



**EDUCACIÓN**  
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO  
NACIONAL DE MÉXICO®



**TECNOLÓGICO NACIONAL DE MÉXICO**

**INSTITUTO TECNOLÓGICO DE TLÁHUAC**

*“LA ESENCIA DE LA GRANDEZA RADICA EN LAS RAÍCES”*

**INGENIERÍA EN SISTEMA COMPUTACIONALES**

***Actividad.***

***“Examen Unidad 4.”***

*ALUMNO:*

*➤ Rodríguez Rodríguez Jorge Antonio*

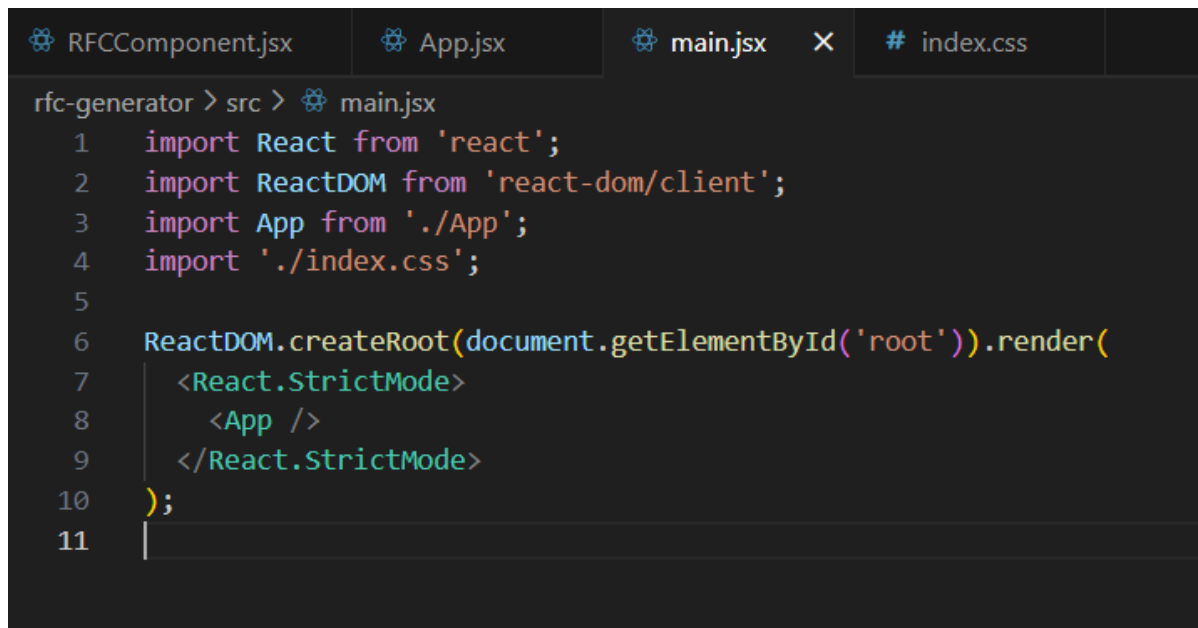
*DOCENTE: Ing. Roldan Aquino Segura*

*GRUPO: 7S2*

*FECHA: 28/Noviembre/2024*



Después de crear nuestro nuevo proyecto de RFC, vamos a trabajar en nuestra primera ventana que será nuestro Main en el cual vamos a agregar nuestros primeros componentes que nos permiten renderizar. El `React.StrictMode`, nos ayudará a ver nuestros errores, prácticamente la parte estricta. Y mandamos a llamar la función `</App>` el cual va a ser renderizado.



```
rfc-generator > src > main.jsx
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import App from './App';
4  import './index.css';
5
6  ReactDOM.createRoot(document.getElementById('root')).render(
7    <React.StrictMode>
8      <App />
9    </React.StrictMode>
10 );
11
```

Esta parte sirve como componente principal donde pasamos las props.

