CODESTYLE

1. Project Structure

```
src/
├── app/
│   ├── (authentication)/
│   │   ├── forgot_password/
│   │   ├── signin/
│   │   ├── signup/
│   ├── (private)/
│   │   ├── alerts/
│   │   ├── coin/
│   │   ├── market/
│   ├── (public)/
│   │   ├── landing_page/
│   │   └── test/
├── components/
│   ├── Box/
│   ├── Container/
│   ├── Form/
│   ├── Heading/
│   ├── Logo/
├── layouts/
│   ├── private_page/
│   ├── public_page/
│   │   ├── Footer/
│   │   ├── Navbar/
├── libs/
│   ├── serverAction/
│   ├── serverFetch/
├── provider/
```

│ ├── AuthProvider/

│ ├── ThemeProvider/

├── types/

├── views/

│ ├── alert/

│ ├── authentication/

│ ├── coin/

│ ├── landing_page/

│ ├── market/

│ ├── userProfile/

2. **Overall Project Management Strategy:**

2.1. app/

This folder organizes the high-level application routes or pages. It's likely structured by **route groups** for different areas of the application.

- **Subfolders:**
    - (authentication)/: Handles user authentication. Includes specific routes for signing in, signing up, and password recovery.
    - (private)/: Contains routes for private or authenticated users, such as alerts, coin, and market. These might be features or dashboards only visible after login.
    - (public)/: Contains routes for publicly accessible pages like the landing_page and test.

Interaction:

- Routes from app/ might use components from the components/ folder for rendering UI and layouts from the layouts/ folder.
- They may rely on libs/ for server-side actions and provider/ for managing global states like authentication or themes.

2.2 components/

This folder is for reusable UI components. These components are modular and likely used throughout the application.

- **Subfolders:**
    - **Box/, Container/, Form/, Heading/, Logo/**: These could represent basic UI building blocks like layout wrappers, form elements, and typography.

Interaction:

- These components are imported by pages in app/ or higher-level UI sections in layouts/.
- For example, the Logo might be displayed in Navbar within the public_page layout.

2.3 layouts/

This folder organizes templates for the overall page structure (headers, footers, sidebars, etc.).

- **Subfolders:**
    - private_page/: Layout used for authenticated user pages (e.g., a dashboard).

- public_page/: Layout used for public pages. Includes Footer/ and Navbar/, likely shared across pages like the landing_page.

Interaction:

- Pages in app/ likely wrap themselves in these layouts.
- Components from components/ (e.g., Logo or Form) are used within the layouts.

2.4 libs/

Contains utility functions or logic for server-side actions and API calls.

- **Subfolders:**
  - serverAction/: Likely contains server-side methods for modifying data (e.g., creating or updating database entries).
  - serverFetch/: Likely contains methods for fetching data from the server or APIs.

Interaction:

- These utilities are imported by both the app/ routes and views/ for dynamic data handling.
- For example, fetching data for coin or market routes could involve serverFetch/.

2.5 provider/

This folder is for context providers that manage application-wide states.

- **Subfolders:**
  - AuthProvider/: Likely handles user authentication and authorization state using a context API.
  - ThemeProvider/: Likely manages the application's theme (e.g., light/dark mode).

Interaction:

- Providers wrap the application at a high level, ensuring all components and pages have access to global states like user authentication or theme settings.
- Layouts in layouts/ or pages in app/ could consume these contexts for dynamic behavior (e.g., showing user-specific content in private routes).

2.6 types/

A folder for storing TypeScript type definitions and interfaces.

- **Purpose:** Contains type definitions for objects used across the application (e.g., User, Market, Coin). Ensures type safety for data.

Interaction:

- Shared by all folders (app/, libs/, views/, etc.) to maintain consistent data structures and prevent runtime errors.

2.7 views/

Contains subfolders for specific features or modules.

- **Subfolders:**
  - alert/, authentication/, coin/, landing_page/, market/, userProfile/: Likely contain feature-specific components or logic. For instance:
    - coin/ might hold detailed components or helper functions specific to the coin-related pages or features.
    - userProfile/ might handle components for displaying and editing user profiles.

Interaction:

- These feature-specific components or modules are used by routes in app/ to render individual pages.
- May interact with utilities in libs/ for data fetching or actions.

How Everything Interacts:

1. **Global State & Utilities:**
    - provider/ manages state (authentication, themes).
    - libs/ handles API interactions (fetching or modifying data).
    - Both are consumed by app/ routes, layouts/, and views/.
2. **UI Building:**
    - components/ provides reusable building blocks.
    - layouts/ creates page templates using components/.
3. **Routing:**
    - app/ defines the main routes, leveraging layouts/ and views/ for the structure and logic.
    - Public and private routes are differentiated (e.g., landing_page vs. market).
4. **Type Safety:**
    - types/ ensures consistent and reliable data flow between all layers.

---

3. State Management

- Use React Context for global state
- Use React Query for server state
- Use local state for component-specific data

4 Styling

- Use Tailwind CSS for styling
- Follow BEM naming convention for custom CSS
- Use CSS modules for component-specific styles

5. Naming Conventions

- Use PascalCase for exported names (public) and for the file names.
- Use camelCase for internal names (private)