¿Cómo organizamos un proyecto de tamaño medio en React?

En este pequeño documento se explora la forma como la unidad de IDI (Investigación, Desarrollo e Innovación) de IAS Software organiza algunos de sus proyectos de innovación escritos en React. No hay una forma correcta de organizar proyectos en React, ya que es una librería de JavaScript para construir interfaces gráficas, y no un framework que define reglas estrictas sobre la estructuración y la arquitectura. A pesar de esto, hay una serie de buenas prácticas que son útiles al crear un nuevo proyecto.

Las reglas del juego

En un proyecto en React se pueden encontrar diferentes tipos de archivos que conforman el proyecto:

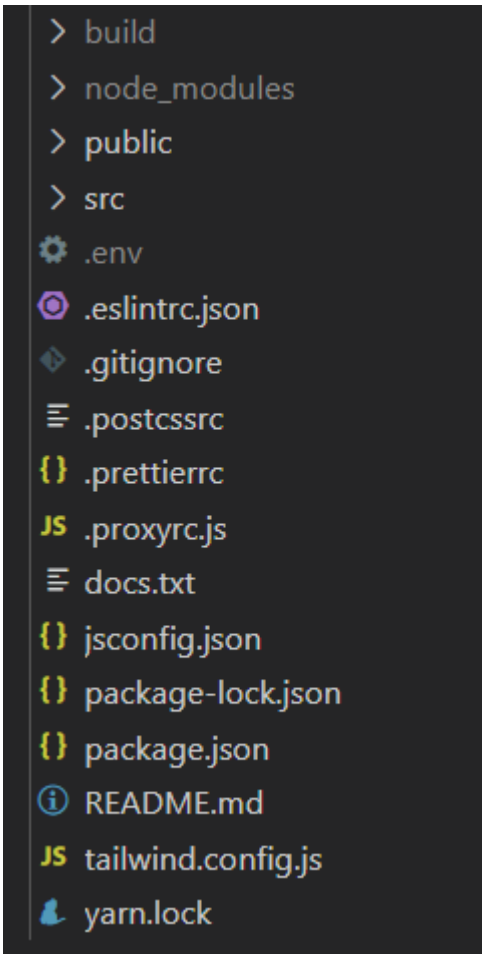
- Contenido audiovisual como JPG, PNG, GIF, SVG, MP3, MP4
- Fuentes de texto
- Animaciones como Lottiefiles en formato JSON u otros formatos
- Funciones puras de JS (JavaScript) o TS (TypeScrip) que se utilizan de forma global
- Componentes de React JSX (JavaScript XML) que se reutilizan en toda la aplicación
- Esquemas de bases de datos como archivos JS que definen las peticiones al Backend
- Test unitarios de los componentes de la app
- Contextos de la app
- Vistas de la app como componentes únicos
- Hooks para manejar los estados de las vistas
- CSS globales y de cada componente que permiten estilizar las vistas
- Archivo de entrada HTML
- Archivo App.jsx que es el componente raíz de la aplicación en React
- Archivo index.jsx que es el punto de entrada para la aplicación en node
- Variables de entorno de la app
- Todos los archivos de configuración como el package, ESLint, gitignore, prettierrc, jsconfig

Estos archivos pueden ser organizados en una estructura que permita encontrar, modificar y añadir fácilmente más funcionalidades y vistas al proyecto.

En la unidad de IDI muchos programadores, de diferentes niveles, contribuyen a la creación de proyectos de innovación, pero el paso de muchos de ellos es temporal;

esto ha conllevado a buscar una estructura que sea fácil de usar y mantener para un programador de cualquier nivel.

