Saurabh Shah

Follow 57 Followers About

React Native — Application architecture with Design pattern — corporate standards

Saurabh Shah Aug 30, 2020 · 9 min read ★

Photo by Xiong Yan on Unsplash

awesome cross-platform mobile app.

I've also written a story for **Industry standard React-Native App setup**. (https://medium.com/@saurabhshah23/perfect-react-native-application-setup-industry-standards-bd3cc623745e)

NOTE: For versions of react and other libraries, please refer before and after screenshots of "package.json".

If you are creating a new application then, the prerequisite will be to create a react-native app using react-native-cli. Please refer to "https://reactnative.dev/docs/environment-setup" for steps to create a basic react native app and run it on emulator/device.

Overview of app architecture features:
This app will support all the major features required for a corporate application. It includes redux + thunk for centralized state, i18n for internationalization or multilingual support, react-native-testing-library for unit testing and paper for Material design component library.
You can skip any of them if it is not required in your application. They all are loosely coupled.

Agenda of this document is to cover details of:

> ✓ *all the dependencies used by a react-native application*
> ✓ *reasoning behind each dependency*
> ✓ *dependency configuration*
> ✓ *app integration steps for each dependency, if any.*

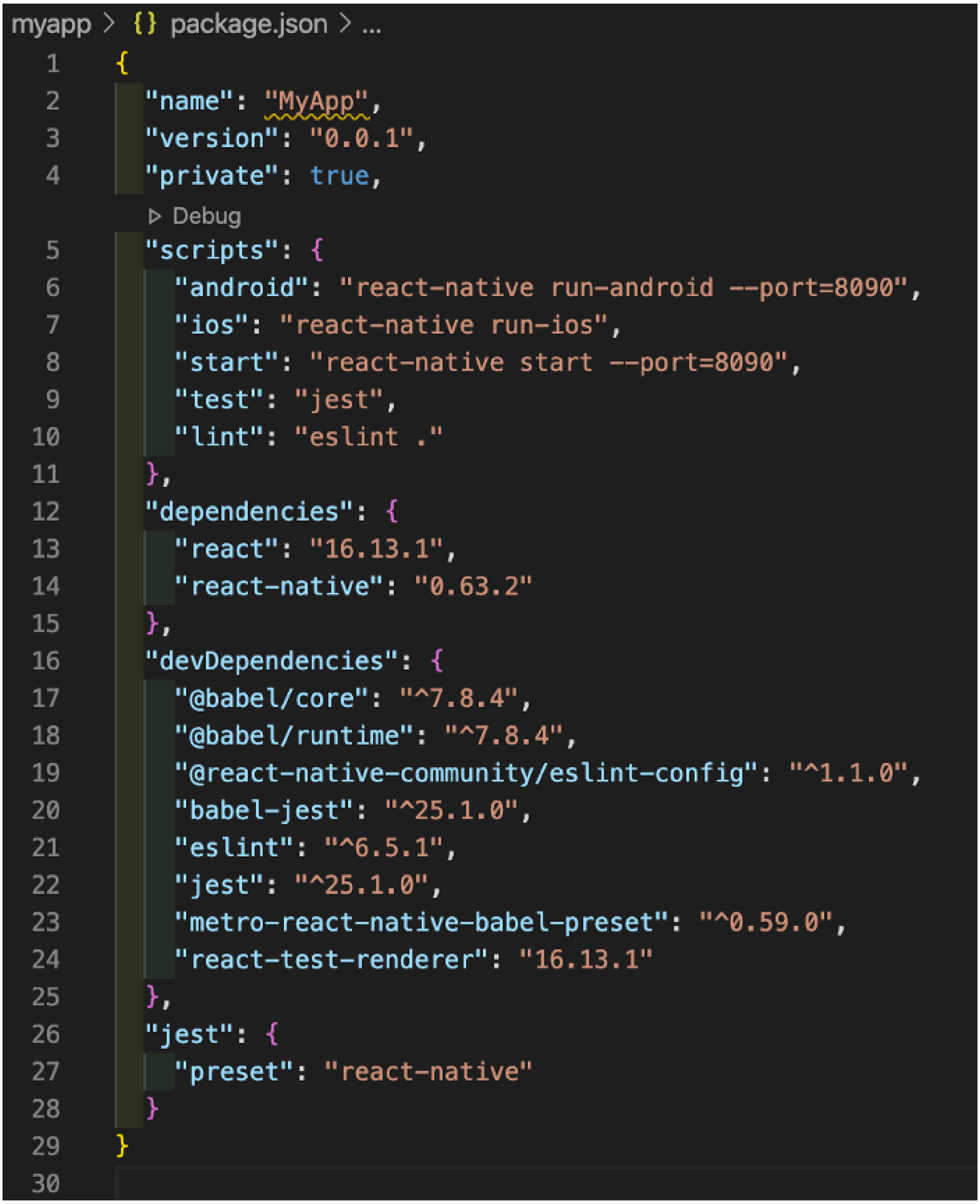
The document is divided in below 3 sections:

