Urbio Technologies L.L.C.
Ontario Tower, Business Bay, Dubai, UAE
IBAN: AE080260001015852341601
Account: 1015852341601
EBILAEAD

Frontend Developer Test Project: Blog Dashboard

Project Overview

Build a simple blog dashboard using Next.js, Material-UI, TypeScript, and RTK Query.

The dashboard will display a list of blog posts fetched from a mock API, allow users to view post details, and add new posts.

Requirements:

1. Next.js Setup:

- Use Next.js to create the application.
- Implement server-side rendering (SSR) or static site generation (SSG) for the blog post list page.
- Use dynamic routing for individual blog post pages.

2. Material-UI (MUI):

- Use Material-UI components for the UI (e.g., cards, buttons, grids, typography).
- Implement a responsive layout that works on both desktop and mobile devices.
- Use MUI's theming system to customize the app's appearance (e.g., primary and secondary colors).

3. TypeScript:

- Define TypeScript interfaces for the blog post data structure.
- Ensure type safety across the application (e.g., props, API responses, state).

4. RTK Query:

- Use RTK Query to manage API calls for fetching and creating blog posts.
- Implement caching and automatic refetching for the blog post list.
- Handle loading and error states gracefully.

5. Functionality:

- Blog Post List Page:
  - Fetch and display a list of blog posts from a mock API.
  - Each post should display the title, author, and a short excerpt.
  - Add a "Read More" button that navigates to the individual post page.
- Blog Post Detail Page:
  - Display the full content of the blog post (title, author, body).
- Add New Post:
  - Include a form to add a new blog post (title, author, body).
  - Use RTK Query to submit the new post to the mock API and update the list.

Mock API

Use a free mock API service like JSONPlaceholder or MockAPI.io to simulate the backend. For example:

1. Fetch posts: GET /posts
2. Create post: POST /posts
3. Fetch single post: GET /posts/:id

Evaluation Criteria

1. Code Quality:

- Clean, modular, and reusable code.
- Proper folder structure and naming conventions.
- Use of TypeScript types and interfaces.

2. UI/UX:

- Responsive and visually appealing design using Material-UI.
- Smooth navigation and user interactions.

3. State Management:

- Efficient use of RTK Query for API calls and caching.
- Proper handling of loading and error states.

4. Performance:

- Optimized rendering (e.g., SSR/SSG for blog posts).
- Minimal unnecessary re-renders.

5. Bonus Points:

- Implement pagination or infinite scrolling for the blog post list.
- Add unit tests for components or API calls.
- Use Next.js API routes to create a custom backend for the blog posts.

Deliverables

1. A GitHub repository with the complete codebase.
2. A live demo of the application (e.g., hosted on Vercel or Netlify).
3. A README file with:
   - Instructions to run the project locally.
   - A brief explanation of the design decisions and challenges faced.

Timeframe

2 days