Task Manager Application (Frontend Only with Auth)

1. Project Overview

The Task Manager Application is a single-page application (SPA) that enables users to manage tasks with an integrated user authentication system. The application uses RxJS for reactive programming and state management, maintaining task data and user session state efficiently. All data is persisted in local storage to allow for session continuity.

2. Features

2.1 User Authentication

- Implement a login system where users can enter their credentials.
- Use RxJS Observables to manage authentication state and local storage to store user information.

2.2 Session Management

- Maintain user sessions by checking for the presence of a token in local storage.
- Redirect users based on their authentication status (e.g., logged in vs. logged out).

2.3 Authorization

- Implement route guards to protect certain routes from unauthorized access.
- Display different UI elements based on the user's authentication status.

2.4 Reactive State Management with RxJS

- Use RxJS Subjects and Observables to manage tasks and authentication states.
- Store tasks and user session data in local storage and synchronize with RxJS streams.

2.5 Local Data Storage

- Use local storage to persist tasks and user session data between sessions.
- Implement methods for adding, updating, deleting, and retrieving tasks from local storage using RxJS for reactive updates.

2.6 Custom Table and Pagination

- Create a custom table component to display tasks in a tabular format without third-party libraries.
- Implement pagination logic to navigate through large sets of tasks, allowing users to select the number of tasks per page and navigate between pages.

2.7 Task Search with Debouncing

- Implement a search feature that allows users to filter tasks. Use the `debounceTime` and `distinctUntilChanged` operators to optimize search input processing.

2.8 Standalone Components

- Create reusable components (e.g., TaskItem, TaskList, Login) as standalone components.

2.9 Routing and Lazy Loading

- Implement Angular Router for navigation.
- Use lazy loading to load feature modules (e.g., TaskModule) on demand.

2.10 Reactive Forms

- Utilize Reactive Forms for creating and managing the login and task forms.
- Implement form validation to ensure valid data input.

2.11 Custom Pipes and Directives

- Create a custom pipe to filter tasks based on their status (e.g., completed, pending).
- Implement custom directives for task highlighting.

2.12 Responsive Design

- Ensure the application is mobile-friendly using CSS Flexbox for layout.

2.13 Unit Testing

- Write unit tests for components and RxJS-related code using Jasmine and Karma to ensure quality.

2.14 Documentation

- Provide clear documentation within the project (README.md) detailing setup instructions, features, and usage.

3. Technical Stack

- Frontend: Angular 17, RxJS

4. Project Structure

bash

Copy code

```
/task-manager-app
├── /src
|   ├── /app
|   |   ├── /core
|   |   ├── /features
|   |   |   ├── /task
|   |   |   ├── /auth
|   |   ├── /components
|   |   |   ├── /custom-table
|   |   |   └── /task-item
|   |   ├── /services
|   |   └── app.module.ts
|   ├── /assets
|   ├── /environments
|   └── index.html
├── angular.json
├── package.json
└── README.md
```

5. Implementation Steps

5.1 Set Up Angular Project

- Create a new Angular project using the Angular CLI.

5.2 Create Auth Module

- Develop a login form component that accepts user credentials (e.g., username and password).
- Implement an authentication service that uses RxJS Observables to manage login/logout functionality.

5.3 Session Management

- Create a session management service that checks local storage for a token on app initialization.
- Use RxJS to dispatch changes to the authentication state.

5.4 Create Task Module

- Develop components for displaying the task list using a custom table component, task details, and task form.
- Implement task management functionalities (CRUD operations) using RxJS for state updates.

5.5 Implement Local Storage with RxJS

- Create methods for adding, updating, deleting, and retrieving tasks from local storage, using RxJS Observables to reactively update the UI.

5.6 Implement Task Search with Debouncing

- Add a search input to filter tasks and use `debounceTime` and `distinctUntilChanged` to limit the number of search operations

5.7 Implement Custom Pagination

- Create pagination logic within the custom table component to navigate through tasks, allowing users to select the number of tasks per page.

5.8 Implement Route Guards

- Create route guards to protect task-related routes, ensuring only authenticated users can access them.

5.9 Develop UI Components

- Design a responsive interface for both login and task management using plain CSS and Angular features.

5.10 Add Routing

- Set up Angular Router to navigate between the login page and the main task list, with guards for protected routes.

5.11 Implement Testing

- Write unit tests for authentication, session management, and RxJS-related code to ensure reliability.

6. Conclusion

This Task Manager Application incorporates user login, session management, and task management features while utilizing RxJS for effective reactive state management. It includes a search functionality with debouncing and distinct value checking to enhance performance, along with retry logic for robustness. The application effectively uses local storage for data persistence and showcases a custom table and pagination system for efficient task management. This project highlights Angular's latest features and best practices for building modern web applications and serves as a practical learning tool for developers interested in Angular and reactive programming with RxJS.