Section-1 Folder Structure & Architecture

Section-2 Design Pattern

Section-3 Architecture Dependencies

Section-1 ARCHITECTURE



myapp > {} package.json > ...

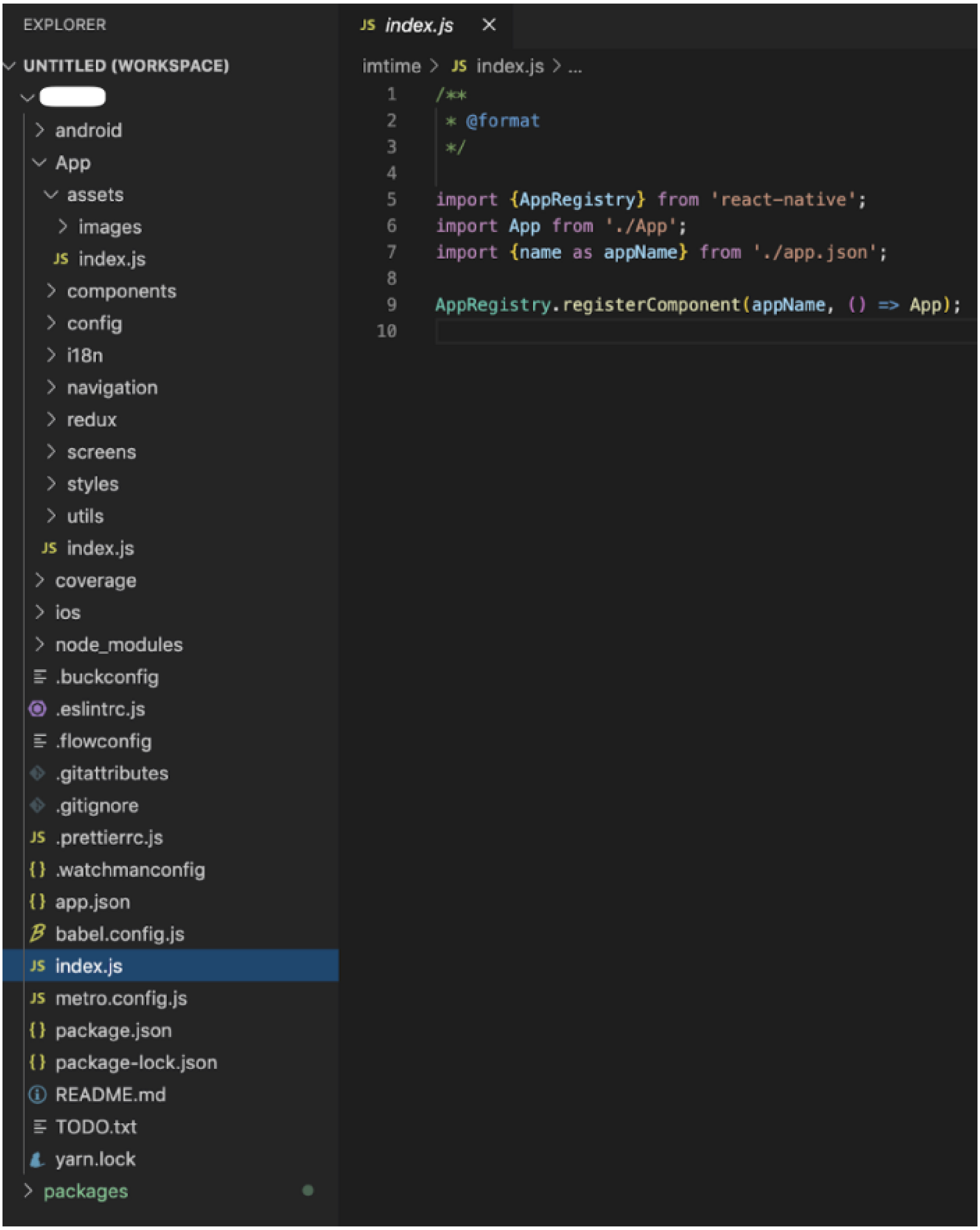
```
{
  "name": "MyApp",
  "version": "0.0.1",
  "private": true,
  ▷ Debug
  "scripts": {
    "android": "react-native run-android --port=8090",
    "ios": "react-native run-ios",
    "start": "react-native start --port=8090",
    "test": "jest",
    "lint": "eslint ."
  },
  "dependencies": {
    "react": "16.13.1",
    "react-native": "0.63.2"
  },
  "devDependencies": {
    "@babel/core": "^7.8.4",
    "@babel/runtime": "^7.8.4",
    "@react-native-community/eslint-config": "^1.1.0",
    "babel-jest": "^25.1.0",
    "eslint": "^6.5.1",
    "jest": "^25.1.0",
    "metro-react-native-babel-preset": "^0.59.0",
    "react-test-renderer": "16.13.1"
  },
  "jest": {
    "preset": "react-native"
  }
}
```

