EXPLICACIÓN PASO A PASO DEL CÓDIGO

1.- INICIALIZACIÓN DE CAMPOS Y BOTONES

```
1  //obtenemos referencias del DOM
2  let fieldset2 = document.getElementById('f2');
3  let fieldset3 = document.getElementById('f3');
4  fieldset2.disabled = true;
5  fieldset3.disabled = true;
6
7  const btnEnvio = document.getElementById('btEnvio');
8  btnEnvio.disabled = true;
9
```

Qué hace:

- Obtiene referencias a los fieldset 2 y 3 y al botón de envío.
- Los desactiva al inicio (disabled = true) porque no queremos que el usuario avance hasta llenar correctamente los datos del primer fieldset.

2.- REGLAS DE VALIDACIÓN

```
10 //definimos reglas de validacion
11 const reglas = {
12   nombre: /^[a-zA-Z\s]+$/,
13   apellido: /^[a-zA-Z\s]+$/,
14   telefono: /^\d{0,9}$/,
15   noches: /^(?:[0-9]|[1-8][0-9]|90)$/
16 }
```

Qué hace:

- Define expresiones regulares (RegEx) para validar campos:

- nombre y apellido → solo letras y espacios.
- telefono → máximo 9 dígitos.
- noches → entre 0 y 90.

3.- VALIDACIOÓN DINÁMICA DEL PRIMER FIELSET(.f1)

```
18 //capturamos primer fieldset y validamos
19 const fieldset1 = Array.from(document.querySelectorAll
   ('.f1 .contdivf1 input'));
20
```

Selecciona todos los inputs dentro de .f1 .contdivf1 y los convierte a un **array** con Array.from (porque querySelectorAll devuelve un NodeList).

```
21 fieldset1.forEach(input => {
22   input.addEventListener('input', e => {
23     const valor = e.target.value.trim();
24     const spanError = e.target.nextElementSibling;
25     const evaluar = reglas[e.target.id];
26
27     if (valor === "") {
28       e.target.classList.remove('inputError');
29       spanError.classList.remove('error');
30       spanError.style.display = 'none';
31       return;
32     }
33     if (!evaluar.test(valor)) {
34       spanError.textContent = "formato inválido";
35       e.target.classList.add('inputError');
36       spanError.classList.add('error');
37       spanError.style.display = 'block';
38       return;
39     } else {
40       spanError.textContent = "";
41       e.target.classList.remove('inputError');
42       spanError.classList.remove('error');
43       spanError.style.display = 'none';
44       return;
45     }
46   })
47
48 })
```

1.- .addEventListener('input', e => { ... })

¿Qué hace?

Le dice al navegador:

"Cada vez que el usuario escriba, borre o modifique el contenido de este input, ejecuta esta función".

- input → es un elemento <input> del formulario.
- input → evento que se dispara en tiempo real mientras el usuario escribe.
- e → es el **objeto evento**.

2.- ¿Qué es e (el evento)?

e es un objeto que **contiene información sobre lo que ocurrió**.

Incluye datos como:

- Qué elemento generó el evento
- Qué tipo de evento fue
- Información del teclado, mouse, etc.

3.- e.target (EXPLICACIÓN)

¿Qué es **target?**

e.target es **el elemento HTML exacto que disparó el evento**.

En este caso:

- Si el usuario escribe en el input **Nombre** → e.target es ese input.
- Si escribe en **Apellido** → e.target es ese otro input.

"e.target === input_que_el_usuario_está_usando"

4.- e.target.value

- Obtiene el **texto que el usuario escribió** dentro del input.
- Es lo mismo que hacer:

Input.value; pero de forma dinámica, sin importar cual input sea.

5.- trim() → Elimina los espacios en blanco al inicio y al final del texto.
Se usa para:

- Evitar que espacios vacíos pasen como texto válido.
- Detectar campos realmente vacíos.

6.- e.target.nextElementSibling

const spanError = e.target.nextElementSibling;

¿Qué significa?