La raíz de un proyecto de la unidad de IDI puede estar encarpetado de la siguiente forma:

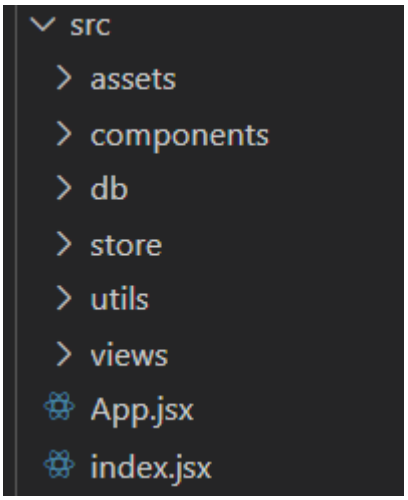


El folder /build, muchas veces conocido como /dist o /output, contiene el proyecto empaquetado por el bundler seleccionado (ej. WebPack, Parcel, Vite, entre otros). Dentro de esta carpeta están los archivos que permiten distribuir el proyecto en el servidor de producción.

La carpeta de /node_modules contiene todas las librerías y dependencias que se utilizan en el proyecto. En la raíz se encuentran los archivos de configuración y las variables de entorno.

La carpeta /public contiene el index.html que es el archivo de entrada HTML sobre el que React construye toda la aplicación. En esta carpeta también se encuentra el favicon.ico y el manifest.json.

El folder más importante para la organización del proyecto es /src que contiene toda la aplicación en React (componentes, vistas, recursos, funciones útiles y más). Vale la pena dedicar toda una sesión a analizar la organización de esta carpeta, que puede estar estructurada de la siguiente forma:



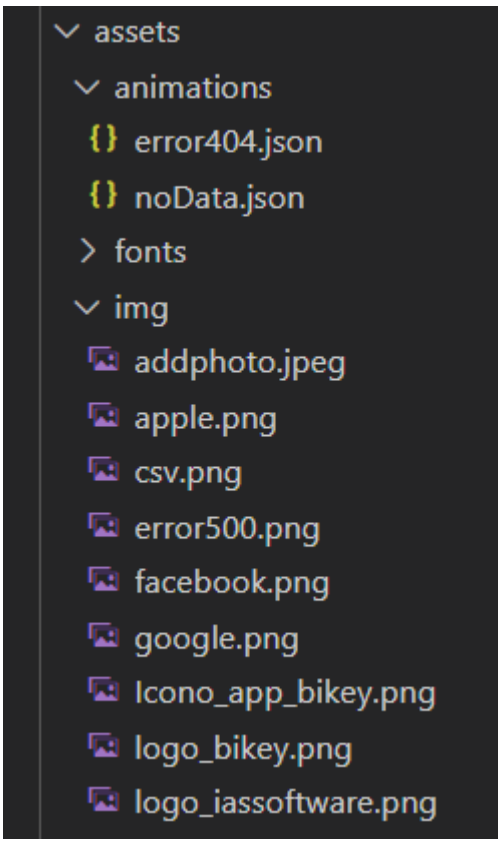
Hablemos de /src

En la raíz de esta carpeta está el punto de entrada. En React, cada vista y componente tiene una jerarquía, en la que App.jsx es el componente más alto. En muchas de las aplicaciones de IDI App.jsx contiene al cliente de Apollo y el componente principal de navegación.

Por otro lado, index.jsx es el primer punto de entrada, definido por node, que renderiza App.jsx.

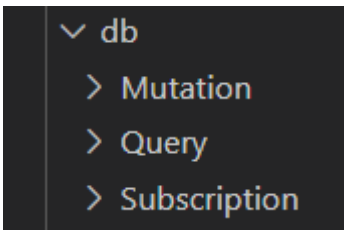
/assests

Aquí se ubican todos los elementos audiovisuales, fuentes de texto y animaciones utilizados alrededor de toda la app.



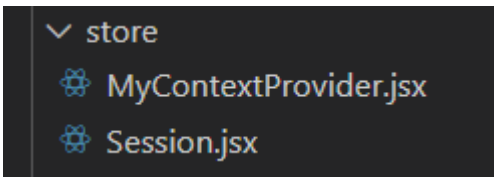
/db

Aquí se ubican los archivos JS con las mutaciones, queries y suscripciones que definen el esquema de GraphQL para peticiones HTTPS al back. Si se utiliza REST, en estas carpetas se ubican los esquemas de consulta POST, GET, PUT y DELETE con sus respectivos paths.