fig-1 default app package.json

Let's first restructure the application code to make it more robust and scalable. This will make it easy for on-boarding new resources by defining a common approach for the development process.

. . .

Please find below the structure of the app and all the changes explained below:



```
/**
 * @format
 */

import {AppRegistry} from 'react-native';
import App from './App';
import {name as appName} from './app.json';

AppRegistry.registerComponent(appName, () => App);
```

/index.js :

- It is the default entry point of every react-native application. There are no changes in this file at all.

/App/... :

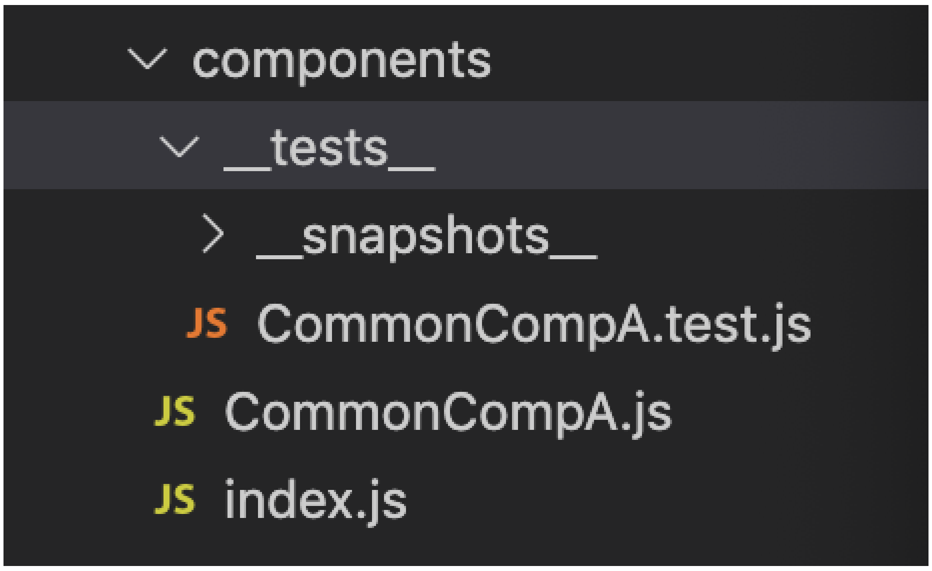
- This will be the core application source code.
- "/App/index.js" file is the container of our app and served as the entry point.