Obtiene el **siguiente elemento HTML que está justo después del input**.
En el HTML tenemos:

```
17        <input type="text" id="nombre" name="Nombre" placeholder="Nombre" autofocus
18          pattern="[A-Za-zÁÉÍÓÚÜÑáéíóúüñ\s]+" required>
19        <span></span>
```

Entonces:

***e.target.nextElementSibling === ***

Ese se usa para:

- Mostrar mensajes de error
- Ocultarlos o modificarlos dinámicamente

7.- etarget.id → Obtiene el atributo **id** del input que disparó el evento
Ejem:
<input id="nombre> ==> e.target.id = 'nombre';

8.- reglas[e.target.id]

```
25    const evaluar = reglas[e.target.id];
```

Tienes un objeto de **reglas**

```
const reglas = {
  nombre: /^[a-zA-Z\s]+$/,
  apellido: /^[a-zA-Z\s]+$/,
  telefono: /^\d{0,9}$/,
  noches: /^(?:[0-9]|[1-8][0-9]|90)$/
}
```

Entonces:

- Si el input tiene id="nombre" → usa reglas.nombre
- Si el input tiene id="telefono" → usa reglas.telefono

Es equivalente a:

const evaluar = reglas["nombre"];

Entonces:

- Cada input se valida con su propia regla
- Todo funciona con una sola función

- return **detiene la función** en el punto donde se usa.
- En este caso evita que:
  - Se sigan ejecutando líneas innecesarias después de detectar que el campo está vacío o es inválido.
  - Se mezclen los estilos de error (por ejemplo, que un campo vacío termine mostrando el mensaje de "formato inválido").
- Es una manera de **controlar el flujo de la función de validación**.

Habilitación de fieldset2 si todo está correcto:

```
50  //habilitamos fieldset
51  fieldset1.forEach(input => {
52    input.addEventListener('input', () => {
53      const campoVacio = fieldset1.some(i => i.value ===
      '');
54      const campoError = fieldset1.some(c => c.classList.
      contains('inputError'));
55      fieldset2.disabled = campoVacio || campoError;
56    })
57  })
```

- .some() revisa si **algún input está vacío o tiene error**.
- Si hay algo mal, fieldset2 permanece deshabilitado.

4.- CONFIGURACIÓN DE FECHAS (FIELDSET2)

```
60  //fechas
61  const fecha = new Date();
62  const anio = fecha.getFullYear()
63  const mes = String(fecha.getMonth() + 1).padStart(2,
    '0');
64  const dia = String(fecha.getDate()).padStart(2, '0');
65  const fechaActual = `${anio}-${mes}-${dia}`;
66
```

Qué hace:

- Obtiene la fecha actual y la formatea como YYYY-MM-DD para usar en los inputs de tipo date.

```
67  //valores de input-fecha
68  const inpFechaInicio = document.getElementById
    ('fechaInicio');
69  inpFechaInicio.setAttribute('min', fechaActual);
70
71  const inpFechaSalida = document.getElementById
    ('fechaSalida');
72  inpFechaSalida.setAttribute('min', fechaActual);
```

Limita la selección de fechas para que no puedan elegir **fechas pasadas**.

5.- VALIDACIÓN DE FECHAS Y RADIOS

VALIDACIÓN DE FECHAS

```
/*-----validación de fechas------- */
const fechas = Array.from(document.querySelectorAll('.f2 input[type=date]'));
let fechVacia;
let radVacio = true;
const radios = Array.from(document.querySelectorAll('.f2 input[type=radio]'));

//valores de fechas
let ordenIncorrecto = false;
fechas.forEach(f => {
  f.addEventListener('input', () => {
    //verificar si algun campo de fecha está vacío
    fechVacia = fechas.some(fech => fech.value.trim() === '');

    //validar orden
    if (inpFechaInicio.value && inpFechaSalida.value) {
      ordenIncorrecto = new Date(inpFechaSalida.value) < new Date(inpFechaInicio.value);
    }else{
      ordenIncorrecto=true;
    }
    //estados
    fieldset3.disabled = fechVacia || radVacio || ordenIncorrecto;
    btnEnvio.disabled = radVacio || fechVacia || ordenIncorrecto;

    if (inpFechaInicio.value && inpFechaSalida.value && ordenIncorrecto) {
      inpFechaSalida.style.borderColor = 'red';
      inpFechaSalida.setAttribute('title', 'Fecha de salida no debe ser menor');
    } else {
      inpFechaSalida.style.borderColor = '';
      inpFechaSalida.removeAttribute('title', 'Fecha de salida no debe ser menor');
    }
  })
})
```

