MarsBased React Style Guide

Until now, we've used create-react-app to bootstrap React applications and we follow its conventions.

1. Do’s and Don’ts

1.1. Use typescript if possible

1.2. Use a generator to bootstrap the project

A project generator saves a lot of boilerplate work and provides common conventions.

We’ve used create react app for all our projects.

We will consider using another system (mostly next.js) for newer projects, especially if they have important SEO requirements.

1.3. Write functional components

Class components will be deprecated. For new components always use functions.

1.4. Use hooks for state management

- In general, prefer React’s hooks for local stage management
- Use a library (with hooks) for global state management
- Avoid redux if possible (too complex)

See patterns (below) for examples and usages.

1.5. Use a declarative API library

It reduces boilerplate code a lot.

We currently use:

- REST: react-query
- GraphQL: apollo-client

1.6. Do use function declarations

For a better readability.

```
// Don't declare arrow functions
const App = () => (
  <div>
    <Logo />
  </div>
);

// DO declare functions
function App() {
  return (
    <div>
      <Logo />
    </div>
  );
}
```

1.7. Do name exports

It allows to export multiple values and it encourages the use of the same naming.

```
export function LoginPage() {
  ...
}
```

1.8. Do inline type props

It reduces the external dependencies of the function and therefore is more easy to move or change it without errors.

```
// DO inline type props
export function LoginForm({user = ''}: {user: string}) {
  ...
}
```

2. General project organization and architecture

We follow https://create-react-app.dev/docs/folder-structure/

For small to medium projects this is recommended:

- One entry point: `src/index.ts`
- One config file: `src/config.ts`
- One router: `src/Router.tsx`
- Declare all app routes at `src/routes.ts` (see below)
- All page components under `pages/` (can be nested to mimic routes path hierarchy)
- All non-page components under `components/` (can be nested)
- All hooks under `hooks/` (can expose Providers)
- One folder for each (external) service. For example `api/` (for rest APIs), `graphql/` for GraphQL or `auth/` for authorization service. They can include type definitions, data transformations, clients or anything related to that service and communication with it.
- One folder for locales: `locales/`
- The rest: utility functions, helpers, etc… under `lib/` (keep it clean, please)

2.1. Project structure example

```
src/
|- index.ts              # entry point
|- config.ts             # application configuration
|- routes.ts             # Application route definitions
|- App.tsx              # Application setup (providers)
|- Router.tsx           # Application router
|- pages/               # page components (access to router)
  |- HomePage.tsx
  |- UserListPage.tsx
  |- UserDetailPage.tsx
  |- admin/             # try to mimic the actual client urls
    |- AdminUserListPage.tsx
|- components/
  |- Layout.tsx         # Basic layout for app
  |- Spinner.tsx        # shared component
  |- Tag.tsx            # shared component
  |- posts/             # folder for specific areas, pages or sections
    |- PostForm.tsx
  |- users/
    |- UserCard.tsx
|- hooks/
   |- useUser.tsx       # hook
   |- useXXX.ts         # another kook
|- locales/
   |- type.d.ts         # Locale type definitions
   |- en-GB.ts          # locale for en-GB
|- api/                 # REST API folder (optional)
   |- index.ts
   |- types.d.ts        # API Entities type definitions
   |- client.ts         # API client
|- graphql/             # GraphQL folder (optional)
   |- types.d.ts
   |- schema.ts
|- auth/                # Authorization service (optional)
   |- types.d.ts
   |- index.ts
|- lib/                 # internal libraries aka "Everything else"
  |- randomColor.ts
|- react-app-env.d.ts   # declare missing types from npm packages
```

If using a repo for both api and client, put the above inside `client/` folder

2.2. References (project structure)

- Route definitions idea taken from Redwood framework

3. Description of the most common patterns used to solve common problems

3.1. State management

- In general, prefer hooks over any other solution
- You-don-t-need-redux. But if so, use hooks interface

3.1.1. Local state management

React hooks are enough most of the time.