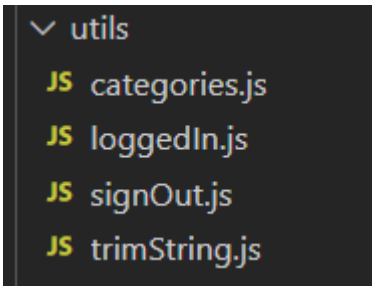
/store

Aquí se ubican los contextos de la app que proporcionan una forma de pasar datos a través del árbol de componentes, sin tener que pasar props manualmente en cada nivel. Esto permite acceder a la información de contexto de forma global alrededor de los componentes de la app.



/utils

En esta carpeta se ubican las funciones puras que se utilizan alrededor de toda la app. Como ejemplo, aquí pueden estar las funciones de JS para hacer operaciones sobre una cadena de texto, operaciones aritméticas, trasformación de horas y mucho más. Estas funciones no utilizan estados, hooks, ni componentes externos, pero hacen operaciones puntuales sobre datos de entrada.



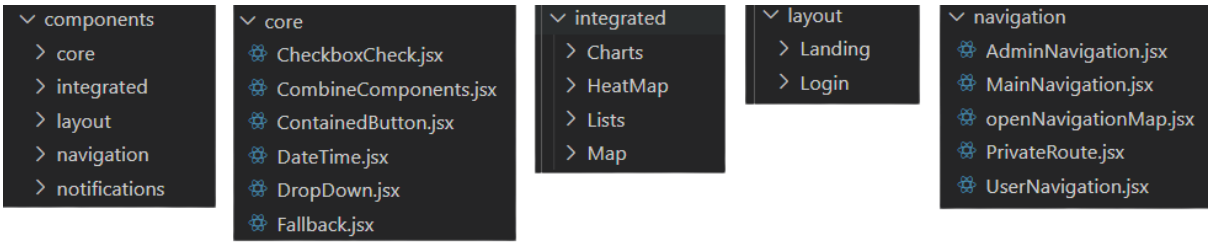
/components

Aquí se ubican los archivos que se van a reutilizar en toda la aplicación. Se utilice o no una librería de diseño, como lo es Material-UI, se deben crear archivos para cada componente personalizado. Estos componentes a menudo van acompañados de archivos CSS para estilizarlos; si se utiliza un framework de CSS, como Tailwind, los estilos pueden ir dentro del mismo markup en los archivos JSX.

En los proyectos de IDI es común encontrar los componentes separados por una jerarquía de carpetas que definen su uso de la siguiente forma:

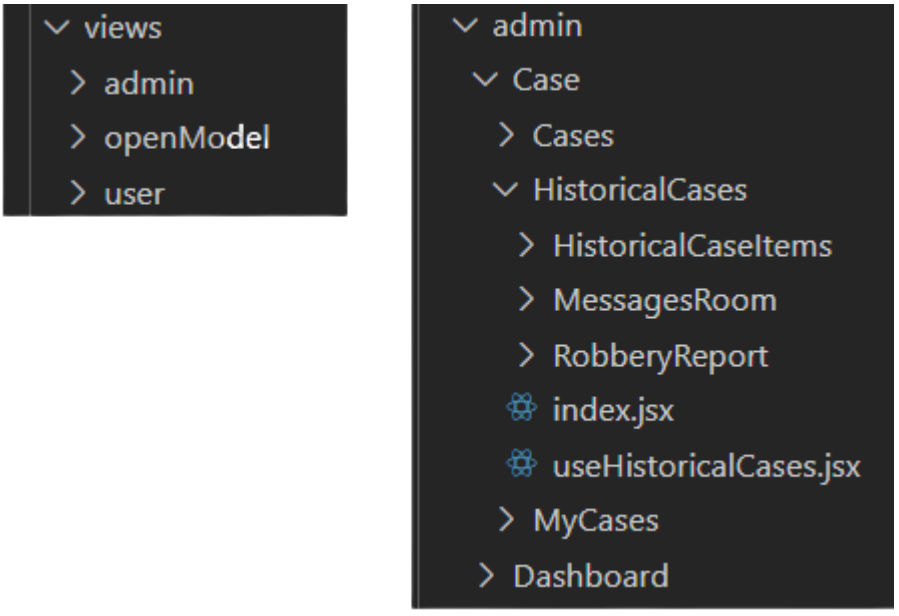
- Una carpeta /core con los componentes individuales centrales de la app como: botones, áreas de texto, menús desplegables, spinner, entre otros.
- Una carpeta /integrated con componentes compuestos, que por lo general son más complejos y se dividen en categorías. Aquí podemos encontrar Gráficas, Mapas, Listas, entre otros.
- El folder de /layout contiene el cuerpo reutilizable de la app. Este esqueleto, a menudo responsive, puede contener la barra de navegación y la cabecera. Este cuerpo es, de alguna forma, el lienzo donde viven las vistas de nuestra aplicación.
- La carpeta de /navigation define las rutas del proyecto utilizando una librería como REACT ROUTER.