- **Selección de elementos:**

```
const fechas = Array.from(document.querySelectorAll('.f2 input[type=date]'));
```

Qué hace:

- document.querySelectorAll('.f2 input[type=date]') busca **todos los inputs de tipo date dentro de .f2**.
- querySelectorAll devuelve un **NodeList**, que es similar a un array, pero no tiene todos los métodos de array.
- Array.from() convierte ese NodeList en un **array real**, así podemos usar métodos como .forEach(), .some(), etc.

fechas ahora es un array con esos dos inputs.

```
const fechas = Array.from(document.querySelectorAll('.f2 input[type=date]'));
let fechVacia;
let radVacio = true;
```

Qué hace:

- fechVacia → variable que almacenará si algún campo de fecha está vacío (true/false).
- radVacio → inicialmente true, indica si **ningún radio está seleccionado**.

```
let ordenIncorrecto = false;
```

Variable que indica si el **orden de fechas es incorrecto**, es decir, si la fecha de salida es anterior a la fecha de inicio.

- **Escuchando cambios en las fechas:**

```
82  fechas.forEach(f => {
83    f.addEventListener('input', () => {
```

• .forEach() recorre **cada input de fecha**.
• *.addEventListener('input', callback)* → cada vez que el usuario cambia la fecha (escribe o selecciona), se ejecuta la función.

• Verificar si hay fechas vacías:

```
84      //verificar si algun campo de fecha está vacío
85      fechVacia = fechas.some(fech => fech.value.trim() === '');
```

Qué hace .some():
- Recorre el array fechas y devuelve true si **al menos un elemento cumple la condición**.
- *Condición: fech.value.trim() === ''* → el input está vacío (sin espacios).

Resultado:
- fechVacia = true → si falta alguna fecha.
- fechVacia = false → si ambos inputs tienen valor.

• Validar el orden de las fechas

```
87      //validar orden
88      if (inpFechaInicio.value && inpFechaSalida.value) {
89        ordenIncorrecto = new Date(inpFechaSalida.value) < new Date(inpFechaInicio.value);
90      }else{
91        ordenIncorrecto=true;
92      }
```

- **Primero** verifica que ambos inputs tengan valor (inpFechaInicio.value y inpFechaSalida.value).
- **Si ambos tienen valor**:
  - Convierte las fechas de texto a objetos Date.
  - Compara: si la **fecha de salida es menor que la de inicio**, ordenIncorrecto = true.
  - Si la fecha de salida es igual o mayor, ordenIncorrecto = false.
- **Si falta alguna fecha**, automáticamente ordenIncorrecto = true.
  - Esto fuerza la desactivación hasta que el usuario complete ambos campos.

• habilitar o deshabilitar controles

```
93      //estados
94      fieldset3.disabled = fechVacia || radVacio || ordenIncorrecto;
95      btnEnvio.disabled = radVacio || fechVacia || ordenIncorrecto;
96
```

• Si **alguna condición de error** es true:
  - Se desactiva fieldset3 (la siguiente sección del formulario).
  - Se desactiva el botón de envío (btnEnvio).
• Lógica: **solo se puede avanzar si**:
  1. Ninguna fecha está vacía.
  2. Al menos un radio está seleccionado.
  3. Las fechas están en orden correcto.

• Mostrar borde rojo y tooltip(title) si el orden es incorrecto

```
    if (inpFechaInicio.value && inpFechaSalida.value && ordenIncorrecto) {
      inpFechaSalida.style.borderColor = 'red';
      inpFechaSalida.setAttribute('title', 'Fecha de salida no debe ser menor');
    } else {
      inpFechaSalida.style.borderColor = '';
      inpFechaSalida.removeAttribute('title', 'Fecha de salida no debe ser menor');
    }
  })
```

- Si ambas fechas tienen valor **y** la fecha de salida es menor:
    - El borde del input de salida se vuelve **rojo** (borderColor = 'red').
    - Se agrega un **tooltip** (title) para mostrar un mensaje de error.
- Si no hay error, se **restauran los estilos y se elimina el tooltip**.

