Prácticas de Código	
Lenguaje	JavaScript
Framework	React js, React Native
Aprobado por	Fabián Portillo

- Todo se maneja por GitHub
- Cada task o feature que se haga sobre el frontend, debe tener su propia rama en GitHub
- Cuando se termine cada feature se debe hacer un pr y debe ser aprobado por todos los que estén trabajando en los proyectos
- Se tendrán dos ramas principales, la de develop y la de producción
- Pull request a develop y una vez aprobado se pasa a producción
- (diagrama de flujo de aprobación con el respectivo encargado)
- Subir a git diario o cada vez que se termine funcionalidad (feature)
- Usa la última versión de React - toca revisar las dependencias
- Utilizar el scaffold de proyecto de Codev.
- Todas las variables en inglés (views, components y demás)
- Uso de redux para el manejo del estado.
- Usar en preferencia redux-saga para el manejo de los middlewares y las acciones en segundo plano como peticiones a la API y demás.
- Usar hooks para la creación de componentes evitar en lo posible el uso de clases.
- Organizar en una sola fuente de la verdad el estado de la aplicación y utilizar una sola carpeta según el scaffold de la aplicación.
- Evitar lo más que se pueda la repetición de código utilizando componentes reutilizables.
- Usar una fuente de la verdad para los colores, fuentes y temas de la aplicación

- Separar los servicios en una sola carpeta
- Organizar en una sola carpeta los archivos fuente de la aplicación
- Asignar a un solo archivo la administración de las rutas o pantallas de la aplicación.
- Todas las funcionalidades de cada componente deben estar en funciones para respetar el principio de responsabilidad única.
- Cada ruta o pantalla debe tener su propio archivo de estilos para implementar.
- Cada componente debe tener un archivo index.jsx o index.js que será el archivo principal por donde se expone la funcionalidad del componente.
- Utilizar acciones de redux para disparar peticiones a la API nunca llamar una solicitud fetch, axios, http u otra desde el propio componente.

1. **Scaffold de proyecto:** Cuando se crea una aplicación en React o React Native esta no provee un scaffold de proyecto específico como si lo hacen frameworks como Angular o Vue.

- Sin embargo, el siguiente scaffold se usa para utilizar una sola fuente de la verdad para cada componente de directorios.

- Componentes en una sola carpeta, vistas en una sola carpeta, estado en una sola carpeta, assets o archivos de la aplicación en una sola carpeta.

- Esto se hace para conservar el principio de una sola fuente de la verdad, de esta manera logramos acceder rápidamente a los directorios que componen nuestra aplicación sin la necesidad de estar adivinando donde esta cada cosa.

- Lo siguiente es la estructura de directorios que se debe utilizar en casa proyecto de React.

  src
    assets
    components
    router o navigator
    redux
    screens o views
    services
    theme
    utils
    App.js

- A continuación, se explica el propósito de cada uno de estos directorios.

- Para empezar, puede guardar el contenido de toda la aplicación en una src o app carpeta.

2. **Assets:** en el directorio assets puede guardar todos los archivos e imágenes que sean necesarios para la aplicación.

- Puede administrar la estructura de subdirectorios dentro de este directorio como guste, lo único importante es saber que aquí van única y exclusivamente los archivos e imágenes de uso de la aplicación.

- Una vez guarde los archivos en estos directorios puede hacer referencia a ellos desde los componentes de la aplicación.

3. **Components**: en el directorio de components deben ir todos los componentes de uso reutilizable en la aplicación.

- Cada componente debe tener un archivo index.jsx y styles.jsx en el caso de una aplicación React Native el archivo styles.jsx devolvería un objeto StyleSheet de react-native el cual se importaría en el index.jsx para utilizar dichos estilos en el componente.

- En el caso de una aplicación React js si está usando Material UI puede utilizar el archivo styles.jsx para devolver un withStyles, makeStyles o styled de material y luego importarlo en el index.jsx y aplicar dichos estilos al componente.

- O bien si prefiere utilizar CSS este archivo también puede ser un styles.css en lugar de un styles.jsx, luego importarlo a index.jsx e implementar los estilos sobre el componente.

- Todos los archivos index.jsx deben retornar el componente y preferiblemente dicho componente debe estar estructurado como un componente de función en lugar de una clase para poder hacer uso de los hooks.

- Los archivos index.jsx y styles.jsx deben estar dentro de una carpeta con el nombre del componente ejemplo:

```
src
  components
    header
      index.jsx
      styles.jsx
```

- Los componentes que se declaren dentro de este directorio no pueden contener lógica de negocio.

- Es decir, estos componentes deben ser adaptables y se deben de poder utilizar prácticamente en cualquier vista de la aplicación.

- Los componentes declarados en este directorio deben poderse utilizar más de una vez, es por eso que estos componentes no pueden tener lógica de negocio, ya que si creamos un componente con una lógica de negocio. Este solo nos servirá para ese propósito y no lo podremos reutilizar.