Let's dive-in to understand the rationale behind this structure:

/assets :

- As the name suggests, all the static assets should reside here.
- Each asset should be registered and exported from the /index.js
- Thus, all assets will be accessible and imported from '/assets'

/components:



```
components
  __tests__
    __snapshots__
    CommonCompA.test.js
  CommonCompA.js
  index.js
```

- Only shared components used across features are placed here.
- All the components should be registered and exported from /index.js for a single access point.
- All the components should bear named export. This will avoid any conflicts.
- Components that consist of complex logic or redux integration, can be further de-structured into "ComponentContainer.js" & "ComponentView.js" as per the "Container-View pattern" (this will be covered ahead in /screens part)

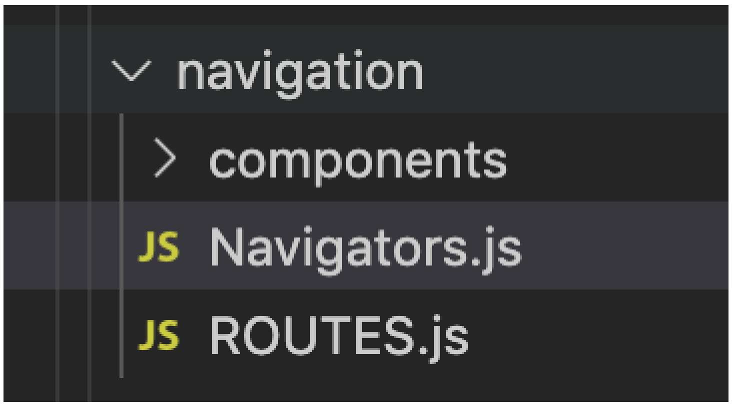
/config :

- All the app's configurations are to be kept at this path.

/i18n :

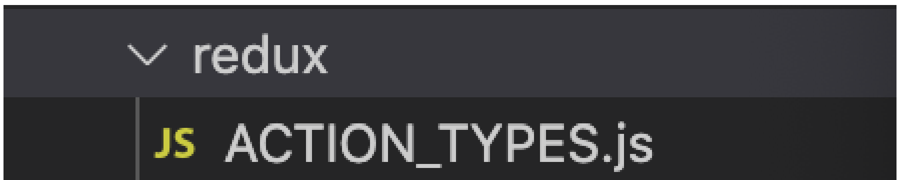
- Internationalisation or multi-lingual support is achieved by the use of the “i18n” library.
- It mainly consists of a configuration file and all the language translations in independent language.json files.
- (More on react-native-i18n implementation in “Section-3 Architecture Dependencies” below)

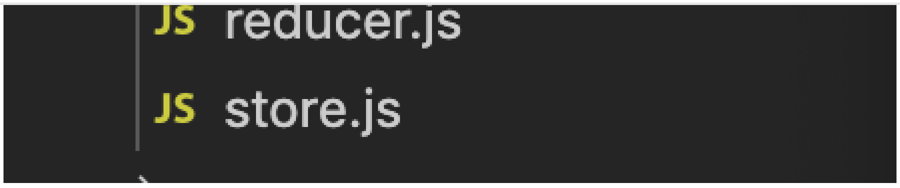
/navigation :



- As the name suggests, all the routing logic resides here.
- This app contains only one stack of navigation. Although, most of the apps will have minimum 2 navigators; viz before and after authentication.
- Ideally, all different navigators should be re-factored in separate files and then used in “Navigator.js”.
- ROUTES.js consists of all the constants for various available routes within our app.
- /components directory will hold all the navigation specific components like headers, title bars, action buttons, like so.
- (More on react-navigation implementation in “Section-3 Architecture Dependencies” below)