VALIDACION DE RADIOS

```
radios.forEach(f => {
    f.addEventListener('change', () => {
    radVacio = !radios.some(r => r.checked);

    // Recalcular fechas
    fechVacia = fechas.some(fech => fech.value.trim() === '');
    if (inpFechaInicio.value && inpFechaSalida.value) {
      ordenIncorrecto = new Date(inpFechaSalida.value) < new Date(inpFechaInicio.value);
    }
    if(inpFechaInicio.value.trim() === '' || inpFechaSalida.value.trim() === ''){
      ordenIncorrecto = true;
    }

    // Actualizar estados
    fieldset3.disabled = radVacio || fechVacia || ordenIncorrecto;
    btnEnvio.disabled = radVacio || fechVacia || ordenIncorrecto;

    // Estilos y tooltip
    if (ordenIncorrecto) {
      // inpFechaSalida.style.borderColor = 'red';
      inpFechaSalida.setAttribute('title', 'Fecha de salida no debe ser menor');
    } else {
      inpFechaSalida.style.borderColor = '';
      inpFechaSalida.removeAttribute('title');
    }
  });

})
```

DESGLOCE

```
107  radios.forEach(f => {
108      f.addEventListener('change', () => {
109      radVacio = !radios.some(r => r.checked);
110
```

- radios es un array con todos los **inputs de tipo radio** dentro del fieldset2.
- .forEach(f => ...) recorre **cada radio individual** (f) para agregar un **evento de escucha**.
- .addEventListener('change', callback) significa:
  - Cada vez que el usuario **selecciona o cambia** un radio, se ejecuta la función.

Validar si algun radio está seleccionado:

```
109      radVacio = !radios.some(r => r.checked);
110
```

- .some(r => r.checked) revisa si **al menos un radio está seleccionado**.
  - Devuelve true si **al menos un radio tiene checked = true**.
- !radios.some(...) invierte el resultado:
  - radVacio = true → **ningún radio está seleccionado**.
  - radVacio = false → **al menos un radio está seleccionado**.

.

Importante: Esto sirve para **habilitar o deshabilitar el siguiente fieldset y el botón de envío**, porque no se puede continuar si no se ha seleccionado ningún radio.

Recalcular fechas:

```
112  // Recalcular fechas
113  fechVacia = fechas.some(fech => fech.value.trim() === '');
114  if (inpFechaInicio.value && inpFechaSalida.value) {
115    ordenIncorrecto = new Date(inpFechaSalida.value) < new Date(inpFechaInicio.value);
116  }
```

- **Paso 1:** Revisa si **algún campo de fecha está vacío**.
  - .some(fech => fech.value.trim() === '') → true si algún input está vacío.
- **Paso 2:** Si ambos inputs tienen valor, revisa **el orden de fechas**:
  - ordenIncorrecto = new Date(salida) < new Date(inicio) → true si la fecha de salida es menor que la de inicio.
- **Paso 3:** Si **alguno de los inputs está vacío**, fuerza ordenIncorrecto = true para prevenir avanzar con datos incompletos.

Esto **sincroniza las fechas con los radios**, porque cambiar un radio también puede afectar si se permite avanzar en el formulario.

Actualizar estado de fieldset3 y boton de envio

```
121  // Actualizar estados
122  fieldset3.disabled = radVacio || fechVacia || ordenIncorrecto;
123  btnEnvio.disabled = radVacio || fechVacia || ordenIncorrecto;
124
```

- Si alguna condición de error es true:
  - radVacio → no hay radio seleccionado

- fechVacia → algún input de fecha está vacío
- ordenIncorrecto → la fecha de salida es menor que la de inicio

- **Entonces:**
  - fieldset3.disabled = true → se deshabilita la sección siguiente del formulario.
  - btnEnvio.disabled = true → se deshabilita el botón de envío.

Esto **garantiza que no se pueda continuar hasta que todas las condiciones sean correctas**.

Mostrar estilos de error y Tooltip

```
125     // Estilos y tooltip
126     if (ordenIncorrecto) {
127       inpFechaSalida.setAttribute('title', 'Fecha de salida no debe ser menor');
128     } else {
129       inpFechaSalida.style.borderColor = '';
130       inpFechaSalida.removeAttribute('title');
131     }
132   });
```

- Si el orden es incorrecto:
  Se agrega un **tooltip** (title) al input de salida con el mensaje de error.
- Si el orden es correcto:
  Se limpia el borde rojo (style.borderColor = '') y se quita el tooltip.