3.1.2. Global state management

Prefer zustand over React contexts or redux.

See zustand documentation.

3.2. External services

- Create a clean interface for each service. For example: `src/api/index.ts` (functions to send http requests to a REST API) or `src/graphql/index.ts` (queries and mutations of a GraphQL endpoint)
- Create a file with the types (models): `<service-name>/types.ts`
- Use declarative data fetching: prefer `useQuery` over `fetch` (available both in Apollo client and react-query)

3.3. GraphQL

We currently use apollo-client

Try to generate types and code as much as possible.

In general, keep all graphql related code inside `graphql/` folder.

3.4. REST

- A single `src/api/index.ts` exports all possible API interactions
- Use `src/api/types.ts` to declare API entity type definitions
- Client specific functionality inside `src/api/client.ts` (like, for example)
- If Auth and API are different services, is common to have two folders (`src/auth` and `src/api`) and the API depends on authorization (JWT tokens, for example). If auth and API are in the same service, the `src/auth` folder can be omitted.

3.5. Routing (only create-react-app)

Custom routing is only required with create react app (next.js has its own routing standards and patterns).

- Use react-router-dom with hooks
- Create a route definitions file `src/routes.ts` with all route paths

- It helps to mimic the actual routes in the `/pages` folder

3.5.1. Page vs component

A Page is a component that:

- It is used in `Router.tsx`
- Can access route parameters (like in: `/post/:id`)
- Lives inside `src/pages/` folder (can be nested to reflect actual url structure)

3.5.2. Route definitions

It's a file to generate route paths. Advantages:

- You get an overview of all available routes on the app
- It helps to prevent errors when declaring routes
- Route completion via editor

```
export default {
  posts: () => `/posts`,
  post: (id: string) => `/posts/${id}`,
  admin: {
    users: () => `/admin/users`,
  },
};
```

Usage:

```
import routes from "./routes";

<Link to={routes.users()}>Users</Link>;
```

3.5.3. Router.tsx

Use the `routes` definitions to create the routes placeholders:

```
<Router>
  <Switch>
    <Route exact path={routes.posts()}>
      <PostsListPage />
    </Route>
    <Route exact path={routes.post(":id")}>
      <PostPage />
    </Route>
    <Route exact path={routes.admin.users()}>
      <AdminUsersListPage />
    </Route>
  </Switch>
</Router>
```

3.5.4. Access route parameters using hooks

For a route like `/posts/:postId/comments/:commentId` we use the following code to access route params:

```
// In next version of react-router types won't be required
const { pageId, commentId } = useParams<Record<string, string>>();
```

3.6. Testing

Follow React guidelines: https://reactjs.org/docs/testing.html

- Favour end-to-end-testing over components test
- More important to cover critical paths than general coverage
- Ensure business logic are pure functions and write unit tests for them when needed

When we write tests, we use Cypress

4. Libraries

4.1. Recommended libraries

- Internationalization: react-intl
- Forms: react-hook-form
- Global state management: zustand
- React query state: react-query
- Http: ky
- GraphQL API: apollo-client
- Routing: react-router-dom

4.2. Other libraries we have used

- Styling
  - tailwindcss
  - styled-components
  - xstyled
- Components
  - antd
- Http
  - axios
- Hooks:
  - react-use

4.3. Libraries worth taking a look into

- State management
  - jotai
  - recoil

- Component library
  - Chakra UI

4.4. References (libraries)

- Blog: Internationalize React apps done right

5. Learning resources

- React docs are quite good. Recommended reading: https://reactjs.org/docs/hello-world.html
- egghead.io is one of our favourite places to learn and Kent C. Dodds is a master, so this can't fail: https://egghezad.io/courses/the-beginner-s-guide-to-reactjs
- To learn Redux, a course at egghead by the creator of Redux itself is a MUST: https://egghead.io/courses/getting-started-with-redux
- This tutorial is quite good for starting with React: https://www.fullstackreact.com/30-days-of-react/
- https://www.reddit.com/r/reactjs/