- Se separan los componentes de notificación, como páginas de error, alertas, y modales, en una carpeta llamada /notifications que también podría ubicarse en los componentes integrados.



/views

Dentro de esta carpeta está el cuerpo de una aplicación, que utiliza todos los recursos anteriormente descritos para definir las diferentes páginas con las que el usuario final interactúa. Cada vista tiene: Un index.jsx de entrada, componentes propios no reutilizables, y hooks personalizados, que definen los estados y la lógica para manejar la integración de los diferentes componentes que conforman la vista. Para los proyectos que tienen roles o suscripciones, normalmente se definen carpetas que contienen las vistas segregadas por rol, un ejemplo es tener una carpeta para el perfil usuario y otro para el administrador. Los roles o las suscripciones pueden vivir en encarpetados distintos, esto depende de la magnitud de los cambios visuales por rol o suscripción.



Conclusiones

No hay una forma única de estructurar proyectos en React, vale la pena buscar una estructura que ayude a mantener bien un proyecto. Esta es una propuesta, y la forma en la que se trabaja en IDI para organizar los proyectos de tamaño medio en React, es escalable y sostenible en el tiempo.

En este documento se presenta una arquitectura fácil de entender para cualquier desarrollador, sea que tenga experiencia o que apenas esté incursionando en el mundo de la programación. Utilizar o no esta forma de estructurar un proyecto depende completamente del equipo de desarrollo, de los objetivos, y del tamaño del proyecto. Para un proyecto grande con muchas estructuras, roles y suscripciones que cambian la forma visual de proyecto, tal vez lo mejor es otra arquitectura; quizás una que combine la forma aquí descrita y la replique en subestructuras utilizando diseño atómico

En resumen un poco más simple y claro….

Estructura Recomendada de Carpetas en React

Una estructura común y eficiente para un proyecto React puede incluir las siguientes carpetas y archivos:

1. src/:
    - Contiene todo el código fuente de la aplicación.
    - Es la carpeta principal dentro del proyecto.
2. src/components/:
    - Incluye todos los componentes reutilizables y modularizados de la aplicación.

Ejemplo:

```
src/
├── components/
│   ├── Header.js
│   ├── Footer.js
│   └── Button.js
```

    -

3. src/pages/:
    - Contiene los componentes que representan páginas completas dentro de la aplicación.

Ejemplo:

```
src/
├── pages/
│   ├── Home.js
```

```
│   ├── About.js
│   └── Contact.js
```

4. src/assets/:
    - Contiene imágenes, íconos, estilos (CSS, SCSS) y otros recursos.

Ejemplo:

```
src/
├── assets/
│   ├── images/
│   ├── styles/
│   └── icons/
```

5. src/utils/:
    - Archivos con funciones auxiliares o lógica compartida que se usan en varios lugares del proyecto.

Ejemplo:

```
src/
├── utils/
│   ├── api.js
│   ├── constants.js
│   └── helpers.js
```

6. src/context/:
    - Archivos relacionados con el contexto (React Context API) para manejar estados globales.

Ejemplo:

```
src/
├── context/
│   ├── AuthContext.js
│   └── ThemeContext.js
```

7. Otros Archivos Importantes en src:
    - App.js: Punto de entrada principal de los componentes.
    - index.js: Archivo raíz que conecta React con el DOM.