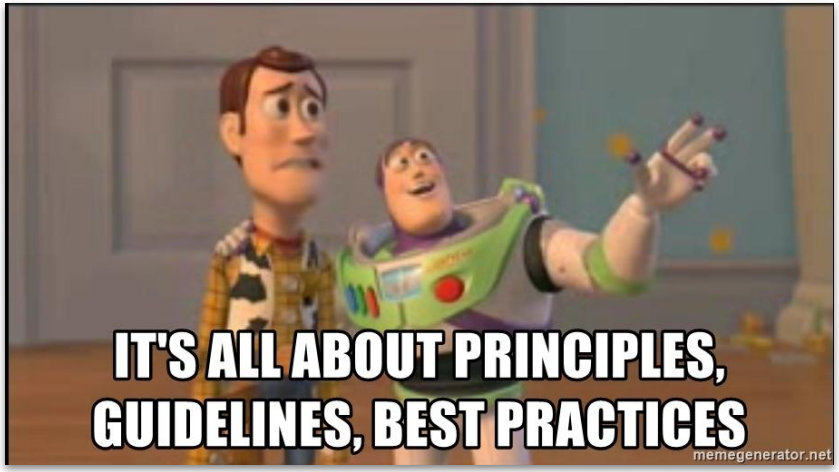
React Native Best Practices

A guide to write React Native Apps using Best Practices

1. Starting with basics

Some basic approach we should always follow.

- ➔ **No Inline Styling**
  Always keep styles separate
- ➔ **Minimum Platform Specific Code**
  Always register an application in the app entry point JS file (app.js) instead of registering twice in index.ios.js and index.android.js
- ➔ **Responsive UI**
  You already know what I mean :P
- ➔ **Use scalable and responsive Graphics**
  Always try to use SVGs or @1x, @2x, @3x slicing images instead of using a single image for assets.



Why Best Practices?

In order to improve your styling skills and eliminate errors in your application, it's crucial to follow best practices. This will help to keep your code better organized and you'll will not end up creating a laggy app. I know many of you already know some/all points lets just do a quick revision

2. React Hooks/ Functional Components for components

It is 2020 and you should be using Hooks! There are plenty of reasons for this: Hooks are declarative, easier to read and understand, and they reduce a lot of this.x, this.y and this.setState() in your code. We didn't have a need to use a Class-based component, which a Functional component using Hooks wouldn't solve.

```
const [showModal, setShowModal] = useState(true)
...
useEffect(() => {
  dispatch(doFetchOrders())
  dispatch(doFetchProducts())
}, [dispatch])
...
const authenticationData: AuthenticationStateType = useSelector
(authenticationSelector)
...
const mapViewRef: { current: MapView } = useRef(null)
```

Tip

Best way will be using class for main screens and react hooks for making components, this can be the better way to handle states and data.

—

3. Lock the dependencies! And say to your application, Live long!

Instead of

```
"react-native-linear-gradient": "^2.3.0"
```

Use:

```
"react-native-linear-gradient": "2.3.0"
```

(Remove ^ from your dependencies from package.json

To prevent breaks in future!)

4. Create Aliases

Create aliases using babel-plugin-module-resolver to avoid nested imports such as import Product from '../../../Components/Product'. Aliases created using babel-plugin-module-resolver look something like this.

```
import { checkDailyStreak } from "@actions/dayStreak";
import { viewedNotification } from "@actions/activity";
import { getSessionId } from "@storage/index";
import { ProfileI, UserAnalyticsI } from
"@interfaces/index";
import { getProfileInfo } from "@actions/index";
import { sendUserAnalytics } from "@actions/events";
```

```
plugins: [
  [
    "module-resolver",
    {
      "root": ".",
      "alias": {
        "@screens": "./src/screens",
        "@images": "./src/images",
        "@styles": "./src/styles",
        "@components": "./src/components",
        "@httpClient": "./src/httpClient",
        "@providers": "./src/providers",
        "@actions": "./src/actions",
        "@store": "./src/store",
        "@storage": "./src/storage",
        "@environment": "./src/environment",
        "@interfaces": "./src/interfaces",
        "@utils": "./src/utils",
      }
    }
  ],
  [
    "@babel/plugin-proposal-decorators",
    {
      legacy: true
    }
  ]
]
```

5. Crush the Crashes

Protect your app from non UI JS crashes and hard native crashes

Native crashes are a true pain for all developers but it doesn't have to be anymore. My suggestion here is, to make use of a package like *react-native-exception-handler* which allows to capture native errors and show an informing message to the user before the app closes unexpectedly.

6. Make errors beautiful

Errors can lead to crashes, If any error occurs when user is performing any wrong actions we should always show them in a Nice way!

```tsx
import * as React from "react";

import { NavigationInjectedProps } from "react-navigation";

import { Content } from "@components/Lesson";
import ContentItemProvider from "@providers/ContentItem/ContentItemProvider";
import ErrorBoundary from "@components/ErrorBoundary/ErrorBoundary";

export class ContentScreen extends React.Component<NavigationInjectedProps> {
    public render(): React.ReactNode {
        return (
            <ErrorBoundary>
                <ContentItemProvider>
                    <Content />
                </ContentItemProvider>
            </ErrorBoundary>
        );
    }
}
```

```tsx
class ErrorBoundary extends React.Component<NavigationInjectedProps, ErrorBoundaryStateI> {
    public readonly state: ErrorBoundaryStateI = {
        error: null,
        errorInfo: null
    };

    public componentDidCatch(error: Error, errorInfo: React.ErrorInfo): void {
        this.setState({error, errorInfo});
    }

    public render(): React.ReactNode {
        if (this.state.errorInfo) {
            return (
                <View style={{ flex: 1, alignSelf: "center", alignContent: "center", justifyContent: "center"
                    <Text
                        style={{
                            fontFamily: Font.family.semiBold,
                            fontSize: Font.size.m
                        }}
                    >
                        Sorry, but something went wrong
                    </Text>
                    <Button
                        buttonStyle={{
                            marginTop: 15
                        }}
                        color={ Colors.darkBlue }
```

Divide And Rule

Not a Single Import from 'React Native' (Classes)

JSX, React Hooks (useState, useEffect, useRef)

Components are children of Container

React Native Imports (View, Text, Image, etc.), Styles

Quick Tips!

Having many console.log statements can slow your app down, Be sure to disable all logs (manually or with a script) when making a release build.

If you want your app to look good on every iOS device you should use SafeAreaView which provides automatic padding when a view intersects with a safe area (notch, status bar, home indicator).

It is exceedingly difficult to set absolute styling for everything. Options help greatly with the layout. Third-party plugins like react-native-resposive will also help.

What Next?

Improving App Performance.

Pure Components, ComponentShouldUpdate, React.memo (To avoid re-rendering) and Redux

—

And,

Happy Independence day!

Thank You.