En donde se va a definir un objeto llamado `userInfo` que contiene información básica del usuario:

- `paterno`: Apellido paterno.
- `materno`: Apellido materno.
- `nombre`: Nombre de pila.
- `fecha Nacimiento`: Fecha de nacimiento en formato ISO (YYYY-MM-DD).

Este objeto se pasa como **props** (propiedades) al componente hijo `RFCComponent`.

La parte del **return**: Es lo que devuelve el componente, y el contenido será renderizado en la interfaz.

En el **JSX**: Es una extensión de JavaScript que permite escribir un HTML-like directamente dentro del código. Aquí:

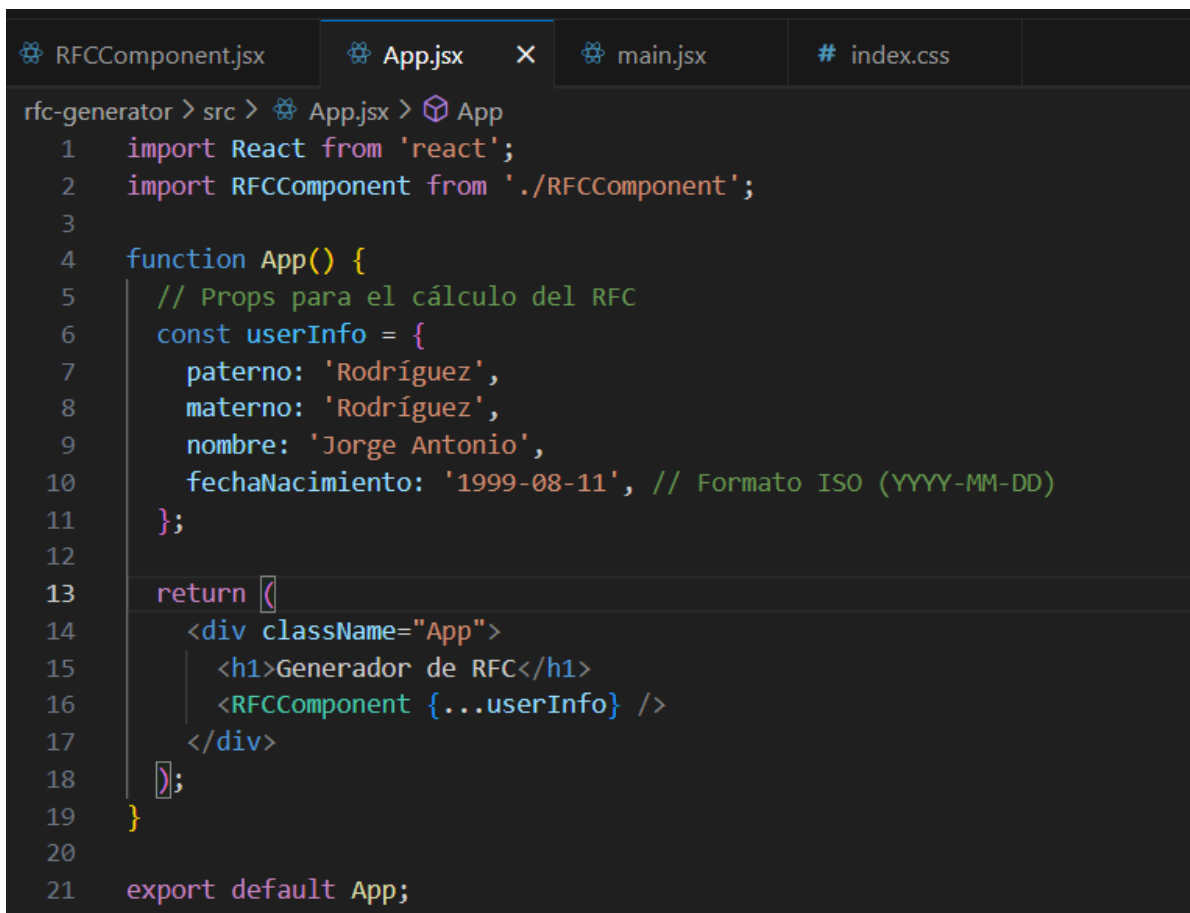
- `<div className="App">`:

- Crea un contenedor principal con la clase CSS App. Esta clase puede usarse para aplicar estilos.
- `<h1>Generador de RFC</h1>`:
  - Renderiza un encabezado con el título "Generador de RFC".
- `<RFCComponent {...userInfo} />`:
  - Renderiza el componente hijo RFCComponent.
  - El operador **spread (...)** pasa todas las propiedades del objeto userInfo como props individuales al componente hijo. Esto es equivalente a escribir:

Por último, el “**export default App;**”

Exporta el componente App como el módulo predeterminado.

Esto permite que sea importado en otros archivos, como en main.jsx, donde se renderiza en la raíz del DOM:



```
rfc-generator > src > App.jsx > App
1  import React from 'react';
2  import RFCComponent from './RFCComponent';
3
4  function App() {
5    // Props para el cálculo del RFC
6    const userInfo = {
7      paterno: 'Rodríguez',
8      materno: 'Rodríguez',
9      nombre: 'Jorge Antonio',
10     fechaNacimiento: '1999-08-11', // Formato ISO (YYYY-MM-DD)
11   };
12
13   return (
14     <div className="App">
15       <h1>Generador de RFC</h1>
16       <RFCComponent {...userInfo} />
17     </div>
18   );
19 }
20
21 export default App;
```

En esta parte les voy a explicar la parte del componente principal que es el RFCComponente el cuál:

- Se define un **componente funcional** llamado RFCComponent.
- Recibe un objeto de **props** como parámetro. Aquí, se usa **desestructuración** para extraer directamente las propiedades paterno, materno, nombre y fechaNacimiento de las props, en lugar de acceder a ellas como props.paterno, props.materno, etc.

En la función calcularRFC

Esta función realiza el cálculo del RFC basándose en las reglas proporcionadas.

Recibe cuatro parámetros: paterno, materno, nombre y fechaNacimiento, que son las mismas props pasadas al componente.

Habrà una subfunción la cual se llamarà obtenerprimervocal.

El cual el objetivo es encontrar la primera vocal en la palabra, después de la primera letra.

- Convierte la palabra a mayúsculas con .toUpperCase().
- Usa .slice(1) para omitir la primera letra de la palabra.
- Itera sobre las letras restantes con un bucle for.
- Si encuentra una vocal (AEIOU), la devuelve.
- Si no encuentra vocal, retorna una cadena vacía.

Para calcular las palabras clave, tendremos que extraer las letras necesarias para el cálculo del RFC:

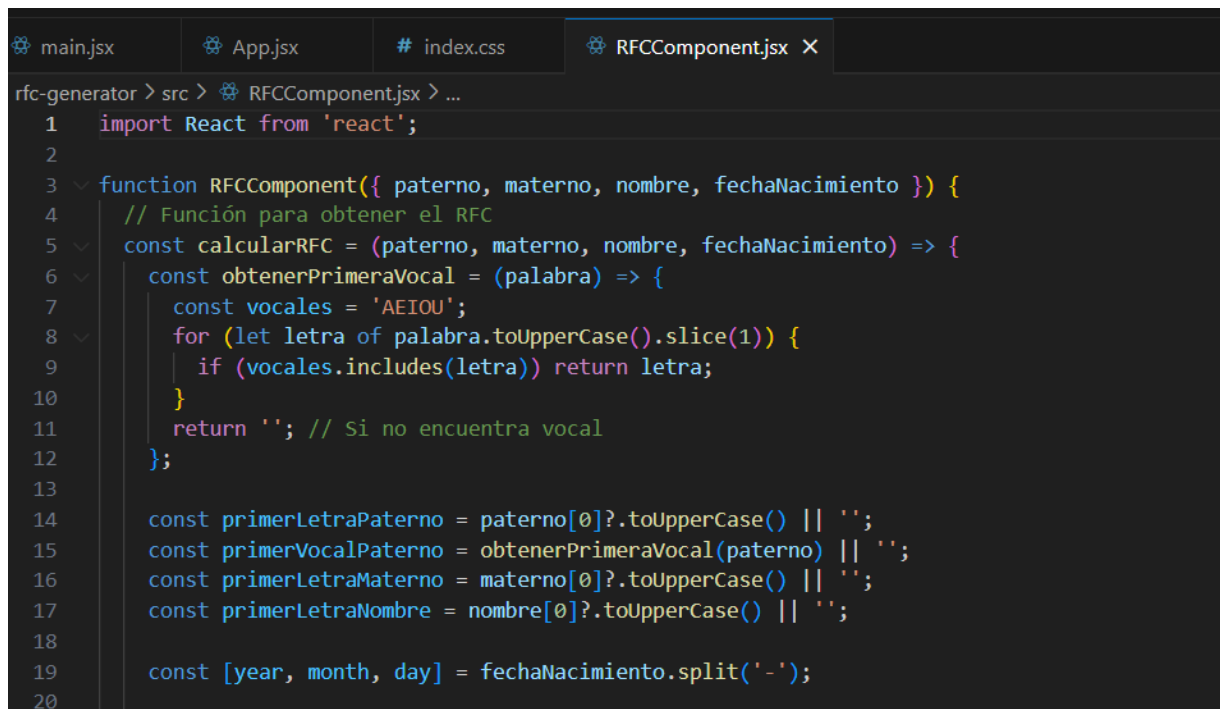
1. **primerLetraPaterno:** Primera letra del apellido paterno.
2. **primerVocalPaterno:** Primera vocal del apellido paterno (usando obtenerPrimeraVocal).
3. **primerLetraMaterno:** Primera letra del apellido materno.
4. **primerLetraNombre:** Primera letra del nombre.

El operador ?. (**encadenamiento opcional**):

- Si la palabra está vacía o es undefined, evita errores accediendo al índice [0]. Si no existe, devuelve undefined.
- En este caso, si no hay valor (undefined), retorna una cadena vacía (|| "").

Para dividir y formatear la fecha debemos extraer el año, mes y día de la fecha de nacimiento, para esto vamos a usar `.split('-')` para dividir la fecha (formato YYYY-MM-DD) en partes.

- year contiene el año (e.g., "1999").
- month contiene el mes (e.g., "08").
- day contiene el día (e.g., "11").



```
1 import React from 'react';
2
3 function RFCComponent({ paterno, materno, nombre, fechaNacimiento }) {
4   // Función para obtener el RFC
5   const calcularRFC = (paterno, materno, nombre, fechaNacimiento) => {
6     const obtenerPrimeraVocal = (palabra) => {
7       const vocales = 'AEIOU';
8       for (let letra of palabra.toUpperCase().slice(1)) {
9         if (vocales.includes(letra)) return letra;
10      }
11      return ''; // Si no encuentra vocal
12    };
13
14    const primerLetraPaterno = paterno[0]?.toUpperCase() || '';
15    const primerVocalPaterno = obtenerPrimeraVocal(paterno) || '';
16    const primerLetraMaterno = materno[0]?.toUpperCase() || '';
17    const primerLetraNombre = nombre[0]?.toUpperCase() || '';
18
19    const [year, month, day] = fechaNacimiento.split('-');
20  }
```

Aquí simplemente vamos a crear el RFC concatenando:

1. Primera letra del apellido paterno.
2. Primera vocal del apellido paterno.
3. Primera letra del apellido materno.
4. Primera letra del nombre.
5. Últimos 2 dígitos del año (`year.slice(-2)`).
6. Mes (`month`).
7. Día (`day`).

Después el uso de la función `calcularRFC` mandará a llamar a **calcularRFC** con las props recibidas (`paterno`, `materno`, `nombre`, `fechaNacimiento`).

Y simplemente almacenará el RFC generada en la constante rfc.