Esto da retroalimentación visual al usuario sobre qué está mal.

6.- ENVÍO DE FORMULARIO

- **Obtener referencias de elementos del DOM(para manipularlas después)**

```
136 /*-------Elementos del DOM---------- */
137 const form = document.getElementById('formulario');
138 const contDatos = document.getElementById('contDiv');
139 const contSpinner = document.getElementById('contSpinner');
140
```

Qué hace:

- form → referencia al formulario completo.
- contDatos → contenedor donde se mostrarán los datos enviados.
- contSpinner → contenedor del spinner (carga falsa).

- **Ocultar contenedores**

```
141 //ocultar contenedores
142 contDatos.style.display = 'none';
143 contSpinner.style.display = 'none';
144
```

Por qué:
- Al cargar la página:
  - El formulario debe verse
  - El spinner y los datos **NO**

Se controla la visibilidad solo con CSS (display).

- **Evento click del boton de envío**

```
145  //Evento click
146  btnEnvio.addEventListener('click', e => {
```

- Se ejecuta cuando el usuario presiona **Enviar**.
- **e** es el **evento**.

• 3.1 Prevenir el envío real del formulario

```
147      e.preventDefault();
```

Importante:

- Evita que el formulario:
  Se envíe al servidor
  Recargue la página

Esto permite manejar todo **con JavaScript**.

• Validación nativa

```
148  //validacion nativa
149  if (!form.checkValidity()) {
150    form.reportValidity();
151    return;
152  }
```

Qué hace:
- checkValidity():
• Devuelve true si **todos los campos cumplen** sus reglas HTML (required, min, pattern, etc.)

- **Si hay errores:**
• reportValidity() muestra los mensajes del navegador
• return corta la ejecución (no continúa).
Esto es una **segunda capa de seguridad** además de la validación JS.

• Ocultamos el formulario:

```
153  //ocultamos el formulario
154  form.style.display = 'none';
```

- El formulario ya no se muestra
- Simula que el usuario **envió los datos**

• Mostramos el spinner(simula carga)

```
156  //mostramos solo el spinner
157  contSpinner.style.display = 'block';
158  contSpinner.innerHTML = '';
159
```

- Se muestra el contenedor del spinner
- Se limpia por si había algo antes

• crear spinner dinamicamente

```
160  //crear spinner dinamicamente
161  const spinner = document.createElement('div');
162  spinner.classList.add('cargando');
163  contSpinner.appendChild(spinner);
164
```

- Se crea un
- Se le agrega una clase CSS (cargando)
- Se inserta dentro del contenedor

El **efecto visual** depende del CSS de .cargando.

• Simulación de espera

```
165  //simulacion de espera
166  setTimeout(() => {
```

- Espera **2500 ms (2.5 segundos)**
- Simula el tiempo de respuesta del servidor

- **Ocultar spinner**

```
167     //ocultamos el spinner
168     contSpinner.style.display = 'none';
169     contSpinner.innerHTML = '';
```

- Se oculta el spinner
- Se limpia el contenedor

- **Mostrar el contenedor de datos**

```
171     //mostramos contenedor de datos
172     contDatos.style.display = 'block';
173     contDatos.classList.add('contDatosDinamicos');
174     contDatos.innerHTML = '';
175
```

- Se muestra el contenedor
- Se le aplica una clase para estilos
- Se limpia por seguridad

- **Crear título y mensaje → se crean elementos html desde JS y se agregan al DOM**

```
176     //crear tutulo y msn
177     const h2 = document.createElement('h2');
178     h2.textContent = 'DATOS DE RESERVA';
179
180     const p = document.createElement('p');
181     p.textContent = 'Hola; gracias por su preferencia';
182
183     contDatos.append(h2, p);
```

- **Obtener datos del formulario (FormData)**

```
185     const forDat = new FormData(form);
186
```

Qué hace:

- Es un objeto especial de JS
- Extrae **todos los campos del formulario**
- Incluye inputs, selects, radios, etc.