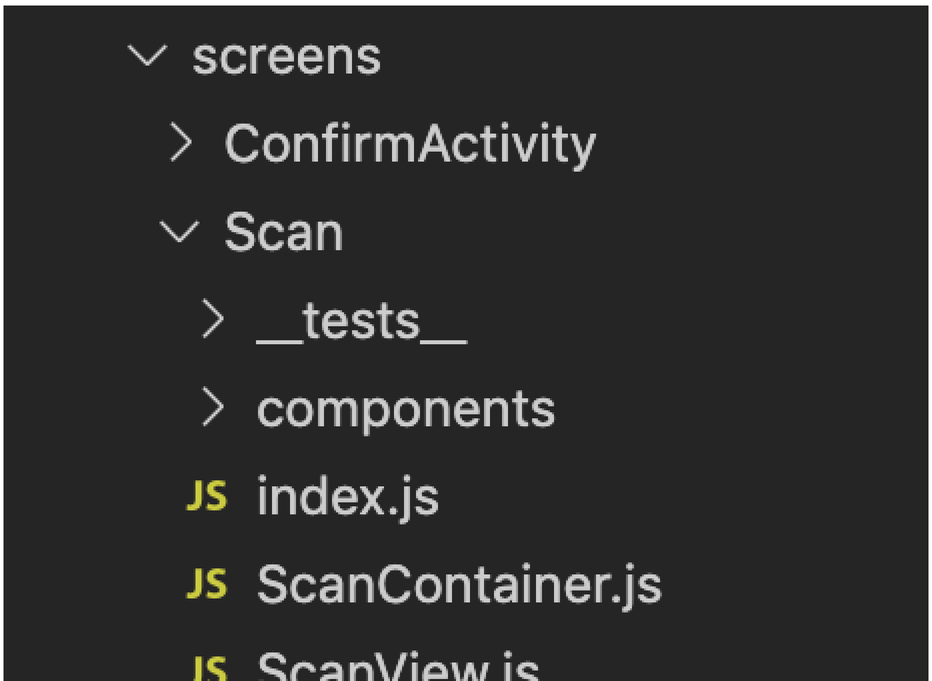
/redux :

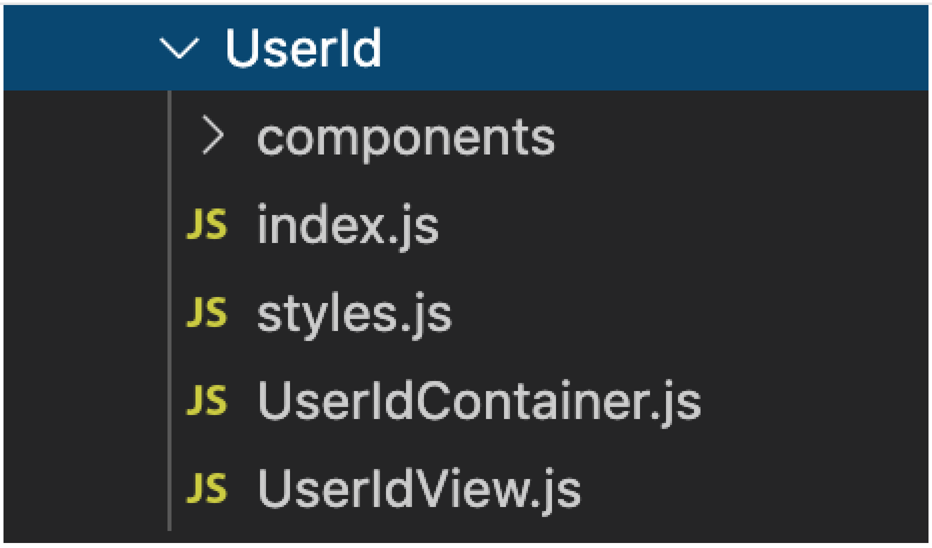




- It holds all the redux resources at one place.
- This includes action creators, reducers and store of our app.
- **CONSTANTS.js** has all the action types.
- /**actions.j**s consist of all the action-creators. Considering the demo app, there is only 1 action-creator, it can be broken into multiple files and can be into distributed locations feature/screen wise.
- /**reducer.js** reduces all the actions to store. Same applies for reducers. For a wider scope of app, this can become an app level root-reducer which merges various feature-level reducers using redux's combineReducers function.
- /**store.js** is the central store of the application. This incorporates all the mapping between reducer, store and middle-wares if any.
- We have a **redux-thunk middleware** in our app for enabling asynchronous dispatching of actions.
- (More on redux, react-redux, redux-thunk implementation in "Section-3 Architecture Dependencies" below)

/screens :




```
UserId
  components
  index.js
  styles.js
  UserIdContainer.js
  UserIdView.js
```

- This is the heart of our application.
- All the various features/screens/pages are defined here. In this case, "Scan", "UserId" and "ConfirmActivity" are 3 different screens of this demo app.
- Each screen consists of an index.js file which exports the screen's container as default module which makes the screen available to be utilised as a component.