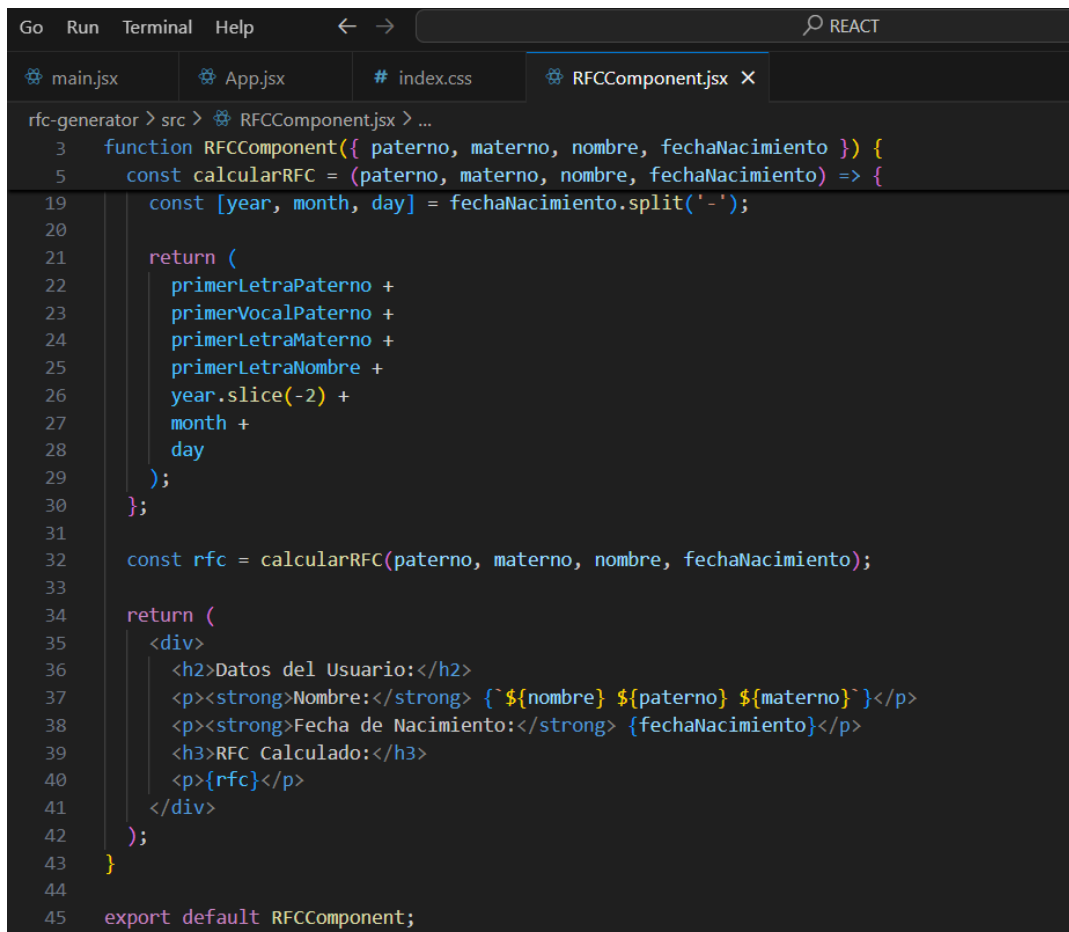
Por último, `<h2>` y `<p>` van a renderizan los datos del usuario (nombre completo y fecha de nacimiento).

Para la interpolación de variables se usará `{}` para insertar las variables (nombre, paterno, materno, fechaNacimiento) en el HTML.

La plantilla de cadena ``${nombre} ${paterno} ${materno}`` combina los valores con espacios entre ellos.

RFC: Muestra el RFC calculada (rfc).

