Universidad Nacional de Colombia - sede Bogotá
Facultad de Ingeniería
Departamento de Sistemas e Industrial
Curso: Ingeniería de Software 1 (2016701)
David Sebastián Hurtado Sánchez

React

React es una biblioteca de JavaScript de código abierto para la construcción de interfaces de usuario, desarrollada y mantenida por Meta (antes Facebook) junto con una comunidad de desarrolladores independientes. Fue lanzada en 2013 y rápidamente ganó popularidad gracias a su enfoque innovador para desarrollar aplicaciones web y móviles de manera eficiente y modular.

Características Principales de React

1. Declarativa:
    - React utiliza un enfoque declarativo para construir interfaces de usuario. Esto significa que describes el "qué" en lugar del "cómo". Por ejemplo, en lugar de manipular manualmente el DOM, React gestiona los cambios necesarios de manera automática.
    - Este enfoque facilita el desarrollo, la depuración y el mantenimiento del código.
2. Basado en Componentes:
    - En React, la interfaz de usuario se divide en componentes reutilizables que encapsulan tanto la lógica como la presentación.
    - Los componentes pueden ser tan pequeños como un botón o tan grandes como una página entera, permitiendo construir aplicaciones modulares.
3. Virtual DOM:
    - React introduce el concepto de Virtual DOM, una representación en memoria del DOM real.
    - Cuando hay un cambio en la interfaz, React actualiza solo los elementos afectados en el DOM real, mejorando el rendimiento.
4. Unidireccionalidad de los Datos (Data Flow):
    - React implementa un flujo de datos unidireccional, lo que significa que los datos fluyen desde el componente padre hacia los componentes hijos.
    - Esto proporciona mayor control sobre cómo los datos cambian y se representan en la interfaz.
5. JSX (JavaScript XML):

- React utiliza JSX, una extensión de JavaScript que permite escribir estructuras HTML directamente dentro del código JavaScript.
- Aunque es opcional, JSX facilita la lectura y escritura del código al combinar lógica y estructura en un solo lugar como en el siguiente ejemplo

```
const App = () => {
  return <h1>Hello, React!</h1>;
};
```

6. Hooks:
    - Los Hooks son una funcionalidad introducida en React a partir de la versión 16.8. Su principal propósito es permitir el uso del estado y otras características de React en componentes funcionales, eliminando la necesidad de usar componentes de clase para manejar estas capacidades. Los Hooks hacen que el código sea más limpio, reutilizable y fácil de entender.

7. Ecosistema Rico:
    - React no es una solución completa. Ofrece herramientas esenciales para construir interfaces, pero su ecosistema incluye bibliotecas populares como:
        - React Router para enrutamiento.
        - Redux o Context API para manejar estados globales.
        - Next.js para renderizado del lado del servidor.

¿Qué son las dependencias y por qué son importantes?

En proyectos React (y en general con JavaScript), las dependencias son paquetes externos (librerías, herramientas, plugins) que se utilizan para extender las funcionalidades del proyecto. Estas dependencias se instalan y administran utilizando gestores de paquetes como npm o yarn.

Ejemplo:

- React y React DOM son dependencias que necesitas para desarrollar interfaces de usuario.
- Axios es una dependencia para realizar llamadas HTTP.

¿Cómo se gestionan las dependencias en React?

La gestión de dependencias incluye instalar, actualizar, eliminar y auditar librerías o herramientas externas. Esto se hace principalmente mediante npm (Node Package Manager) o yarn.

1. **Inicializar un proyecto**
   Cuando inicias un proyecto de React, el sistema crea un archivo package.json. Este archivo lista todas las dependencias utilizadas en el proyecto, junto con sus versiones.
   Si usas Create React app:

```
npx create-react-app my-app
cd my-app
```

   Esto genera un archivo package.json que incluye dependencias básicas como:

```
{
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0"
  }
}
```

2. **Instalar dependencias**
   Puedes instalar una dependencia usando npm o yarn:
   - Npm:

```
npm install nombre-del-paquete
```

   - Yarn:

```
yarn add nombre-del-paquete
```

   Ejemplo:

```
npm install axios
```

   Esto agrega Axios a tu proyecto y actualiza package.json:

```
"dependencies": {
  "axios": "^1.2.0"
}
```

3. **DevDependencies (Dependencias de desarrollo)**

   - Son librerías necesarias solo para el entorno de desarrollo (no en producción).
   - Se instalan con la bandera --save-dev (npm) o --dev (yarn).

