This code convention is mostly based on Airbnb React/JSX Style Guide for React UI elements and React/Redux style guide for organizing store and connecting React Containers to store.

Basic React containers / components:

1. Naming
   1.1. For declaring and referencing React components use PascalCase and camelCase for their instances
   1.2. For Higher-order Component use prefix ‘with’ for external function and ‘With’ for internal class or function
   1.3. For JSX file names use PascalCase, for ts/js file names use camelCase
   1.4. For JSX props use camelCase and avoid using DOM component prop names
   1.5. For declaring Typescript interfaces use PascalCase, for all JS instances use camelCase
   1.6. Event handlers should begin with handle and end with the name of the event they handle
2. Alignment
   2.1. If props fit in one line, keep JSX tag in the same line, otherwise an each prop should be placed in the new own line
   2.2. In conditional rendering, JSX tag can stay on same line as condition if it fits one line, otherwise use parentheses to wrap it and place it in a new line
3. Quotes
   3.1 Use double quotes for JSX attributes, and single quotes for all other JS
4. Refs
   4.1. For refs use ref callbacks in element, and declare ref as React.createRef()
5. Tags
   5.1. Always use self-close tags for elements that have no children
   5.2. If component has multi-line properties, close its tag on a new line
6. Methods
   6.1. In most cases use arrow functions to handle events in props, except parts where it can hurt performance and can be replaced with standard function.
   6.2. As class methods use arrow functions over binding functions in constructor
7. Import order
   7.1. To keep consistent import section, import modules in following order:
      - Libraries
      - Theme
      - Components
      - HOCs
      - Consts
      - Services
      - Store
      - Types
      - Utils

Connecting React to Redux:

1. Organization
   1.1. React components are separated in two groups based on their connection with Redux. Components that are aware of redux and are connected to Redux State are separated to src/containers subfolder, Components that rely only on passed props or/and made to be reusable in containers or other components are placed in src/components subfolder.
   1.2. We split containers and components into domains (entities) based on feature they are implementing, like /containers/Auth contains all containers that handle user authentication and /components/Form contains all components that are used to create UI Forms
   1.3. For containers index.ts file is used to perform a connection process to Redux State, and .tsx file contains main layout of container page. Layout should be very simple and basic with all complex parts imported as components
2. Redux
   2.1. Co-locate reducers, actions, action-types based on the feature they implement. This keeps us focused on the managing a slice of application state, making for better reuse across pages/components. Grouping by file type, ie. src/actions, src/reducers, src/action-types, src/selectors, etc. doesn't scale well for large applications, is less reusable because files related to a slice of state are spread out over the filesystem; and making changes means developers end up having to edit multiple files all over the filesystem as well.
   2.2. As store is sliced for simpler parts, we use Higher Order Reducers to combine reducers to create instance specific reducers for each slice of state
   2.3. Use string constants instead of inline strings for action types, all action types constants should be declared within actionTypes.ts file
   2.4. For accessing store from connected components we use selectors to maintain better reusability while keeping all store related functions in one place
   2.5. Reducers should be pure functions and work on the assumtion that state is immutable. Reducer is intended to accept a state along with action and return a completely new state
   2.6. Try to keep your state shape flat; and normalize data where possible
3. Naming
   3.1. Action types - use constants, [VERB]_[VERB], eg. FETCH_USER or UPDATE_MATERIAL_REQUEST
   3.2. Action creators [verb][Noun]() eg. fetchCurrentUser(), acceptMaterialRequest()
   3.3. Selectors - get[NOUN]() or select[NOUN]() eg selectActiveRows()
4. Utils
   4.1. Group related util functions under a common name. ie, async helper functions might be in src/utils/async.js
   4.2. Expose each function as named export.
   4.3. Don't use a default export.
   4.4. Util functions must be pure.
   4.5. Util functions should be reusable, but have a single purpose.

Styling components

1. Styled-components
   1.1. Use styled-components styled wrapper for most cases when styling components or creating styles based on variable values
   1.2. Styles declared with styled-components should be positioned after import section and before TS Interfaces sections, except Interfaces that describe their properties
   1.3. Theme prop should be always extracted from properties and not imported directly, while using in styled-components style
2. StyleSheet
   2.1. Use StyleSheet styles in cases where styles are passed with prop different than style, when using animated values and functions (e.g. interpolation) or to describe small number of styles
   2.2. Styles declared with StyleSheet should be positioned after class or component in the end of the file but before HOC wrappers if present.