# 1. ReactJS y su Aplicación en el Proyecto del Hospital

**- Explica qué es ReactJS y justifica su uso en el proyecto del hospital.**

**ReactJS** es una biblioteca de JavaScript de código abierto desarrollada por Facebook en 2013, diseñada para construir interfaces de usuario interactivas y reutilizables. Su enfoque se basa en componentes, lo que permite a los desarrolladores crear elementos de la interfaz de manera modular y eficiente. Además, React utiliza un Virtual DOM, lo que optimiza la actualización de la interfaz al minimizar las manipulaciones directas del DOM real, mejorando así el rendimiento de las aplicaciones web.

**Justificación de su uso en el proyecto  de centro médico:**

**ReactJS** facilita la gestión de cambios en la interfaz por su **enfoque basado en componentes**, permite dividir la interfaz en componentes reutilizables, esto facilita el desarrollo y mantenimiento del proyecto, ya que cada componente es creado y probado de manera independiente.

La estructura basada en componentes de ReactJS facilita el mantenimiento y la escalabilidad del sistema. A medida que el hospital crece y se añaden nuevos servicios o funcionalidades, es más sencillo actualizar y expandir la aplicación.

# 2. Analiza por qué utilizar una SPA desarrollada con ReactJS sería beneficioso para el sistema del hospital.

La capacidad de React para construir aplicaciones de una sola página (SPA) es una ventaja ya que en lugar de recargar la página completa cada vez que un usuario navega entre secciones, React puede actualizar dinámicamente el contenido de la página sin perder el estado de la aplicación. Esto proporciona una experiencia de usuario más fluida y rápida en un sitio web que puede tener múltiples usuarios accediendo a diferentes secciones al mismo tiempo. Desarrollar una SPA con ReactJS no solo mejora la navegación y la carga de datos, sino que también optimiza la experiencia del usuario por ofrecer una interfaz más fluida y rápida.

# 3. Manejo del DOM Virtual en la Web del Hospital

Gracias al Virtual DOM, React logra mejorar significativamente el rendimiento y simplifica la construcción de interfaces.
**Virtual DOM** es mucho más rápido que operar directamente en el DOM real.
**Detección de cambios** eficiente, gracias al algoritmo de Diffing, React solo actualiza los nodos donde realmente hay un cambio.
**Interfaz de usuario más fluida**: React puede mantener una navegación sin retardos visibles.

# 4. Comparación de ReactJS con Otros Frameworks para el Proyecto

En comparación, **Angular** podría ser excesivo por ser más rígido y tener una curva de aprendizaje mayor, mientras que **VueJS,** aunque es una alternativa viable, no cuenta con el mismo nivel de escalabilidad y soporte comunitario que React.

**ReactJS sería la mejor opción para la creación de la página web del centro médico por:**

**Flexibilidad:** permite integrar fácilmente servicios externos como APIs REST para el manejo de datos (ej., citas médicas, horarios, usuarios) y cualquier base de datos backend.

**Manejo dinámico:** su ciclo de vida de componentes y el Virtual DOM optimizan la velocidad y el rendimiento de actualizaciones frecuentes, ideal para una app interactiva.

**Modularización:** la construcción de componentes permite mantener el proyecto organizado, escalable y fácil de mantener.

**Componentización**: Facilita la división de la interfaz hospitalaria en piezas reutilizables, organizadas por funciones específicas logrando un diseño mantenible y escalable.

**Flujo de datos unidireccional:** Garantiza un sistema consistente y predecible en el manejo de información médica y funcionalidad de citas al sincronizar el frontend con el backend de manera eficiente.

**JSX:** Simplifica la creación de interfaces dinámicas y personalizables y asegura interactividad entre componentes. Es una extensión de sintaxis que combina HTML y JavaScript dentro de un solo archivo.

Ejemplo de cómo aplicarlo: Renderizar un componente de lista que muestra los doctores disponibles según el filtro del paciente.

