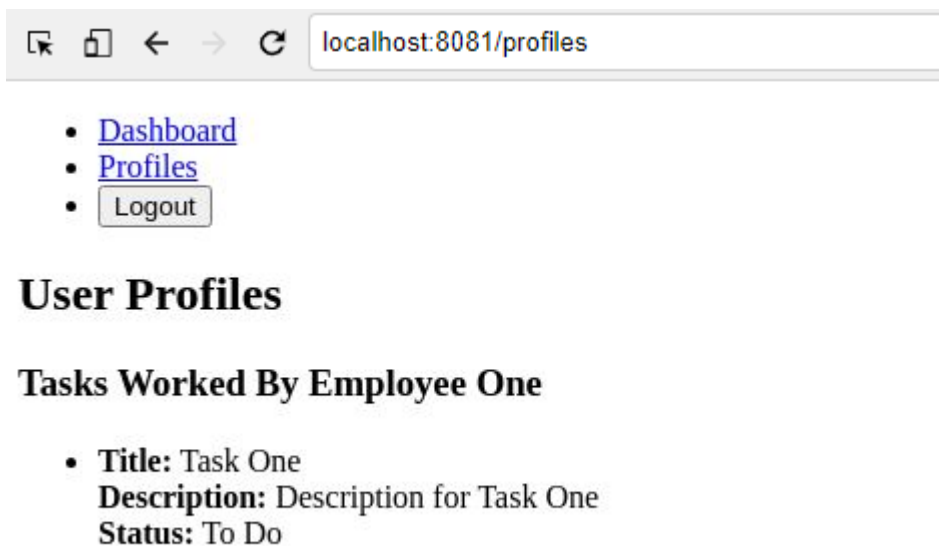
Tasks

- **Task Two:** Description for Task Two - *In Progress*

Users

- Employee Two (employee2@example.com)

*** User Profile Page***



• [Dashboard](#)

• [Profiles](#)

• [Logout](#)