/services :

- Services are to manage all api requests. You can see them as a bridge/an adapter between the server API and the view layer (scenes and components) of your application.
- It can take care of network calls your app will make, get and post content, and transform payloads as needed before being sent or saved in the store of your app (such as Redux).
- The screens and components will only dispatch actions, read the store and update themselves based on the new changes.
- Actions will use services. Thunk is a redux middleware used to handle asynchronous actions and side-effects.
- You can use "axios" for REST API calls in your service handlers or you can use any SDK like firebase sdk for direct DB interaction inside handler methods.

/styles :

- This module holds our application-level styles.
- It can include theme definition (font, colours, typography) of the app UI, and global styles.

> *<View styles={[globalStyles.wrapper, styles.textWrap]}>*
> …
> *</View>*

/utils :

- All the utility/helper methods, validations, etc that can be shared across our entire project are added here.

/__tests__ :

- Jest framework is default supported by react for unit testing the application.
- All the unit test files are placed inside "__tests__" dir alongside the corresponding .js files.
- It can be components, miscellaneous functions, containers, or like so.
- (More on Jest and react-native-testing-library implementation in "Section-3 Architecture Dependencies" below)

Section-2 Design Pattern

Photo by JJ Ying on Unsplash

Selecting a design pattern is an opinionated decision and should not impact the performance of the application. We are implementing the "Container — View pattern" in this application. One of the other famous design patterns is the "Atomic pattern".

- **Container and View pattern**: (ref: https://reactpatterns.com/ & https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0)
- Container-View pattern is the most efficient and widely used feature building pattern in react environment.
- Container Component is the entry point of the Feature/Screen. Responsibilities of a container component are:
- data fetching
- redux integration
- side-effects handling, heavy computation or data mapping
- Finally pass the required props down to the View.
- View Component: should contain only the presentation part.
- All the UI/presentation logic will reside here.
- Further complex elements can be broken down into individual components for ease of maintenance.
- Presentational components utilize props, render, and context.
- Presentational components receive data and callbacks from props only, which can be provided by its container or parent component.

Section-3 Architecture Dependencies

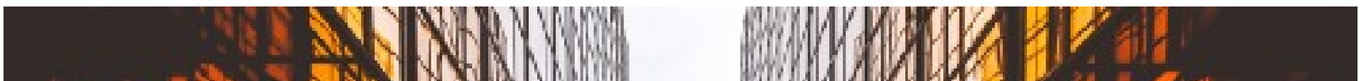


Photo by Alex wong on Unsplash

We'll start adding features/capabilities to our application one-by-one now. List of all the dependencies for our ReactNative project and its integration for a better architecture:

Step-1: React-navigation v5.5.1

(ref: https://reactnavigation.org/)

It will avail routing and navigation capability to our React native app. It provides us with various navigator implementations like Stack-navigation, Tabs-navigation or Drawer/Side-tray navigation.

React-navigation library also provides features like maintaining history of user journey along with gestures and animations that you may expect from a native mobile app.

- Use the following command to add a react-navigation library. We'll also require stack-navigation for this app.

> *$ npm install -S @react-navigation/native @react-navigation/stack*
> *$ npm install react-native-reanimated react-native-gesture-handler react-native-screens react-native-safe-area-context @react-native-community/masked-view*

Step-2: Unit testing framework

(ref: https://reactnative.dev/docs/testing-overview)
(ref: https://jestjs.io/docs/en/tutorial-react-native)

(ref: https://callstack.github.io/react-native-testing-library/docs/api/)

Jest is default supported in react-native applications for unit testing. This will enable us to perform unit testing and generating code coverage reports.

