DOCUMENTO DE ESTILO DE CÓDIGO Y BUENAS PRÁCTICAS

1. Convención de nombres: Se utiliza PascalCase para nombres de clases y componentes React
Por ejemplo:

```
export default function Formulario() { ... }
```

Métodos y Funciones:

- Usar camelCase para funciones privadas, públicas y auxiliares. Ejemplos:

```
// Para métodos públicos
const validarCampo = (campo, valor) => { ... };

// Para funciones privadas
const validarFechaEntrega = (fecha) => { ... };
```

Variables y Atributos:

- Utiliza nombres de variables claros y descriptivos.
- Usar camelCase para variables y atributos. Ejemplos:

```
const [fechaRetiro, setFechaRetiro] = useState("");
const calleRetiroRef = useRef(null);
```

Constantes:

- Usar UPPER_SNAKE_CASE para constantes. Ejemplo:

```
const GOOGLE_MAPS_API_KEY= "AIzaSyAJ5-3XydV_Zi8c6L086Zhjz24qaVu6q-8";
```

Prefijos/Sufijos:

- Usar prefijos como handle, validate, set y sufijos como Ref para referencias. Ejemplos:

```
const handleLoadScript = () => { ... };
const calleRetiroRef = useRef(null);
```

2. Formato del Código

- Indentación:

  Usar 4 espacios para la indentación del código. Ejemplo:

```
useEffect(() => {
        const hoy = new Date().toISOString().split("T")[0];
        setFechaMinima(hoy);
}, []);
```

Longitud de Línea:

- Limitar la longitud de las líneas a 80-120 caracteres. Ejemplo:

```
// Correcto
const scriptSrc =
https://maps.googleapis.com/maps/api/js?key=${GOOGLE_MAPS_API_KEY}&librari
es=places;

// Incorrecto (línea demasiado larga)
const scriptSrc =
https://maps.googleapis.com/maps/api/js?key=AIzaSyAJ5-3XydV_Zi8c6L086Zhjz24q
aVu6q-8&libraries=places;
```

Espacios en Blanco:

- Organizar el código con espaciado y sangría para facilitar la lectura. Esto ayuda a estructurar visualmente el código. Ejemplo:

```
const handleLoadScript = () => {
// Configuración del autocomplete
};

useEffect(() => {
        loadGoogleMapsScript(handleLoadScript);

return () => {
// Limpiar listeners
        };
}, []);
```

Llaves:

- Colocar las llaves en la misma línea que las definiciones de funciones o estructuras. Ejemplo:

```
if (!window.google || !window.google.maps) {
        const script = document.createElement("script");
```

```
script.src =
https://maps.googleapis.com/maps/api/js?key=${GOOGLE_MAPS_API_KEY}
&libraries=places;
script.onload = handleLoadScript;
document.body.appendChild(script);
} else {
        handleLoadScript();
}
```

3. Comentarios y Documentación
Comentarios de Línea:

- Comentar líneas clave para explicar por qué se hace algo. Ejemplo:

```
// Configurar el autocomplete para la dirección de retiro
const autocompleteRetiro = new
window.google.maps.places.Autocomplete(calleRetiroRef.current, { ... });
```

Documentación de Funciones:

- Documentar cada función con un breve comentario sobre su propósito y parámetros. Ejemplo:

```
/**
 * Valida la fecha de retiro en tiempo real.
 * @param {string} fecha - La fecha de retiro a validar.
 */
const validarFechaRetiro = (fecha) => { ... };
```

4. Estructura del Código y Distribución de Archivos

- Organización por Funcionalidad.
- Agrupar archivos en carpetas según su funcionalidad. Ejemplos:

  src/components/: Componentes de React como Formulario.js y DataForm.js.

  src/services/: Servicios y lógica externa, como la carga de scripts.

  src/utils/: Funciones auxiliares.

- Estructura de Directorios:

  public/: Archivos estáticos (p. ej., imágenes, íconos).

  src/: Código fuente de la aplicación.

  components/: Componentes de React.

  services/: Servicios externos y lógicos.

  utils/: Funciones auxiliares.

  assets/: Recursos como imágenes y estilos CSS.

  hooks/: Hooks personalizados de React.

5. Modularización:

- Dividir el código en módulos independientes y reutilizables. Cada módulo debe tener una única responsabilidad y ser fácil de mantener. Ejemplo:

  // src/services/mapService.js export const loadGoogleMapsScript = (onLoadCallback) => { ... };

6. Archivos de Configuración:

- Mantener archivos de configuración en el directorio raíz para gestionar dependencias y configuraciones del proyecto.

7. Operadores aritméticos y simplificaciones:

- Simplificar cálculos con operadores como +=, -=, *= y +=, evitando escribir expresiones largas y redundantes