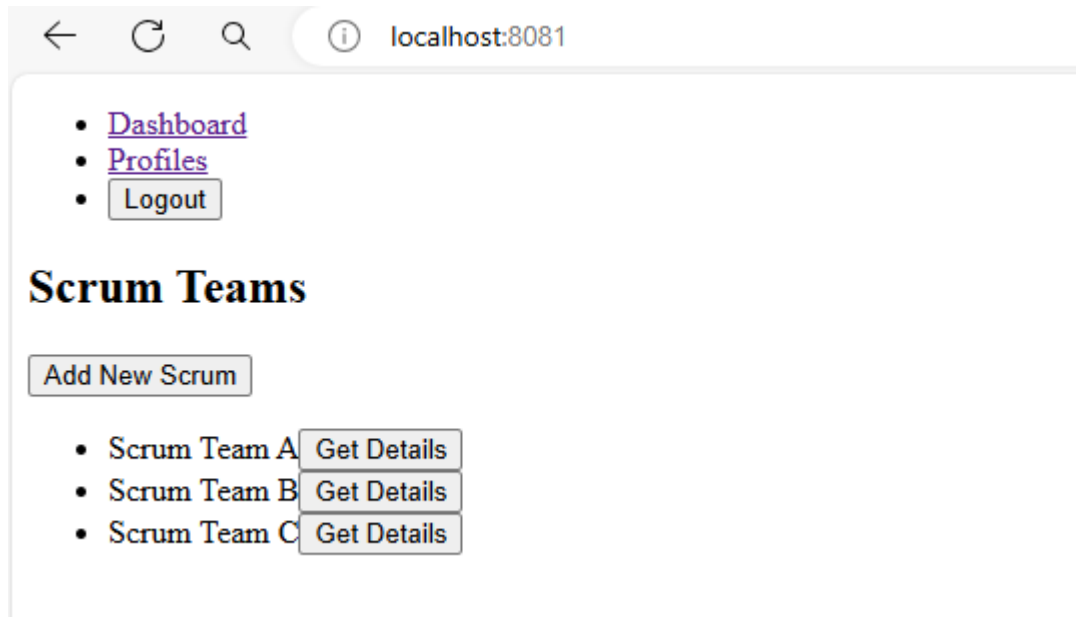
User Profiles

Tasks Worked By Employee One

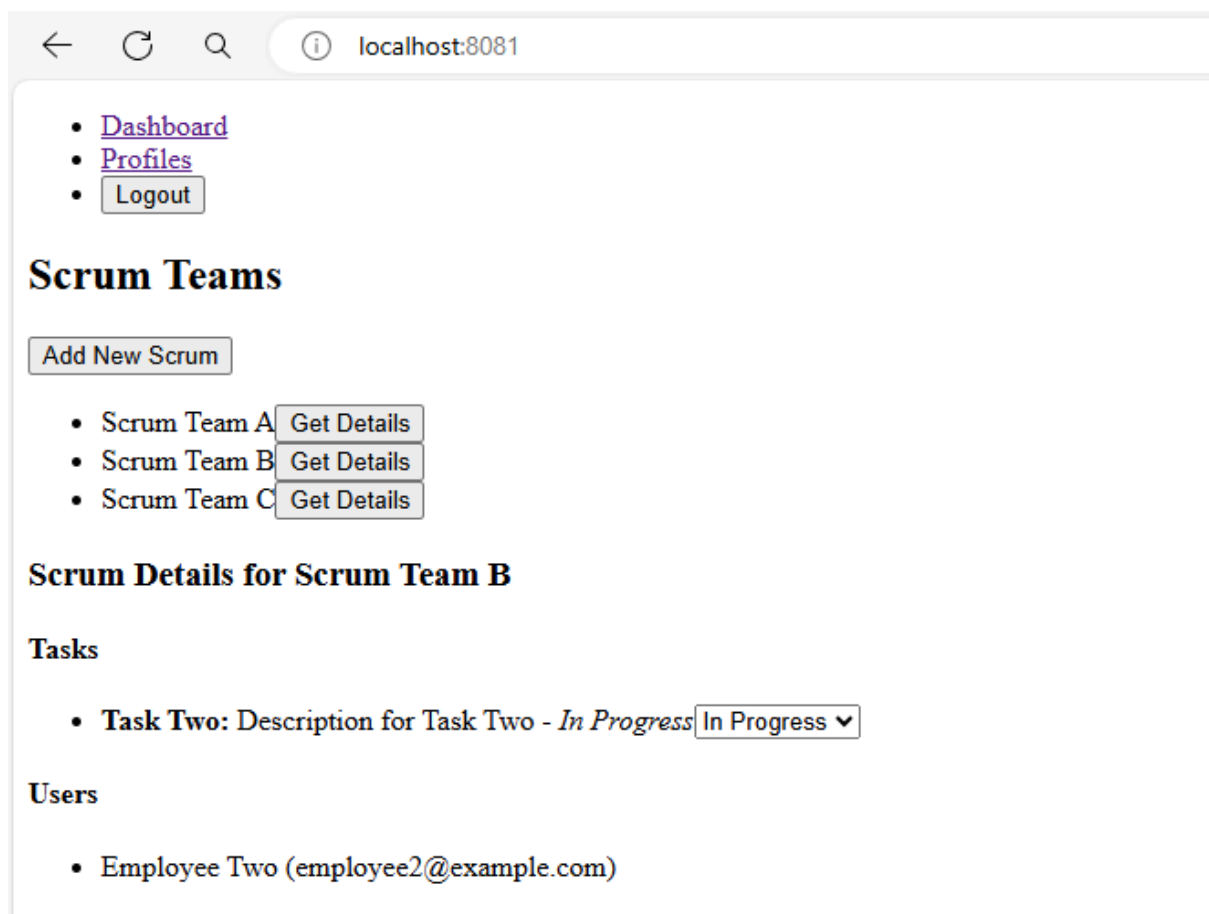
- **Title:** Task One
Description: Description for Task One
Status: To Do