   Ejemplo:

```
npm install eslint --save-dev
```

Esto agrega ESLint como una herramienta para revisar el código en el desarrollo, pero no se incluirá en la aplicación final.

4. **Instalar todas las dependencias**

   Si descargas un proyecto existente, las dependencias están listadas en package.json, pero no incluidas en tu carpeta local. Puedes instalarlas todas con:

   ```
   npm install
   ```

   O con:

   ```
   yarn install
   ```

5. **Actualizar dependencias**
   A lo largo del tiempo, las dependencias reciben actualizaciones. Puedes ver si hay versiones más recientes ejecutando:

   ```
   npm outdated
   ```

   Y puedes actualizar a la última versión compatible usando:

   ```
   npm update
   ```

6. **Eliminar dependencias**
   Si decides que ya no necesitas una dependencia, puedes eliminarla con:

   ```
   npm uninstall nombre-del-paquete
   ```

La URL instalación oficial es https://es.react.dev/learn/installation

Estructura Recomendada del Proyecto React

La estructura de carpetas de un proyecto en React debe ser modular y escalable, A continuación un ejemplo de como debería ser la estructura.

```
my-app/
├── public/
├── src/
│   ├── components/
│   ├── pages/
│   ├── hooks/
│   ├── context/
│   ├── services/
│   ├── assets/
│   ├── styles/
│   ├── utils/
│   ├── tests/
│   ├── App.js
│   ├── index.js
├── package.json
├── README.md
```

1. **public/**:
   - Contiene los archivos estáticos públicos.
   - **Propósito**: Estos archivos no pasan por el proceso de compilación y son accesibles directamente desde el navegador.
   - Ejemplo de contenido:
     - index.html: Archivo HTML principal donde se inyectará la aplicación React.
     - Archivos como favicon.ico, imágenes estáticas o configuraciones (e.g., manifest.json).
2. **src/**:
   - Carpeta principal donde resides todo el código fuente de la aplicación.
   - Contiene subcarpetas organizadas por funcionalidad.
3. **src/components/**:
   - Contiene los **componentes reutilizables** que son bloques básicos de la interfaz.
   - **Propósito**: Facilitar la reutilización de elementos comunes (e.g., botones, menús).
   - Ejemplo:

```
components/
├── Button/
│   ├── Button.js
│   ├── Button.css
│   ├── Button.test.js
├── Header/
│   ├── Header.js
│   ├── Header.css
```

4. **src/pages/**:
    - Contiene los **componentes principales de las páginas** de la aplicación.
    - **Propósito**: Cada archivo representa una página única (e.g., Home, About, Contact).
    - Ejemplo:

      ```
      pages/
      ├── Home.js
      ├── About.js
      ├── Contact.js
      ```

5. **src/hooks/**:
    - Carpeta para **custom hooks**.
    - **Propósito**: Encapsular lógica reutilizable basada en hooks.
    - Ejemplo:

      ```
      hooks/
      ├── useFetch.js
      ├── useForm.js
      ```

6. **src/context/**:
    - Carpeta para la implementación de la **Context API**.
    - **Propósito**: Manejar el estado global de la aplicación.
    - Ejemplo:

      ```
      context/
      ├── ThemeContext.js
      ├── AuthContext.js
      ```

7. **src/services/**:
    - Carpeta para las **llamadas a APIs o servicios externos**.
    - **Propósito**: Centralizar todas las peticiones HTTP.
    - Ejemplo:

      ```
      services/
      ├── api.js
      ├── userService.js
      ```

8. **src/assets/**:
    - Carpeta para **recursos estáticos** (imágenes, íconos, fuentes).
    - **Propósito**: Organizar los recursos multimedia del proyecto.
    - Ejemplo:

      ```
      assets/
      ├── images/
      ├── icons/
      ├── fonts/
      ```

9. **src/styles/**:
    - Carpeta para **estilos globales** de la aplicación.
    - **Propósito**: Centralizar y organizar los estilos compartidos.
    - Ejemplo:

```
styles/
├── globals.css
├── variables.css
```

10. **src/utils/**:
    - Carpeta para **funciones utilitarias**.
    - **Propósito**: Contiene funciones genéricas que pueden ser reutilizadas en varios lugares.
    - Ejemplo:

      ```
      utils/
      ├── formatDate.js
      ├── calculateTotal.js
      ```

11. **src/tests/**:
    - Carpeta para las **pruebas unitarias**.
    - **Propósito**: Mantener el código testeado para garantizar calidad y funcionalidad.
    - Ejemplo:

      ```
      tests/
      ├── App.test.js
      ├── utils.test.js
      ```

12. **App.js**:
    - Archivo principal del componente raíz de la aplicación.
    - Importa las rutas y estructura principal.
13. **index.js**:
    - Punto de entrada principal.
    - Renderiza la aplicación en el DOM.
14. **package.json**:
    - Archivo que contiene las dependencias, scripts y configuración del proyecto.
15. **README.md**:
    - Archivo para documentar tu proyecto (propósito, cómo ejecutar, etc.).

