Jackpot Application - Setup and Architecture Overview

Table Of Content

Project Overview

Jackpot is a project creating a fast, smooth platform where users can easily play original games. It offers a user-friendly, reliable, and engaging experience with minimal load times, ensuring players can enjoy their favorite games seamlessly across devices.

How to Run the Program

1. **Install dependencies**
   Run the following command to install all necessary packages:

   ```
   yarn install
   ```

2. **Start the development server**
   Run the following command to start the app in development mode:

   ```
   yarn dev
   ```

3. **Accessing the app**
   Open your web browser and navigate to the following URLs to access the different parts of the app:

   - **Home Page:** http://localhost:3000/
   - **Provider Page:** http://localhost:3000/provider
   - **Favorite Page:** http://localhost:3000/favorite

Application Structure

The project follows an **Atomic Architecture** design pattern. This approach facilitates clean and maintainable code by breaking down the UI into small, reusable components such as atoms, molecules, organisms, and templates. This modular structure enhances scalability and ease of testing.

Technology Stack and Design Choices

1. State Management - Zustand

- **Why Zustand?**
  Zustand is a lightweight and minimalistic state management library. It reduces boilerplate often encountered in alternatives like Redux. Zustand's simple API allows quick implementation of global state management, which is ideal for this game app with less complexity.

2. Data Fetching & Caching - React Query

- **Why React Query?**
  React Query offers efficient client-side data fetching and caching mechanisms. It eliminates repetitive API calls by caching data and automatically re-fetching only when necessary, improving performance and user experience.

3. Styling - SCSS Modules

- **Why SCSS Modules?**
  SCSS modules provide encapsulated and locally scoped CSS. This avoids global namespace conflicts and style leaks, leading to predictable styling behavior across the app.

4. Type Safety - TypeScript

- **Why TypeScript?**
  TypeScript enforces static type checking at compile time. It helps catch errors early, improves code maintainability, and enhances developer experience with IDE support.

5. (.env) file

A `.env` file stores environment variables to configure app settings securely and flexibly outside the source code. (commiting in github for testing)

Packages

Packages	Purpose
@tanstack/react-query	Handles data fetching, caching, and synchronization for React apps.
axios	Makes HTTP requests to APIs or external data sources.
clsx	Conditionally joins and manages CSS class names.
next	Provides server-side rendering and routing for React applications.
react	Core library for building user interfaces with components.
react-dom	Renders React components to the DOM in web browsers.
react-intersection-observer	Detects when elements enter or leave the viewport.
sass	Adds support for SCSS syntax and advanced styling features.
zustand	Simplifies global state management with a minimal API.

Main Files and custom hooks

File/Folder	Purpose
hooks/PageHooks/*	Page-specific business logic (custom hooks)
hooks/queries/*	Fetching data with React Query (custom hooks)
store/actions.ts	Zustand actions – logic for updating state
store/types.ts	TypeScript types/interfaces for the store
store/gameStates.ts	Zustand logic for managing game filters
services/	API service abstraction layer (network requests)
types/	Global app type definitions
baseLocalization/baseLocalization.ts	For keep tracking all the Language related Strings
constants/constants.ts	All the constants are in one place
globalStyles/variables.ts	Device related size
globalStyles/colors.ts	Reusable Colors
helper/categorizeGamesByCategory.ts	groups games by their valid categories
utils/debounce.ts	Debounce Function for input

Key Considerations

There are a few things to keep in mind regarding the current implementation:

- The API does not return consistent or accurate data when using the provider parameter to filter results. This may lead to missing or incorrect games on the Provider-specific screen.

- When applying the order parameter (either ascending or descending), the API response is inconsistent and may not reflect the intended sorting.

- The "Favorite" feature uses Zustand with persistence to store user favorites locally; sorting and searching are handled directly within the saved state, rather than relying on the API.

- The application has three main routes:
    - The Home page.
    - When a user clicks on any provider, they are navigated to a dedicated provider screen.
    - The Favorite page, accessed from the category section, which lists all favorite games.

Folder Structure

Used the atomic folder structure, which helped me keep track of everything. Here is the basic folder structure.

```
src/
├── app/
│   ├── lib/
│   ├── provider/
│   ├── favorite/
│   ├── favicon.ico
│   ├── globals.css
│   ├── layout.tsx
│   ├── page.module.scss
│   └── page.tsx
│
├── assets/
│
├── baseLocalization/
│
├── components/
│   ├── atoms/
│   ├── molecules/
│   ├── organisms/
│   └── templates/
│       ├── HomeTemplate/
│       ├── ProvidersTemplate/
│       └── FavoriteTemplate/
│
├── constants/
│
├── globalStyles/
│
├── helper/
│
├── hooks/
│   └── PageHooks/
│
├── services/
│
├── store/
│
├── types/
│
├── utils/
```