3.3 ADMIN LOGIN

*** Admin Home Page***



*** Team Details***



*** Add New Scrum***

[←](#) [↻](#) [🔍](#) [ℹ](#) localhost:8081

- [Dashboard](#)
- [Profiles](#)
- [Logout](#)

Scrum Teams

[Cancel](#)
Scrum Name:
Task Title:
Task Description:
Task Status: To Do ▼
Assign To: Select a user ▼
[Create Scrum](#)

- Scrum Team A [Get Details](#)
- Scrum Team B [Get Details](#)
- Scrum Team C [Get Details](#)

*** Update Task Status***

[←](#) [↻](#) [🔍](#) [ℹ](#) localhost:8081

- [Dashboard](#)
- [Profiles](#)
- [Logout](#)

Scrum Teams

[Add New Scrum](#)

- Scrum Team A [Get Details](#)
- Scrum Team B [Get Details](#)
- Scrum Team C [Get Details](#)

Scrum Details for Scrum Team B

Tasks

- **Task Two:** Description for Task Two - *In Progress* In Progress ▼

Users

- Employee Two (employee2@example.com)

To Do
In Progress
Done

*** Profiles Page***



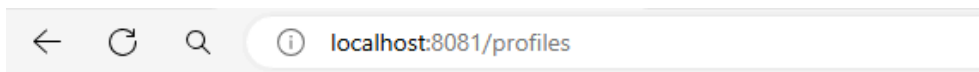
- [Dashboard](#)
- [Profiles](#)
- [Logout](#)

User Profiles

[Add New User](#)

- **Name:** Employee One
Email: employee1@example.com
[Get History](#)
- **Name:** Employee Two
Email: employee2@example.com
[Get History](#)
- **Name:** Employee Three
Email: employee3@example.com
[Get History](#)

*** Add New User***



- [Dashboard](#)
- [Profiles](#)
- [Logout](#)

User Profiles

[Cancel](#)

Name:

Email:

Password:

Role:

[Create User](#)

- **Name:** Employee One
Email: employee1@example.com
[Get History](#)
- **Name:** Employee Two
Email: employee2@example.com
[Get History](#)
- **Name:** Employee Three
Email: employee3@example.com
[Get History](#)

*** Get History of an User***

[←](#) [↻](#) [🔍](#) [localhost:8081/profiles](#)

- [Dashboard](#)
- [Profiles](#)
- [Logout](#)

User Profiles

[Add New User](#)

- **Name:** Employee One
Email: employee1@example.com
[Get History](#)
- **Name:** Employee Two
Email: employee2@example.com
[Get History](#)
- **Name:** Employee Three
Email: employee3@example.com
[Get History](#)

Tasks Worked By Employee Two

- **Title:** Task Two
Description: Description for Task Two
Status: In Progress
- **Title:** Task 3
Description: Description for Task 3
Status: In Progress

4 BUSINESS-REQUIREMENT:

As an application developer, develop the Agile Track System (Single Page App) with below guidelines:

User Story #	User Story Name	User Story
US_01	Welcome Page	<p>As a user I should be able to visit the welcome page as default page.</p> <p>Acceptance Criteria:</p> <p>AppComponent:</p> <ol style="list-style-type: none">Should use React Router to manage routing between:<ul style="list-style-type: none">Dashboard (/)Login (/login)Signup (/signup)Profiles (/profiles)Should wrap the entire app in a UserProvider to share user data across components using context.Should render a navigation bar with:<ul style="list-style-type: none">Links to relevant pagesLogout functionality (if user is logged in)Login link (if user is not logged in) <p>UserContext:</p> <p>State Variables:</p> <ol style="list-style-type: none">user:<ul style="list-style-type: none">Type: object or nullPurpose: Stores the currently logged-in user's data.Initial Value: null (retrieved from localStorage if available) <p>Functions Provided via Context:</p> <ol style="list-style-type: none">login(userData):<ul style="list-style-type: none">Saves user data to localStorage.Updates the user state.logout():<ul style="list-style-type: none">Clears user data from localStorage.Resets user to null. <p>Authentication Flow:</p> <ol style="list-style-type: none">On App Load:

		<ul style="list-style-type: none"> Automatically checks if a user is stored in <code>localStorage</code> using <code>useEffect</code>. If found, sets the user in context. <p>2. Login Functionality:</p> <ul style="list-style-type: none"> Calls <code>login(userData)</code> with the user's credentials/data. Stores user info and updates state. <p>3. Logout Functionality:</p> <ul style="list-style-type: none"> Calls <code>logout()</code>, clears storage, and navigates back to <code>/login</code>. <p>Nav Component Behavior:</p> <ol style="list-style-type: none"> Always Visible: <ul style="list-style-type: none"> Dashboard link (<code>/</code>) If <code>user</code> exists (i.e., logged in): <ul style="list-style-type: none"> Show: <ul style="list-style-type: none"> → Profiles link (<code>/profiles</code>) → Logout button If <code>user</code> is not logged in: <ul style="list-style-type: none"> Show: <ul style="list-style-type: none"> → Login link (<code>/login</code>) Logout Button: <ul style="list-style-type: none"> Triggers <code>handleLogout()</code> → logs out and redirects to <code>/login</code>. <p>HTML Structure:</p> <ol style="list-style-type: none"> App Component: <ul style="list-style-type: none"> Contains a top-level <code><div></code> with class <code>"app"</code>. Includes a navigation bar and a main section with route-based component rendering. Nav Component: <ul style="list-style-type: none"> Uses a <code><nav></code> with an unordered list <code></code>. List items conditionally render based on the <code>user</code> state. The Logout button should be placed as a <code><button></code> inside a list item.
--	--	---

		<pre>=====</pre> <p>DashboardComponent:</p> <p>Acceptance Criteria:</p> <ol style="list-style-type: none"> 1. Display the heading "Scrum Teams" using an <code><h2></code> tag. 2. Fetch and list all Scrum teams from the backend on initial render. 3. Provide a "Get Details" button for each Scrum to view its tasks. 4. For admin users only: <ul style="list-style-type: none"> • Display a button to toggle the new Scrum creation form. • Allow adding a new Scrum along with an initial task. 5. Render <code>ScrumDetails</code> component to show details for the selected Scrum. <p>State Variables:</p> <ol style="list-style-type: none"> 1. scrums <ul style="list-style-type: none"> • Type: array • Purpose: List of all Scrum teams. 2. selectedScrum <ul style="list-style-type: none"> • Type: object or null • Purpose: Stores the currently selected Scrum for detail view. 3. showForm <ul style="list-style-type: none"> • Type: boolean • Purpose: Toggles visibility of the new Scrum form. 4. users <ul style="list-style-type: none"> • Type: array • Purpose: List of users fetched from the backend (for task assignment). 5. newScrumName, newTaskTitle, newTaskDescription, newTaskStatus, newTaskAssignedTo <ul style="list-style-type: none"> • Type: string • Purpose: Hold form values for new Scrum and task creation. <p>Fetching Data:</p> <p>When the component mounts:</p> <ol style="list-style-type: none"> 1. Fetch all Scrum teams from
--	--	--