Jest gives below features out of the box:

> ✓ *Minimal configuration*
> ✓ *Watches only changed files*
> ✓ *Fast*
> ✓ *Snapshot testing (explained later)*
> ✓ *Coverage out of box*

In this app, we'll use 'react-native-testing-library' as a testing framework. React-native-testing-library will enable us to traverse through the element tree and simulate user actions. It provides light utility functions on top of react-test-renderer letting you always be up to date with latest React features and write any component tests.

Use below commands:

> *$ npm install — save-dev react-native-testing-library*
> *$ npm install — save-dev @testing-library/jest-native*

The 4 important test scripts we have

> *"scripts": {*
> *"test": "jest — coverage",*
> *"test:update": "jest — coverage — updateSnapshot",*
> *"test:watch": "jest — watch",*
> *"coverage": "jest — coverage && open ./coverage/lcov-report/index.html",*
> *}*

- test: It will go through all the test files and execute them. This command will also be used in pre-hooks and CI checks.
- test:watch: This will watch all the test files. It is very useful while writing tests and quickly seeing results.
- test:update: This command will update snapshots for all the presentational components. If the snapshot is not there, it will create it for you. We will discuss snapshots in detail in coming chapters.

Step-3: Internationalisation (i18n)

(ref: https://www.npmjs.com/package/react-native-i18n)

react-native-i18n is a library to enable multilingual support to our app. It comes handy with various features like getting the default language based on user local, fallback for missing translations, etc.

Use following command to add internationalisation in the app:

> *$ npm i — save react-native-i18n*

Step-4: Redux & Thunk

(ref: https://redux.js.org/)

Redux is used for the provision of a central store. All the data that needs to be shared across features, modules or react-component tree, can be added to the redux store.

Thunk is the middleware that works hand-in-hand with redux for handling the side-effects like fetching data from database, REST APIs, application processing delays, etc. It is promise based.

> *$ npm install — save redux react-redux*
> *$ npm install — save redux-thunk*

Step-5: UI Library — Material design library (react-native-paper)

(ref: https://callstack.github.io/react-native-paper/)

React Native Paper is a cross-platform UI component library which follows the material design guidelines, with global theming support and an optional babel-plugin to reduce bundle-size.