Para finalizar esta parte haremos la exportación. Exporta el componente para que pueda usarse en otros archivos, como en App.jsx



```
Go Run Terminal Help  REACT
main.jsx App.jsx # index.css RFCComponent.jsx X
rfc-generator > src > RFCComponent.jsx > ...
3  function RFCComponent({ paterno, materno, nombre, fechaNacimiento }) {
5    const calcularRFC = (paterno, materno, nombre, fechaNacimiento) => {
19     const [year, month, day] = fechaNacimiento.split('-');
20
21     return (
22       primerLetraPaterno +
23       primerVocalPaterno +
24       primerLetraMaterno +
25       primerLetraNombre +
26       year.slice(-2) +
27       month +
28       day
29     );
30   };
31
32   const rfc = calcularRFC(paterno, materno, nombre, fechaNacimiento);
33
34   return (
35     <div>
36       <h2>Datos del Usuario:</h2>
37       <p><strong>Nombre:</strong> `${nombre} ${paterno} ${materno}`</p>
38       <p><strong>Fecha de Nacimiento:</strong> {fechaNacimiento}</p>
39       <h3>RFC Calculado:</h3>
40       <p>{rfc}</p>
41     </div>
42   );
43 }
44
45 export default RFCComponent;
```

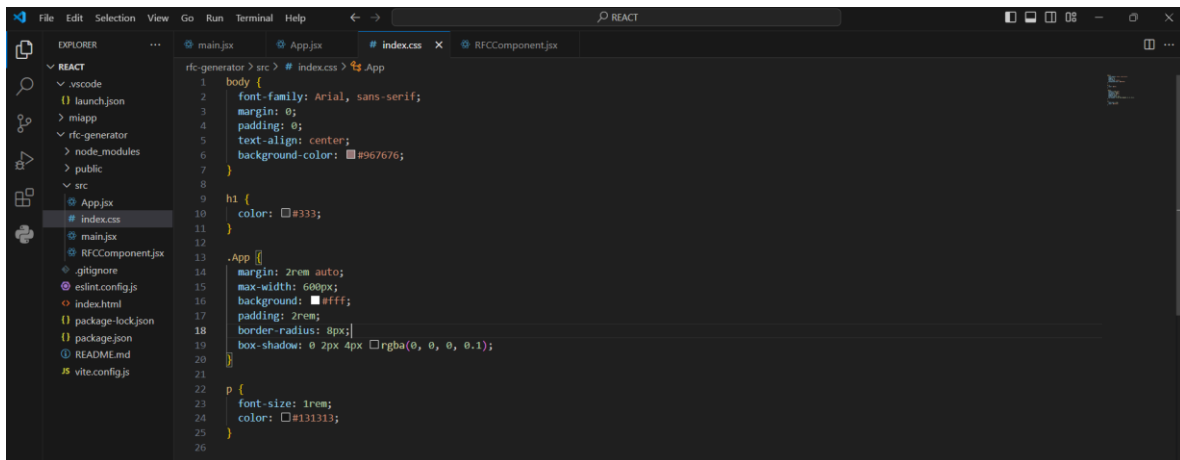
En esta ultima parte solamente vamos a trabajar en los estilos siguientes:

body: Establece la fuente, elimina márgenes/padding predeterminados, alinea el texto al centro, y le da un fondo gris claro a toda la página.

h1: Ajusta el color del encabezado principal a un gris oscuro para hacerlo legible pero menos agresivo que el negro.

.App: Aplica estilos al contenedor principal de la aplicación, centrándolo, dándole un fondo blanco, y agregando un poco de espacio interno, bordes redondeados, y una sombra suave para hacerlo visualmente más atractivo.

p: Ajusta el tamaño y color del texto de los párrafos para mejorar la legibilidad.



```
1 body {
2   font-family: Arial, sans-serif;
3   margin: 0;
4   padding: 0;
5   text-align: center;
6   background-color: #967676;
7 }
8
9 h1 {
10  color: #333;
11 }
12
13 .App {
14   margin: 2rem auto;
15   max-width: 600px;
16   background: #fff;
17   padding: 2rem;
18   border-radius: 8px;
19   box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
20 }
21
22 p {
23   font-size: 1rem;
24   color: #131313;
25 }
26
```

Este será nuestro resultado.