		<p>http://localhost:4000/scrums.</p> <ol style="list-style-type: none"> Fetch all users from http://localhost:4000/users. Store results in scrums and users state respectively. <p>Add Scrum Flow:</p> <ol style="list-style-type: none"> Only visible to users with the role admin. Clicking "Add New Scrum" shows a form. Form fields: <ul style="list-style-type: none"> Scrum Name (text, required) Task Title (text, required) Task Description (text, required) Task Status (dropdown: To Do, In Progress, Done) Assign To (dropdown populated from user list) On submission: <ul style="list-style-type: none"> Send POST request to /scrums to create a new Scrum. Then POST to /tasks with task details, including history and scrumId. Refresh the Scrum list after successful addition. Hide form and reset all inputs. <p>Scrum Detail Retrieval:</p> <ol style="list-style-type: none"> Clicking "Get Details" fetches and sets selected Scrum by ID. Uses: GET http://localhost:4000/scrums/{id} Renders the ScrumDetails component if a Scrum is selected. <p>HTML Structure:</p> <ol style="list-style-type: none"> Display an <h2> for the section title. Conditionally render the form inside a div for admin users. List all scrums using a ul and li structure. <ul style="list-style-type: none"> Each item includes: <ul style="list-style-type: none"> → Scrum name → A button labeled "Get Details" to trigger fetch. Form elements: <ul style="list-style-type: none"> Inputs for Scrum name, task title, and description (type
--	--	--

		<p><code>"text")</code></p> <ul style="list-style-type: none"> • Dropdowns for task status and assigned user • Submit button to create the Scrum <p>=====</p> <p>LoginComponent:</p> <p>Acceptance Criteria:</p> <ol style="list-style-type: none"> 1. Display the heading "Login" using an <code><h2></code> tag. 2. Render a form with: <ul style="list-style-type: none"> • Email input (required) • Password input (required) • Submit button 3. On form submission: <ul style="list-style-type: none"> • Authenticate user by matching email/password via GET request to backend. • On success: <ul style="list-style-type: none"> ➔ Store user in context. ➔ Redirect based on role: <ul style="list-style-type: none"> ➢ <code>admin</code> → <code>/</code> ➢ <code>others</code> → <code>/profiles</code> • On failure: Show alert <code>"Invalid email or password"</code> 4. Provide a "Sign Up" button to navigate to the <code>/signup</code> route. <p>State Variables:</p> <ol style="list-style-type: none"> 1. email <ul style="list-style-type: none"> • Type: string • Purpose: Stores input email. 2. password <ul style="list-style-type: none"> • Type: string • Purpose: Stores input password. <p>Context Function:</p> <ol style="list-style-type: none"> 1. login(userData) from <code>UserContext</code> <ul style="list-style-type: none"> • Updates local storage and user context. <p>Authentication Logic:</p>
--	--	--

		<ol style="list-style-type: none"> Form submission is handled by <code>handleLogin()</code>. Makes a GET request to: <code>http://localhost:4000/users?email={email}&password={password}</code> If a matching user is found: <ul style="list-style-type: none"> Calls <code>login(user)</code> Redirects: <ul style="list-style-type: none"> → Admins → Dashboard (/) → Others → Profiles (<code>/profiles</code>) If no match: <ul style="list-style-type: none"> Shows browser alert. <p>HTML Structure:</p> <ol style="list-style-type: none"> Heading using <code><h2></code> with the text "Login". A form that includes: <ul style="list-style-type: none"> An email input field (type "<code>email</code>") bound to the <code>email</code> state. A password input field (type "<code>password</code>") bound to the <code>password</code> state. A submit button to trigger login. A standalone button labeled "<code>Sign Up</code>" that routes users to the signup page when clicked. <p>=====</p> <p>SignUpComponent:</p> <p>Acceptance Criteria:</p> <ol style="list-style-type: none"> Display the heading "Sign Up" using an <code><h2></code> tag. Render a form with: <ul style="list-style-type: none"> Name input (required) Email input (required) Password input (required) Submit button labeled "Sign Up" On form submission: <ul style="list-style-type: none"> Send a POST request to create a new user with:
--	--	--