# 5. Configuración y Ejecución de ReactJS

### 1 – Crear el proyecto:

```
npx create-react-app medical-center
cd medical-center
```

*“**npx**” descarga y ejecuta el paquete “**cd**” change directory" cambia a la carpeta medical-center*

### 2 - Iniciar el servidor de desarrollo:

```
npm start
```

*Esto ejecuta la aplicación en http://localhost:3000 e inicia un servidor web con recarga automática mediante Webpack.*

### Estructura de directorios inicial creada:

```
medical-center /
├── node_modules/ (paquetes instalados)
├── public/       (archivos estáticos, ej: index.html)
├── src/          (código principal del proyecto)
|   ├── App.js    (componente raíz de React)
|   ├── index.js  (archivo de entrada para montar React en el DOM)
├── .gitignore    (archivos a ignorar en git)
├── package.json  (archivo de configuración del proyecto)
├── README.md     (archivo con instrucciones iniciales)
```

3. Migrar el HTML existente a componentes React básicos: Convertir cada sección del sitio (header, contacto, lista de doctores) en un componente reutilizable React.

4. **Integrar las funcionalidades JavaScript al entorno React:** Extraer scripts existentes y convertir en lógica de React.
5. **Añadir estilos Bootstrap:**

```
npm install Bootstrap

import "bootstrap/dist/css/bootstrap.min.css";
```

6. **Añadir partes del sitio en React:** Implementar primero funcionalidades simples (como menús y formularios), luego funcionalidades más específicas.

## Ejemplo de cómo migrar la sección Contacto a React:

Para migrar la sección de contacto a React, transformaremos el código estático de HTML en un componente React funcional. Seguiremos un enfoque en el que haremos que los inputs sean controlados mediante **React state** usaremos **JSX** para estructurar la UI y habilitaremos el manejo dinámico de la interacción con el formulario.

**1 . Configura el estado local para manejar los inputs**
En React, los campos de formulario son controlados mediante valores de estado **(`state`),** lo que garantiza que el contenido del formulario se mantenga sincronizado con la UI.

2. **Eventos en el formulario**
Usaremos el evento **`onChange`** para actualizar el valor de los inputs y **`onSubmit`** para manejar el envío del formulario.

3. **Estructura del componente:**
- Creamos un nuevo componente llamado **`ContactSection`.**
- Migramos el HTML directamente a JSX (agregando las modificaciones necesarias, como atributos propios de React).
- Utilizamos React Hooks **(`useState`)** para manejar el estado.

