Contents

[**Project Structure** 2](#_Toc81330188)

[Aliases for imports 2](#_Toc81330189)

[Using SASS 2](#_Toc81330190)

[**Naming Conventions** 3](#_Toc81330191)

[File Naming 3](#_Toc81330192)

[Variable Naming 3](#_Toc81330193)

[Git branch 3](#_Toc81330194)

[**Coding best practices** 4](#_Toc81330195)

[State, Props & Lifecycle 4](#_Toc81330196)

[Handling Events 4](#_Toc81330197)

[Conditional Handling 4](#_Toc81330198)

[List & Keys 4](#_Toc81330199)

[Optimizing performance 4](#_Toc81330200)

[Refs & DOM 5](#_Toc81330201)

[Higher Order components 5](#_Toc81330202)

[General 5](#_Toc81330203)

[ES6 Feature 5](#_Toc81330204)

[**Unit Tests** 5](#_Toc81330205)

[**Redux** 6](#_Toc81330206)

[ Do Not Mutate State 6](#_Toc81330207)

[ Reducers Must Not Have Side Effects 6](#_Toc81330208)

[ Do Not Put Non-Serializable Values in State or Actions 6](#_Toc81330209)

[ Only One Redux Store Per App 6](#_Toc81330210)

[ Structure Files as Feature Folders with Single-File Logic 6](#_Toc81330211)

[ Put as Much Logic as Possible in Reducers to calculate new state. 6](#_Toc81330212)

[ Name State Slices Based On the Stored Data 6](#_Toc81330213)

[ Organize State Structure Based on Data Types, Not Components 6](#_Toc81330214)

[ Keep State Minimal and Derive Additional Values 6](#_Toc81330215)

[ Model Actions as Events, Not Setters 6](#_Toc81330216)

[ Write Meaningful Action Names 6](#_Toc81330217)

[ Allow Many Reducers to Respond to the Same Action 6](#_Toc81330218)

[ Avoid Dispatching Many Actions Sequentially 6](#_Toc81330219)

[**Security** 6](#_Toc81330220)

[Best Practices 6](#_Toc81330221)

# **Project Structure**

## Aliases for imports

Aliasing our app and environments folders will enable us to implement clean imports which will be consistent throughout our application.

Import statements looking something like

Text

Description automatically generated

should be used as below

Text

Description automatically generated with medium confidence

make sure you add configuration in tsconfig.json as below

Text

Description automatically generated

## Using SASS

SASS is a styles preprocessor which brings support for variables, functions, mixins etc.

The global styles for the project are placed in a styles folder. Any new component styles can be added under this folder.

**styles.scss** file imports individual component styles.

# **Naming Conventions**

## File Naming

|  |  |  |
| --- | --- | --- |
| Type | Convention | Example |
| Component | <PascalCase>.tsx/jsx | UserLogin.tsx |
| Services | <camelCase>.ts/js | userService.ts |
| CSS | <PascalCase>.css/scss | CookieBanner.css |
| Spec File | <PascalCase>.spec.tsx/jsx | UserLogin.ts/js |

## Variable Naming

|  |  |  |
| --- | --- | --- |
| Type | Convention | Example |
| Property | lowerCamelCase | userInfo |
| Const, let | lowerCamelCase | isLoggedIn |
| CONSTANT Application level | UPPER\_SNAKE\_CASE | USER\_TOKEN |
| Attributes | camelCase | onClick |

## Git branch

|  |  |
| --- | --- |
| Type | Example |
| Feature | feature/ticketId-description |
| Bugfix | bugfix/ticketId-description |
| Test Cases | tests/ticketId-description |

# **Coding best practices**

## State, Props & Lifecycle

* free up resources taken by resource when they are destroyed.
* do not modify state directly - use a function
* neither parent nor child components should know if a certain component is stateful or stateless.
* Avoid mutating state when working with arrays
* Treat props as read-only. Do not try to modify them

## Handling Events

* Binding in the constructor to avoid performance problems of re-rendering

## Conditional Handling

* For a component to hide itself return null from render.

## List & Keys

* elements generated inside a map() call need key.
* keys within arrays should be unique among their siblings.

## Optimizing performance

* instead of writing shouldComponentUpdate() inherit from React.PureComponent wherever possible
* avoid, in-place mutation by using Object.assign, spread operator (...), and non-mutating operations
* use small components as much as possible
* render lists in dedicated components
* don't use array indexes as keys

## Refs & DOM

* when to use refs:
  + managing focus, text selection, or media playback.
  + triggering imperative animations.
  + integrating with third-party DOM libraries.

## Higher Order components

* Use Higher order components where appropriate
* pass unrelated props through to the wrapped component
* wrap the display name for easy debugging
* don't use HOCs inside the render method, create as variables outside
* apply HOCs outside the component definition so that the resulting component is created only once.

## General

* Use optional chaining if things can be null
* Create PURE functions and avoid side-effects
* Remove all console.log
* No unneeded comments
* Methods that are longer than the screen should be refactored into smaller units
* Commented out code should be deleted, not committed.

## ES6 Feature

* Prefer const. Use it for all variables whose values never change.
* Otherwise, use let – for variables whose values do change.
* Avoid var.

# **Unit Tests**

* Name test specification files the same as the component they test.
* Name test specification files with a suffix of .spec. Example user.spec.ts.
* If your mocks/stubs are being used multiple places then create separate folder mocks and keep all mocks there.
* Don't test more than one thing in a test
* No logic should exist within your test code.
* Code that needs to talk to a network, or, database is mocked.

# **Redux**

### Do Not Mutate State

### Reducers Must Not Have Side Effects

### Do Not Put Non-Serializable Values in State or Actions

### Only One Redux Store Per App

### Structure Files as Feature Folders with Single-File Logic

### Put as Much Logic as Possible in Reducers to calculate new state.

### Name State Slices Based On the Stored Data

### Organize State Structure Based on Data Types, Not Components

### Keep State Minimal and Derive Additional Values

### Model Actions as Events, Not Setters

### Write Meaningful Action Names

### Allow Many Reducers to Respond to the Same Action

### Avoid Dispatching Many Actions Sequentially

* Use the Object Shorthand Form of mapDispatch with connect

# **Security**

## Best Practices

* Keep application with the latest library releases.
* Avoid using APIs marked in the documentation as “*Security Risk*.”
* Avoid using deprecated API’s.
* Custom Error Page for Error Handling. Do not disclose sensitive information in error responses, including system details, session identifiers or account information
* Proper Authentication and Session Management.
* Role based Security to protect client-side resources.
* Logging and Auditing- Do not log sensitive data.
* Avoid direct DOM manipulation.
* Specify proper character sets, such as UTF-8, for all sources of input.
* Proper input data validation for length, type of data, data range & allowed characters wherever applicable.
* Use proper HTTP methods to pass data to server for example-Use only HTTP POST requests to transmit authentication credentials.
* On Logout functionality all the session related information including local, or session storage should be completely removed.
* Use of encryption/decryption technique.