		<p>→ name, email, password, and default role: "employee"</p> <ul style="list-style-type: none"> • After successful registration, redirect to Login page (/login). <p>4. Handle errors gracefully and log to console if signup fails.</p> <p>State Variables:</p> <ol style="list-style-type: none"> 1. name <ul style="list-style-type: none"> • Type: string • Purpose: Stores input for user's name. 2. email <ul style="list-style-type: none"> • Type: string • Purpose: Stores input for user's email. 3. password <ul style="list-style-type: none"> • Type: string • Purpose: Stores input for user's password. <p>Functionality:</p> <ol style="list-style-type: none"> 1. Submission is handled by handleSignUp(e): <ul style="list-style-type: none"> • Prevents default form submission. • Makes POST request to: <ul style="list-style-type: none"> → http://localhost:4000/users → with user details and assigns default role "employee". • On success, navigates to the /login route. • On error, logs the issue in the console. <p>HTML Structure:</p> <ol style="list-style-type: none"> 1. Use an <h2> for the section heading with text "Sign Up". 2. Form includes: <ul style="list-style-type: none"> • Input field for Name (type "text") • Input field for Email (type "email") • Input field for Password (type "password") • A submit button with text "Sign Up" <p>=====</p>
--	--	--

		<p>UserProfileComponent:</p> <p>Acceptance Criteria:</p> <ol style="list-style-type: none"> 1. Display heading "User Profiles" using an <code><h2></code> tag. 2. If user is an admin: <ul style="list-style-type: none"> • Show a button to toggle the Add New User form. • Display a list of all non-admin users with: <ul style="list-style-type: none"> → Name, Email, and a "Get History" button to fetch tasks. • Show tasks assigned to a selected user. 3. If user is not an admin: <ul style="list-style-type: none"> • Automatically show only their own task list. <p>State Variables:</p> <ol style="list-style-type: none"> 1. users <ul style="list-style-type: none"> • List of users (filtered for non-admins). 2. tasks <ul style="list-style-type: none"> • List of tasks assigned to the selected user. 3. selectedUser <ul style="list-style-type: none"> • The user selected for viewing task history. 4. showForm <ul style="list-style-type: none"> • Toggles display of the "Add New User" form. 5. newUserName, newUserEmail, newUserPassword, newUserRole <ul style="list-style-type: none"> • Stores values from the "Add User" form. <p>Data Fetching Logic:</p> <ol style="list-style-type: none"> 1. On mount: <ul style="list-style-type: none"> • If admin → fetch all users and display list (excluding admins). • If employee → auto-select a logged-in user and fetch their tasks. 2. Task Fetching: <ul style="list-style-type: none"> • Makes GET request to: <ul style="list-style-type: none"> → <code>http://localhost:4000/tasks?assignedTo={userId}</code> <p>Add User Flow:</p> <ol style="list-style-type: none"> 1. Admin clicks "Add New User" to reveal the form.
--	--	--

		<p>2. Form fields:</p> <ul style="list-style-type: none"> • Name (text) • Email (email) • Password (password) • Role (dropdown: "employee", "admin") <p>3. On submission:</p> <ul style="list-style-type: none"> • Sends POST request to <code>/users</code> with form data. • Refreshes user list and resets form. <p>View Task History:</p> <ol style="list-style-type: none"> 1. Clicking "Get History" fetches and displays tasks assigned to the selected user. 2. Tasks show: <ul style="list-style-type: none"> • Title • Description • Status <p>HTML Structure:</p> <ol style="list-style-type: none"> 1. Use <code><h2></code> for section title. 2. If admin: <ul style="list-style-type: none"> • Show a toggle button labeled "Add New User" / "Cancel" • Show a form with fields: <ul style="list-style-type: none"> → Name (text) → Email (email) → Password (password) → Role (select dropdown) • Display a user list with: <ul style="list-style-type: none"> → Name → Email → Button labeled "Get History" 3. If employee: <ul style="list-style-type: none"> • Show only a heading like "Tasks Worked By {user.name}" • Display a list of tasks, each with: <ul style="list-style-type: none"> → Title → Description
--	--	---