Convenciones para Nombres de Carpetas y Archivos

Estándares Generales

Consistencia: Mantén un estilo uniforme en todo el proyecto.

Claros y descriptivos: Los nombres deben describir el contenido o propósito de manera clara. Algunos estilos recomendados para los proyectos son

1. PascalCase: En este estilo, cada palabra comienza con una letra mayúscula, sin espacios ni guiones. Se utiliza principalmente para nombres de componentes y clases.
   Ejemplo:
   Correcto:
   - ButtonComponent
   - UserProfile

   Incorrecto:

- buttoncomponent
- user_profile

2. camelCase: La primera palabra comienza con una letra minúscula, y las siguientes palabras empiezan con una mayúscula. Se utiliza para nombrar funciones, variables, y hooks personalizados.
   Ejemplo:
   Correcto:
   - fetchData
   - useAuth

   Incorrecto:
   - FetchData
   - use_auth

3. kebab-case: Todas las palabras están en minúsculas y se separan con un guion (-).Se utiliza principalmente para archivos estáticos (como imágenes) o nombres de carpetas genéricas.

   Ejemplo:

   Correcto:

   - user-profile-picture.jpg
   - global-styles.css

   Incorrecto:

   - userProfilePicture.jpg
   - GlobalStyles.css

Guías de Estilo para JavaScript y React

1. ESLint
   - URL oficial: https://eslint.org/docs/latest/user-guide/configuring/
   - Propósito:
     - ESLint es una herramienta para analizar y encontrar problemas en el código JavaScript.
     - Permite definir reglas específicas de estilo para mantener un código consistente y limpio.
   - Configuración típica en React:

2. Prettier
   - URL oficial: https://prettier.io/docs/en/
   - Propósito:
     - Prettier es un formateador de código que garantiza un estilo uniforme en todo el proyecto.
   - Instalación:

```
npm install --save-dev prettier
```

3. Airbnb JavaScript Style Guide
   - URL oficial: https://github.com/airbnb/javascript

- Propósito:
  - Proporciona un conjunto de reglas bien definidas y aceptadas ampliamente para escribir JavaScript de alta calidad.
- Configuración en React:
  - Instalar las dependencias necesarias:

```
npx install-peerdeps --dev eslint-config-airbnb
```

Guías de Estilo para CSS

Stylelint

- URL oficial: https://stylelint.io/
- Propósito:
  - Herramienta para analizar estilos CSS y detectar errores o inconsistencias.
- Configuración típica:
  - Archivo .stylelintrc.json:
  - {
  - "extends": "stylelint-config-recommended",
  - "rules": {
  - "indentation": 2,
  - "string-quotes": "double"
  - }
  - }

Guías de Estilo para Node.js

- Node.js Best Practices:
  - URL oficial: https://github.com/goldbergyoni/nodebestpractices
  - Propósito: Conjunto de prácticas recomendadas para proyectos en Node.js.

Otras Herramientas y Guías

1. TypeScript
   - Guía oficial de estilo para TypeScript:
     - URL: https://typescript-eslint.io/
     - Propósito: Configurar ESLint para trabajar con TypeScript y mantener consistencia en proyectos.
2. React Docs: Guía de Mejores Prácticas
   - URL oficial: https://react.dev/learn
   - Propósito:
     - Proporciona pautas oficiales para estructurar y escribir aplicaciones React.

Material Adicional

En mi experiencia se me complico poder usar react por este motivo dejo este video adicional de la instalacion en windows https://www.youtube.com/watch?v=2SwKpVVNfIE.

Y me gustaria compartir este tutorial de react desde cero donde podemos fortalecer conceptos basicos para su uso

https://www.youtube.com/watch?v=7iobxzd_2wY&list=PLUofhDIg_38q4D0xNWp7FEHOTcZhjWJ29