- Tome como ejemplo un componente Card, este componente se encargará de renderizar una tarjeta con una foto, un título, un texto y una serie de botones opcionales.

Lizard

Lizards are a widespread group of squamate reptiles, with over 6,000 species, ranging across all continents except Antarctica

SHARE LEARN MORE

- En lugar de crear este componente y escribir el contenido estáticamente, lo que debemos hacer es dejar que el contenido de la tarjeta dependa de las propiedades que se envíen en las props. De tal forma que se pueda llamar en distintas vistas y su imagen, titulo, texto y botones sean diferentes, pero compartiendo el estilo y diseño que preparo dicho componente.

- Ejemplo:

```
<Card
    image="https://anyweb.com/anyimage.jpg"
    title="My title"
    text="Lorem ipsum dolor ..."
    buttons={[
        {
            text: "Aceptar",
            onPress: () => {
                console.log("Aceptado");
            }
        },
        {
            text: "Cancelar",
            onPress: () => {
                console.log("Cancelado");
            }
        }
    ]}
/>
```

4. **Router o navigator:** En todo proyecto implementado en React debe haber un directorio específico para las rutas del proyecto.

- En el caso de un proyecto Web puede usar un archivo router/router.jsx que devuelva el enrutador de la aplicación ya armado con React Router, importarlo en el archivo principal src/App.js y llamar el enrutador dentro del componente principal de la aplicación.

- En el caso de una aplicación React Native el proceso sería igual solo que aquí se usaría un NavigationContainer de React Navigation, en navigator/navigator.jsx puede crear un componente que contenga el NavigationContainer junto con la lista de Stack.Navigator screens, importar este componente en src/App.js componente principal y llamar al Navigator componente.

- Lo importante es tener en una sola fuente de la verdad todas las rutas o pantallas de la aplicación, desde este punto pueden partir diferentes implementaciones de código que usted puede tomar siguiendo las recomendaciones anteriores.

5. **Redux:** En el directorio redux se deben guardar todos los archivos referentes a acciones, reducers, middlewares, requests y demás, todos estos archivos deben estar modularizados por carpetas de la siguiente manera:

```
src
   redux
      users
          constants.js
          actions.js
          redicer.js
          sagas.js
      contacts
          constants.js
          actions.js
          redicer.js
          sagas.js
      business
         constants.js
          actions.js
          redicer.js
          sagas.js
```

- Como puede ver en el ejemplo anterior la carpeta redux está separada por módulos y cada módulo contiene 4 archivos a continuación se explica la funcionalidad de cada uno.

- Para el siguiente ejemplo obtendremos datos de una API usando redux-saga, para esto necesitamos redux, react-redux y redux-saga así que ejecute los siguientes comandos desde el terminal parado en la raíz de su proyecto de React o React Native para instalar estos paquetes

  npm install redux react-redux redux-saga

- En src/redux/users/constants.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/users/constants.js

- En src/redux/users/actions.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/users/actions.js

- En src/redux/users/reducer.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/users/reducer.js

- En src/redux/users/sagas.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/users/sagas.js

- En src/redux/index-reducer.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/index-reducer.js

- En src/redux/index-sagas.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/redux/index-sagas.js

- En src/views/example/index.jsx agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/views/example/index.jsx

- En src/App.js agregue el siguiente código:

  https://github.com/Gabriel1777/redux-saga-example/blob/master/src/App.js

- Cada archivo está documentado explicando su funcionalidad también puede bajar este ejemplo de Github y probarlo usted mismo desde el siguiente repositorio:

  https://github.com/Gabriel1777/redux-saga-example

- Una vez descargado ejecute desde la terminal ubicado en la ruta del proyecto: npm install
- Luego simplemente ejecute npm start y esto le abrirá el ejemplo en el navegador http://localhost: 3000.

- El ejemplo anterior se ejecuta en React js pero en el caso de React Native es exactamente igual puede utilizar los mismos archivos, solo tiene que cambiar los por <View> los <p> por un <Text> y el <button> por un <Button> todos importados de react-native el ejemplo se varía así modificando src/views/example/index.jsx archivo:

  https://github.com/Gabriel1777/practices/blob/master/frontend/examplenative.jsx

- El ejemplo anterior muestra la forma en la que se deben de realizar las peticiones a la API utilizando redux-saga pero este ejemplo también le sirve para inicializar la estructura de un proyecto si así lo requiere.

6. **Screens O Views:** Cada aplicación está compuesta por una serie de vistas o de pantallas cada una renderizada desde un componente.

- Las vistas son simples componentes que representan una parte de la aplicación, la estructura de las vistas sigue la misma metodología que los componentes que están dentro del directorio components.

- Igualmente deben tener un directorio con un archivo index.jsx y styles.jsx.

```
src
   views
      home
         index.jsx
         styles.jsx
```

- En un proyecto de React Native este directorio puede llamarse screens y en un proyecto de React js puede llamarse views, aunque el nombre views también aplica para ambos casos.