Cada campo se guarda como:

nombreDelCampo → valorIngresado

ejem

nombre → Juan
apellido → Pérez
telefono → 123456789

```
187     for (const [campo, valor] of forDat.entries()) {
188       const parrafo = (document.createElement('p'));
189       parrafo.innerHTML = `
190     <span class="campo">${campo}</span>
191     <span class="valor">${valor}</span>`;
192
193       contDatos.appendChild(parrafo);
194     }
```

- **¿Qué es forDat.entries()?**

Devuelve un **iterador** con pares:
[campo, valor]

- **El for...of**

```
187     for (const [campo, valor] of forDat.entries())
```

◆**for...of**
- Recorre **valores**, no índices
- Ideal para:
  - Arrays
  - Iteradores
  - FormData

◆ **[campo, valor]**

Esto es **desestructuración de arrays**.
Cada elemento que devuelve entries() es un array:
["nombre", "Juan"]

Entonces:
campo = "nombre"
valor = "Juan"

Sin desestructuración sería:
for (const par of forDat.entries()) {
const campo = par[0];
const valor = par[1];
}

- **Crear el elemento <p>(**Crea dinamicamente un parrafo html**)**

```
188     const parrafo = (document.createElement('p'));
```

Todavía no se muestra en la página,solo existe en memoria

- **Usar innerHTML**

```
189       parrafo.innerHTML = `
190     <span class="campo">${campo}</span>
191     <span class="valor">${valor}</span>`;
```

Qué hace:
- Inserta HTML dentro del <p>
- Usa **template literals** (backticks `)

Qué es ${campo} y ${valor}:
- Interpolación de variables
- Sustituye por los valores reales

Ejem

```
<p>
  <span class="campo">nombre</span>
  <span class="valor">Juan</span>
</p>
```

Se usan para poder **estilizar cada parte con CSS**.

- **Insertar el párrafo en el DOM**

```
193        contDatos.appendChild(parrafo);
```

- **BOTÓN CERRAR (recargar página)**

```
195    //boton cerrar dinámico
196    const btnCerrar = document.createElement('button');
197    btnCerrar.setAttribute('title', 'Cerrar')
198    btnCerrar.classList.add('btnClosed');
199    btnCerrar.textContent = 'X';
200    contDatos.appendChild(btnCerrar);
201    btnCerrar.addEventListener('click', () => location.reload());
```

- Se crea un botón
- Se le asignan atributos y estilos
- Recarga la página
- Restablece el formulario al estado inicial

7.- EVENTO RESET DEL FORMULARIO

```
206  //evento al boton reset
207  form.addEventListener('reset', () => {
208    //estados iniciales(desactivados)
209    fieldset2.disabled = true;
210    fieldset3.disabled = true;
211    btnEnvio.disabled = true;
212  })
```

Qué hace:

- Cuando el usuario presiona **Reset**:
  - Se deshabilitan nuevamente los fieldsets
  - Se desactiva el botón de envío

Devuelve el formulario a su **estado inicial**.

MÉTODOS Y FUNCIONES USADAS

Método / Función	Qué hace
document.getElementById()	Selecciona un elemento por su ID.
document.querySelectorAll()	Selecciona todos los elementos que coinciden con un selector CSS.
Array.from()	Convierte un objeto iterable (NodeList) a un array real.
addEventListener('input', callback)	Escucha cambios de un input mientras se escribe.
addEventListener('change', callback)	Escucha cambios en radios o selects.
trim()	Elimina espacios al inicio y final de un string.
RegExp.test(valor)	Verifica si un valor cumple la expresión regular.
nextElementSibling	Obtiene el siguiente hermano en el DOM (útil para el span de error).
.some()	Verifica si **algún elemento** del array cumple una condición.
new Date(valor)	Convierte un string en fecha para comparar.
.classList.add/remove()	Agrega o quita clases CSS a un elemento.
.setAttribute()	Establece un atributo HTML (como min o title).
.removeAttribute()	Quita un atributo HTML.
document.createElement()	Crea un nuevo elemento HTML.
FormData(form)	Permite recorrer los campos del formulario.
form.checkValidity()	Verifica si el formulario cumple con HTML5 validations.
form.reportValidity()	Muestra mensajes de error nativos de HTML5 si hay campos inválidos.
setTimeout(callback, ms)	Ejecuta un bloque de código después de un tiempo definido en milisegundos.
location.reload()	Recarga la página actual.
entries()	Devuelve un iterador con pares [clave, valor] de un objeto iterable (FormData, Map, Array, etc.)