React Native with React Query Project Structure

TanStack Query (React Query) for server state. Caching, mutations, and automatic refetching.

#react-native #expo #react-query #tanstack #api #state

PNG PDF Copy Prompt

Project Directory

```
my-app/
  app.json
  package.json
  tsconfig.json
  babel.config.js
  .gitignore
  app/
    _layout.tsx QueryClientProv…
    index.tsx
    (tabs)/
      _layout.tsx
      index.tsx
      posts.tsx Uses useQuery
    posts/
      [id].tsx Post detail wit…
  api/ API functions
    client.ts Axios/fetch ins…
    posts.ts Post API functi…
    users.ts
  hooks/ Query hooks
    queries/
      usePosts.ts useQuery wrapper
      usePost.ts
      useUser.ts
    mutations/
      useCreatePost.ts useMutation wra…
      useUpdatePost.ts
  lib/
    queryClient.ts QueryClient con…
    queryKeys.ts Centralized que…
  types/
    api.ts API response ty…
    post.ts
  components/
    PostCard.tsx
    LoadingSpinner.tsx
    ErrorView.tsx
  constants/
    api.ts API URLs
  assets/
```

Why This Structure?

TanStack Query manages server state separately from UI state. Automatic caching, background refetching, stale-while-revalidate, and mutation handling. Stop manually tracking loading/error/data—let React Query handle it.

Key Directories

`api/` - Raw API functions (no hooks), return promises

`hooks/queries/` - useQuery wrappers for fetching data

`hooks/mutations/` - useMutation wrappers for creating/updating

`lib/queryClient.ts` - QueryClient with default options

`lib/queryKeys.ts` - Factory functions for consistent cache keys

Query and Mutation Hooks

```
// hooks/queries/usePosts.ts
export function usePosts() {
  return useQuery({
    queryKey: queryKeys.posts.all,
    queryFn: () => api.getPosts(),
    staleTime: 1000 * 60 * 5, // 5 min
  });
}

// hooks/mutations/useCreatePost.ts
export function useCreatePost() {
  const queryClient = useQueryClient();
  return useMutation({
    mutationFn: api.createPost,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: queryKeys.
    },
  });
}
```

Getting Started

1. `npx create-expo-app@latest my-app`
2. `npx expo install @tanstack/react-query`
3. Create `lib/queryClient.ts` with defaults
4. Wrap app in `QueryClientProvider`
5. Create query hooks in `hooks/queries/`

When To Use This

- App fetches data from APIs
- Need caching and background refetching
- Want optimistic updates for mutations
- Tired of manual loading/error state tracking
- Building data-heavy applications

Trade-offs

`Another dependency` - Adds ~30KB to bundle

`Learning curve` - Query keys, stale time, etc. to learn

`Overkill for simple apps` - Few API calls may not need it

Best Practices

- Centralize query keys in `lib/queryKeys.ts`
- Keep API functions pure—no hooks, just promises
- Wrap useQuery in custom hooks for reuse
- Set sensible `staleTime` defaults
- Use `queryClient.invalidateQueries` after mutations