		<p>→ Status</p> <ol style="list-style-type: none"> If admin has selected a user: <ul style="list-style-type: none"> Show the same task structure for the selected user. <p>=====</p> <p>ScrumDetailsComponent:</p> <p>Acceptance Criteria:</p> <ol style="list-style-type: none"> Display a heading “Scrum Details for {scrum.name}” using an <code><h3></code> tag. Show a list of tasks for the given Scrum. For each task: <ul style="list-style-type: none"> Display title, description, and current status. If user is admin, allow status change via dropdown. Show a list of users assigned to tasks in this Scrum. Redirect to the login page if no user is found in local storage. <p>State Variables:</p> <ol style="list-style-type: none"> tasks <ul style="list-style-type: none"> Type: array Purpose: Stores tasks belonging to the current Scrum. users <ul style="list-style-type: none"> Type: array Purpose: Stores users assigned to any of the tasks in the Scrum. <p>Data Fetching Logic:</p> <p>On Component Mount:</p> <ol style="list-style-type: none"> Checks for logged-in user in <code>localStorage</code> <ul style="list-style-type: none"> If not found → redirects to <code>/login</code> Task Fetching: <ul style="list-style-type: none"> Sends <code>GET</code> request to: <p>→ <code>http://localhost:4000/tasks?scrumId={scrum.id}</code></p> Users Fetching: <ul style="list-style-type: none"> Sends <code>GET</code> request to <code>/users</code> Filters users based on matching <code>assignedTo</code> fields from tasks
--	--	--

		<p>Task Status Update (Admins Only):</p> <ol style="list-style-type: none"> 1. A dropdown is rendered next to each task if the user is an admin. 2. Options: To Do, In Progress, Done 3. On change: <ul style="list-style-type: none"> • Triggers PATCH request to: <ul style="list-style-type: none"> → <code>http://localhost:4000/tasks/{taskId}</code> • Updates status and adds a new entry to the task's history. <p>HTML Structure:</p> <ol style="list-style-type: none"> 1. Section begins with <h3>: "Scrum Details for {scrum.name}" 2. Display a Tasks section using <h4>: <ul style="list-style-type: none"> • Use a ul to list tasks. • Each task shows: <ul style="list-style-type: none"> → Title → Description → Status → If admin, a dropdown (select) to update status • Display a Users section using <h4>: <ul style="list-style-type: none"> → Use a ul to list users. → Each item should include the user's name and email. <p>** Kindly refer to the screenshots for any clarifications. **</p>
--	--	---

5 VALIDATIONS

- All required fields must be fulfilled with valid data.
- When logging into the system all the fields must be filled.
- When adding a new scrum into the system all fields are mandatory to be filled.
- When adding a new user into the system all fields are mandatory to be filled.

6 CONSTRAINTS

- You should be able to press the “TAB” key and “SHIFT + TAB” to navigate from top field to bottom field and vice-versa.
- Once a user is logged in there should not be a “/login” url available until logged out.

7 MANDATORY ASSESSMENT GUIDELINES

1. All actions like build, compile, running application, running test cases will be through Command Terminal.
2. To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal.
3. This editor Auto Saves the code.
4. If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B-command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.
5. These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.
6. To test any Restful application, the last option on the left panel of IDE, you can find ThunderClient, which is the lightweight equivalent of POSTMAN.
7. This is a web-based application, to run the application on a browser, use the internal browser in the workspace. Click on the second last option on the left panel of IDE, you can find Browser Preview, where you can launch the application.

Note: The application will not run in the local browser

8. You can follow series of command to setup React environment once you are in your project-name folder:
- a. `npm install` -> Will install all dependencies -> takes 10 to 15 min
 - b. `npm run start` -> To compile and deploy the project in browser. You can press <Ctrl> key while clicking on localhost:8080/8081 to open project in browser -> takes 2 to 3 min
 - c. `npm run json-start` -> As we are using a json server to mimic our db.json file as a database. So, this command is useful to start a json server.
 - a. `npm run jest` -> to run all test cases and see the summary. It takes 5 to 6 min to run.
 - d. `npm run test` -> to run all test cases. **It is mandatory to run this command before submission of workspace** -> takes 5 to 6 min
9. You may also run “`npm run jest`” while developing the solution to re-factor the code to pass the test-cases.
10. Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on **“Submit Assessment”** after you are done with code.
11. You need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.