- A diferencia de los componentes que están en el directorio components estos no deben ser reutilizables, al contrario estos componentes son los que se encargan de ejecutar la lógica de negocio de la aplicación y de reutilizar los componentes ubicados en el directorio components.

- Cada vista representa una parte o módulo de la aplicación y de la misma manera administra la lógica de negocio de dicho modulo.

- Un ejemplo de esto puede ser una vista de Home, Login o una vista de Perfil de Usuario.

7. **Services:** Existirán casos donde las aplicaciones consuman servicios de terceros para poder llevar a cabo su funcionalidad.

- Un ejemplo de esto puede estar en funcionalidades como Login con (Google, Facebook, Apple, etc …) algún servicio de Firebase como (Push Notifications, Dynamic Links, Authentication Phone), integraciones con Google Maps o funcionalidades nativas en el caso de las aplicaciones móviles como acceso a la cámara, imágenes, localización etc …

- En estos casos es importante organizar la lógica de la integración con el servicio en la carpeta src/services de esta manera se mantiene en un solo directorio la lógica de la integración de todos los servicios que está consumiendo la aplicación.

- Cada servicio puede ser un componente, función, clase o exportar varias funciones.

- También puede llamar ese componente, función, clase o funciones en cualquier parte de la aplicación, lo importante es mantener toda la lógica de servicios en un solo directorio.

8. **Theme:** Todas las aplicaciones deben manejar un tema global, este debe contener los colores y fuentes de la aplicación.

- El tema de la aplicación debe ser cargado en un solo archivo junto con los colores primarios, secundarios , fuentes y demás.

- Si está trabajando en un proyecto de React Native deberá crear 2 archivos un src/theme/colors.jsx y un src/theme/fonts.jsx

- En el src/theme/colors.js exportara un JSON con los colores hexadecimales de la aplicación

```
export const colors = {
  primary: '#003954',
  secondary: '#05B8BC',
  contrast: '#2D2D2D',
  border: '#E0E0E0',
  third: '#135879'
};
```

- En el src/theme/fonts.jsx exportara una constante con las diferentes variables de fuentes que usara el aplicativo.

```
import {Platform} from 'react-native';

export const fonts = {
  black: Platform.OS === 'ios' ? 'Poppins-Black' : 'Poppins-Black',
  bold: Platform.OS === 'ios' ? 'Poppins-Bold' : 'Poppins-Bold',
  book: Platform.OS === 'ios' ? 'Poppins-Regular' : 'Poppins-Regular',
  light: Platform.OS === 'ios' ? 'Poppins-Light' : 'Poppins-Light',
  medium: Platform.OS === 'ios' ? 'Poppins-Medium' : 'Poppins-Medium',
  normal: Platform.OS === 'ios' ? 'Poppins' : 'Poppins-Regular',
};
```

- Ahora en todos los estilos que aplique en todos los componentes de la aplicación deberá importar estos 2 archivos y utilizar la fuente y colores para los estilos, un ejemplo en src/views/example/styles.jsx aplicando el color primario y la fuente tipo bold:

```
import {StyleSheet} from 'react-native';
import {colors} from '../../theme/Colors';
import {fonts} from '../../theme/Fonts';

const styles = StyleSheet.create({
  title: {
    fontFamily: fonts.bold,
    color: colors.primary,
  },
});

export default styles;
```

- Si está trabajando en un proyecto Web de React js puede utilizar la misma metodología, aunque si está utilizando Material UI puede encerrar su aplicación en un ThemeProvider para definir temas y colores para todo el árbol de la aplicación.

- Lo importante es asignar a un solo archivo a la definición de colores y fuentes de la aplicación, de esta manera se pueden cambiar una sola vez en el archivo que se declararon los valores y ya se reflejarían los cambios en toda la aplicación.

- De esta manera se maneja una solo fuente de la verdad para cada caso.

9. **Utils:** en el directorio utils se declaran todas las constantes de la aplicación además de los métodos personalizados.

- Por ejemplo en src/utils/constants.js normalmente se declara la constante del dominio del Backend de la aplicación por ejemplo:

```
export const ROUTE_ENDPOINT = 'https://api.anyapplication.com.co/api';
```

- De aquí se exporta para que los sagas encargados de realizar las peticiones a la API puedan obtener el dominio a donde deben disparar.

- Y de esta manera si se desea cambiar el dominio se haría solamente en un parte de la aplicación.

- También puedes guardar lógica personaliza y exportarla para los demás componentes, un ejemplo podría ser una función que recibe un número y lo transforma en un string con formato $.

- De esa manera estaría declarando su función en una sola parte de la aplicación y utilizándola en varios componentes.

RESPONSABLE DE APROBACIONES.

Por CODEV	**Ricardo Ramos Cuervo** **312 449 3543** **comercial@codev.com.co**