Mini-Project: Data Fetching in Next.js

Advanced User Management Dashboard using Next.js and reqres.in

**Title: Advanced User Management Dashboard using Next.js and reqres.in**

Scenario: You have been tasked with creating an advanced User Management Dashboard for a company that wants to manage and display its employee data. The dashboard should be built using Next.js and should utilize SSG, SSR, and SWR for data fetching, based on the requirements of different parts of the application. The application should have multiple pages and functionalities to test your understanding of various Next.js concepts.

**Objective: Create a Next.js application that includes the following features:**

1. A homepage that displays a list of users with their basic information (name, email, and avatar). This list of users should be fetched using SSG (getStaticProps), as the user data does not change frequently, and SSG allows for fast page loading and SEO benefits.
2. Implement pagination for the user list on the homepage, with each page displaying up to 10 users. Use dynamic routing to handle the pagination.
3. A user details page that shows more information about a specific user when their name is clicked on the homepage. This page should use SSR (getServerSideProps) to fetch the user's details, as this information might change more often and requires more up-to-date data.
4. A real-time activity feed that displays recent updates or changes to user profiles. The activity feed should be fetched using SWR, as it allows for real-time data fetching and updates without requiring a page refresh.
5. A user search functionality that allows searching for users by name. The search results should be fetched and displayed using SWR to provide real-time search results.
6. An admin page to manage user roles, which should be protected with authentication (you can use a simple client-side authentication mechanism for this task). The admin page should use SSR to ensure the latest user role data is fetched and displayed.
7. Use dynamic routing to create the user details page and handle the user ID parameter.
8. Style the application using CSS modules or any CSS-in-JS solution like styled-jsx or styled-components.

**Task:**

1. Set up the Next.js project with the required dependencies.
2. Implement the homepage (SSG) using getStaticProps to fetch the user list and display their basic information. Make sure to handle pagination using dynamic routing.
3. Create a user details page using dynamic routing and getServerSideProps (SSR) to fetch the user's details when their name is clicked on the homepage.
4. Implement the real-time activity feed using SWR to fetch and display recent updates or changes to user profiles.
5. Add a user search functionality that uses SWR to fetch and display real-time search results.
6. Create an admin page with authentication for managing user roles. Use SSR to fetch the latest user role data.
7. Implement client-side authentication for the admin page.
8. Style the application using your preferred styling solution (CSS modules, styled-jsx, or styled-components).
9. Bonus: Deploy your application to Vercel or any other hosting platform, and test its functionality.

This task serves as an extensive mini-project that tests your understanding of multiple Next.js concepts, including SSG, SSR, SWR, dynamic routing, authentication, and styling. By completing this task, you'll demonstrate your ability to combine these concepts in real-world app development.

**Approach:**

a high-level structure of the application and the main components. You can use this as a starting point and then create the necessary components and pages yourself.

1. **pages/index.js** - The homepage with the user list and pagination
2. **pages/users/[page].js** - The dynamic route for handling pagination
3. **pages/user/[id].js** - The user details page with dynamic routing based on the user ID
4. **pages/activity-feed.js** - The real-time activity feed using SWR
5. **pages/search.js** - The user search page with real-time search results using SWR
6. **pages/admin.js** - The admin page for managing user roles, protected with authentication
7. **components/UserList.js** - The component for displaying the list of users
8. **components/UserDetails.js** - The component for displaying user details
9. **components/ActivityFeed.js** - The component for displaying the real-time activity feed
10. **components/SearchBar.js** - The component for the user search functionality
11. **components/AdminPanel.js** - The component for managing user roles
12. **components/Layout.js** - The layout component for the application, containing the header, footer, and any other shared elements

For fetching data, use **getStaticProps** for the homepage (SSG), **getServerSideProps** for the user details and admin pages (SSR), and SWR for the activity feed and search pages.

Remember to style your components using CSS modules, styled-jsx, or styled-components, and implement client-side authentication for the admin page.

Once you have set up the application structure, you can create the required components and pages, fetch data using the appropriate methods, and style your application as per your preferences.