Código en React:

```
import React, { useState } from "react";

function ContactSection() {
  // Estado para cada campo del formulario
  const [formData, setFormData] = useState({
    nombre: "",
    email: "",
    telefono: "",
    asunto: "",
    mensaje: "",
  });

  const [formResult, setFormResult] = useState(""); // Para mostrar el resultado del envío del formulario

  // Manejar el cambio en los campos de entrada
  const handleInputChange = (event) => {
    const { id, value } = event.target; // Obtiene el "id" y "value" del input que cambió
    setFormData((prevData) => ({
      ...prevData,
      [id]: value, // Actualiza el valor del campo correspondiente
    }));
  };

  // Manejar el envío del formulario
  const handleSubmit = (event) => {
    event.preventDefault(); // Evita la recarga de la página
    console.log("Formulario enviado:", formData);
```

```
    // Simulación de validación
    if (!formData.nombre || !formData.email || !formData.mensaje) {
      setFormResult("Por favor completa los campos obligatorios.");
    } else {
      setFormResult("¡Mensaje enviado correctamente!");
    }

    // Reiniciar formulario, vacía los campos después de que el usuario lo haya enviado asignando valores vacíos
    setFormData({
      nombre: "",
      email: "",
      telefono: "",
      asunto: "",
      mensaje: "",
    });
  };

  return (
    <section className="contact-section py-2">
      <div className="container">
        <h1 className="text-center mb-4">Contáctanos</h1>
        <div className="contact-form">

          {/* Formulario implementado con React */}
          <form onSubmit={handleSubmit} className="row g-3">
            <div className="col-md-6">
              <label htmlFor="nombre" className="form-label">
                Nombre:
              </label>
              <input
                type="text"
                className="form-control"
                id="nombre"
                placeholder="Ingrese nombre y apellido"
                value={formData.nombre} // Input controlado
                onChange={handleInputChange} // Maneja el cambio
              />
            </div>
            <div className="col-md-6">
              <label htmlFor="email" className="form-label">
                Correo electrónico:
              </label>
              <input
                type="email"
                className="form-control"
                id="email"
                placeholder="nombre@dominio.com"
                value={formData.email}
                onChange={handleInputChange}
              />
            </div>
            <div className="col-md-6">
              <label htmlFor="telefono" className="form-label">
                Teléfono móvil:
              </label>
              <input
                type="text"
                className="form-control"
                id="telefono"
                placeholder="Formato: +56 9 12345678"
                value={formData.telefono}
                onChange={handleInputChange}
              />
            </div>
            <div className="col-md-6">
              <label htmlFor="asunto" className="form-label">
                Asunto:
              </label>
              <input
                type="text"
```

```
              className="form-control"
              id="asunto"
              placeholder="Motivo de su consulta"
              value={formData.asunto}
              onChange={handleInputChange}
            />
          </div>
          <div className="col-12">
            <label htmlFor="mensaje" className="form-label">
              Mensaje:
            </label>
            <textarea
              id="mensaje"
              className="form-control"
              rows="3"
              placeholder="Ingrese su mensaje"
              value={formData.mensaje}
              onChange={handleInputChange}
            ></textarea>
          </div>
          <div className="col-12">
            <button type="submit" className="btn btn-primary w-100">
              Enviar mensaje
            </button>
          </div>
        </form>
      </div>
      {/*  mostrar un mensaje de feedback al usuario después de que envía el formulario */}
      <div id="form-result" className="mt-1">
        {formResult && <p>{formResult}</p>}
      </div>
    </div>

    {/* Mapa de ubicación */}
    <div className="contact-map mt-5">
      <h2>Nuestra ubicación</h2>
      <h3>Visítanos en: Calle Falsa 123, Ciudad Falsa.</h3>
      <iframe
        src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d3329.997087571647!2d-70.6482!3d-
33.4372!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.1!3m3!1m2!1s0x0%3A0x0!2zMzPCsDI2JzE0LjAiUyA3MMKwMzgnNTMuNSJX!5e0!3m2!1
ses!2scl!4v1635959123456!5m2!1ses!2scl"
        width="100%"
        height="400"
        style={{ border: "0" }}
        allowFullScreen=""
        loading="lazy"
        referrerPolicy="no-referrer-when-downgrade"
      ></iframe>
    </div>
  </section>
 );
}

      export default ContactSection;
```

## Explicación del código migrado a React:

Estado del formulario **(`formData`):**
- Cada cambio en los inputs actualiza este estado usando **`onChange`.**
- Los valores de los inputs **(`value`) y textarea** están vinculados al estado **`formData`** para que React mantenga el control sobre su contenido.

Envío del formulario **(`handleSubmit`)**
- Cuando el usuario envía el formulario, se muestra un mensaje en **`formResult`.**

**Mapa:**
- El iframe del mapa se reutiliza directamente como estaba en el HTML, ya que React permite incluir contenido externo sin cambios.

**Ventajas**
Código más dinámico y organizado:
- Los inputs están controlados, lo que facilita leer y manejar los valores en tiempo real.
- React centraliza la lógica del formulario (en lugar de depender de scripts separados en JavaScript).
- La sección completa ahora está encapsulada como un componente React **(`ContactSection`),** que puedes importar y reutilizar en cualquier parte de tu aplicación.

Alumna: Camila Peña R
Curso: Desarrollo Front End
12-12-2024