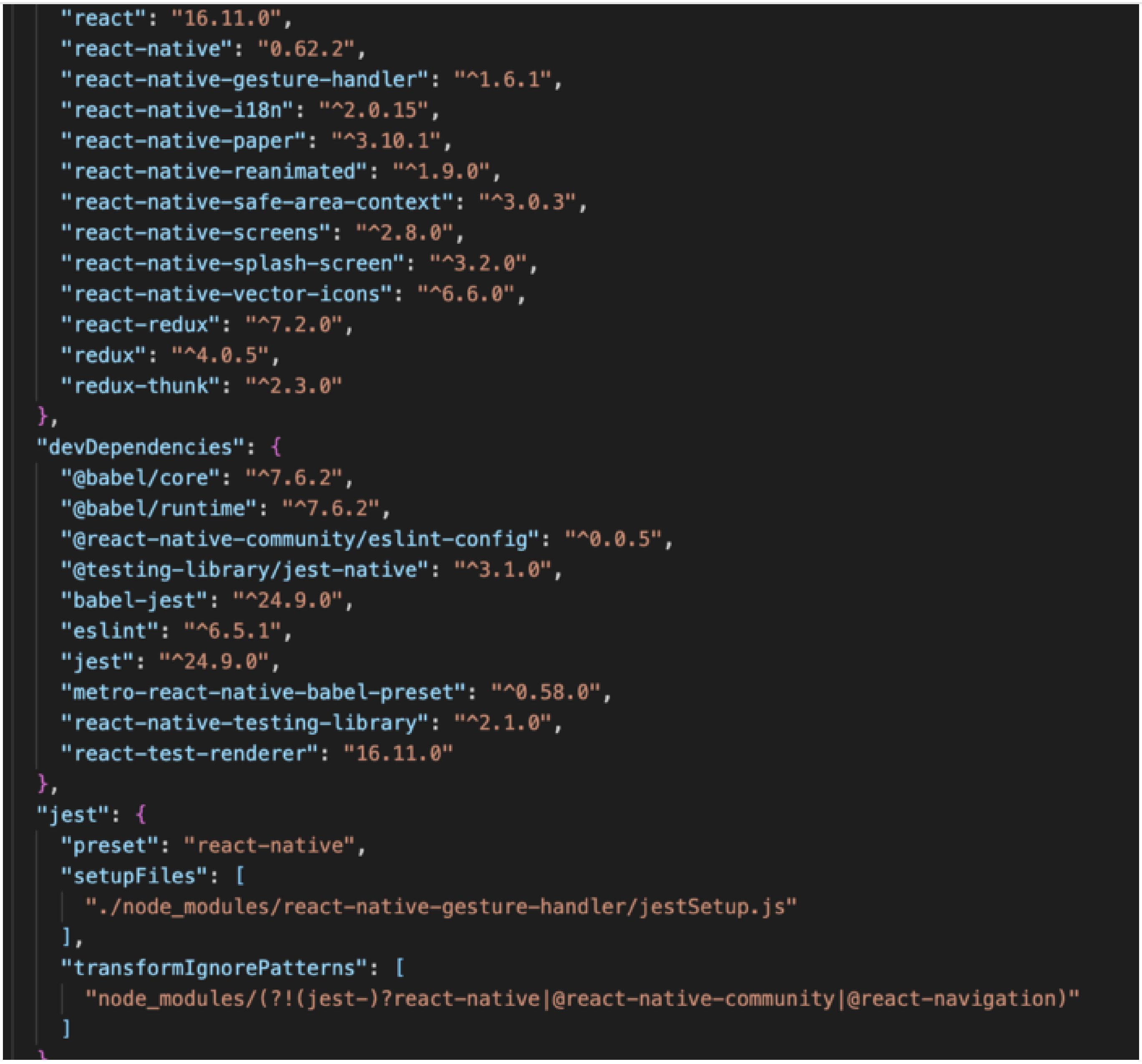
It uses vector-icons library, a set of customisable icons for React native with support for NavBar/TabBar/ToolbarAndroid, image source and full styling.

Use the below command to add it in the project.

> *$ npm install — save react-native-paper*
> *$ npm install — save react-native-vector-icons*

Quick look of the "package.json":

```
},
"dependencies": {
```



```
    "react": "16.11.0",
    "react-native": "0.62.2",
    "react-native-gesture-handler": "^1.6.1",
    "react-native-i18n": "^2.0.15",
    "react-native-paper": "^3.10.1",
    "react-native-reanimated": "^1.9.0",
    "react-native-safe-area-context": "^3.0.3",
    "react-native-screens": "^2.8.0",
    "react-native-splash-screen": "^3.2.0",
    "react-native-vector-icons": "^6.6.0",
    "react-redux": "^7.2.0",
    "redux": "^4.0.5",
    "redux-thunk": "^2.3.0"
  },
  "devDependencies": {
    "@babel/core": "^7.6.2",
    "@babel/runtime": "^7.6.2",
    "@react-native-community/eslint-config": "^0.0.5",
    "@testing-library/jest-native": "^3.1.0",
    "babel-jest": "^24.9.0",
    "eslint": "^6.5.1",
    "jest": "^24.9.0",
    "metro-react-native-babel-preset": "^0.58.0",
    "react-native-testing-library": "^2.1.0",
    "react-test-renderer": "16.11.0"
  },
  "jest": {
    "preset": "react-native",
    "setupFiles": [
      "./node_modules/react-native-gesture-handler/jestSetup.js"
    ],
    "transformIgnorePatterns": [
      "node_modules/(?!(jest-)?react-native|@react-native-community|@react-navigation)"
    ]
```

Voila! thats it. This should be able to make you app last for years without any scalability and maintenance issues.

. . .

My other react-native stories:

React JS — Architecture + Features + Folder structure + Design Pattern

😀Thank you😀
😀Clapping is motivating
😀Do not hesitate to **applause**

React Native Architecture Design Patterns Internationalization Redux