CURSO DE INICIACIÓN

React y Librería SpaDit

DEPARTAMENTO DE INFORMÁTICA TRIBUTARIA

Mayo 2025

React y Librería SpaDit

Sintaxis JSX

Renderizado Condicional

Renderizado de Listas

Manejo de Eventos

Props y Componentes

Hook useState

Hook useEffect

* Hook useContext

* Hook useRef

* Hook useReducer

React con TypeScript

React Router

Consola SpaDit

Componente Axios

Componente SpaditContexto

Componente Utilidades

Componente QueryForm

* Caso Práctico

APP base del Curso

https://svn.dit.aeat/repos/javadit/SPAD-CURS/develop/SPAD-CURS_APP

Descargar código de SPAD-CURS_APP en:

C:\AEAT\VSCode\workspaces\desa\proyectos\SPAD-CURS_APP

Abrir VsCode: C:\AEAT\VSCode\Code.lnk

En VsCode abrir la carpeta SPAD-CURS_APP

Twiki VsCode > https://www.aeat/twiki6sh/bin/view/Infra/VsCode

Twiki SpaDit > https://www.aeat/twiki6sh/bin/view/Infra/SpaDit

Sintaxis JSX

JSX es una extensión que permite escribir **HTML dentro de ficheros JavaScript**. Es una característica fundamental que facilita la creación de interfaces de usuario.

```
const nombre = "React";
return (
  <div className="container">
    <h1>Hola, {nombre}</h1>
    <p>Este es un ejemplo de JSX.</p>
  </div>
);
```

- Un componente debe retornar **un único elemento raíz**.
- Las expresiones de JavaScript se incluyen entre **llaves {}**.
- Los nombres de funciones, variables y atributos de los componentes siguen convenio **camelCase**.
- Los nombres de los componentes siguen convenio **PascalCase**.
- Algún atributo que cambia de nombre, por ejemplo, **className** en lugar de class.
- Se recomienda mantener la lógica fuera del JSX y definirla en funciones o variables.
- El uso de **fragmentos** (<>...</>) permite evitar elementos adicionales en el DOM.

Renderizado Condicional

El **renderizado condicional** permite mostrar u ocultar componentes o elementos del DOM en función de una condición lógica. Es esencial para construir interfaces dinámicas e interactivas.

```
{condición ? <ComponenteA /> : <ComponenteB />}
// Si la condición es verdadera, renderiza ComponenteA, si es falsa, renderiza ComponenteB

{condición && <Componente />}
// Si la condición es verdadera, renderiza Componente, si es falsa, no renderiza nada
```

Renderizado de Listas

El **renderizado de listas** permite mostrar colecciones de datos como elementos individuales. Comúnmente se utiliza el método **Array.map()** para crear estos elementos.

```
const ListaUsuarios = ({ usuarios }) => {
    return (
        <ul>
            {usuarios.map(usuario => (
                <li key={usuario.id}>{usuario.nombre}</li>
            ))}
        </ul>
    );
};
```

- Es importante usar **key** para proporcionar una identificación única y estable a cada elemento de la lista.
- No se recomienda usar el índice como key si la lista puede cambiar de orden, ya que puede generar problemas de renderizado.

Manejo de Eventos

El **manejo de eventos** permite que la interfaz responda a las acciones del usuario, como clicks, escritura, movimientos del ratón, etc. Es una parte fundamental para hacer las aplicaciones interactivas.

```
<button onClick={() => alert("¡Hola!")}>Haz clic</button>
// Al hacer click en el botón, se muestra una alerta

<input onChange={(e) => console.log(e.target.value)} />
// Al cambiar el valor del input, se muestra el valor en consola

<input onChange={handleChange} />
// Al cambiar el valor del input, se ejecuta la funcion handleChange

<button onClick={() => handleClick("Hola")}>Haz clic</button>
// Al hacer click, se ejecuta la funcion handleClick con parámetro
```

- Eventos comunes: onClick, onChange, onSubmit, onKeyDown, onKeyUp, onMouseOver.
- Los eventos pueden aceptar funciones anónimas o funciones predefinidas.

Props y Componentes

Las **props** (propiedades) permiten **pasar datos** desde un componente padre a sus hijos. Son fundamentales para crear componentes reutilizables y modulares.

```
const Saludo = ({ nombre, edad }) => (
    <div>
        <h1>Hola, {nombre}</h1>
        {edad && <p>Tienes {edad} años.</p>}
    </div>
);

<Saludo nombre="Juan" edad={25} />
```

- Las props son **inmutables** dentro del componente hijo.
- Se puede aplicar **destructuración** para acceder a las props de manera más clara.
- **Props por defecto**: Se pueden asignar valores predeterminados en la destructuración.
- **Props como funciones**: Permiten manejar eventos o lógica en componentes hijos.
- **Validar** las props con PropTypes o TypeScript para reducir errores.
- Para evitar el prop drilling (pasar props a múltiples niveles), considerar el uso de **useContext**.
- Mantener los componentes pequeños y con una única responsabilidad.

Hook useState

useState es un hook fundamental en React. Permite a un componente funcional mantener valores que cambian con el tiempo (estado). Cada vez que el estado cambia, el componente **se vuelve a renderizar** con el nuevo valor.

```
const [contador, setContador] = useState(0);

return (
    <div>
        <p>Contador: {contador}</p>
        <button onClick={() => setContador(contador + 1)}>Incrementar</button>
    </div>
);
```

- El estado puede ser de **cualquier tipo**: Un número, un string, un array, un objeto, etc.
- **Nunca modificar el estado directamente**. Siempre usa la función setEstado para actualizar el estado. Modificar directamente el estado puede causar problemas de consistencia y no actualizará el componente.
- La actualización del estado no ocurre de manera inmediata, por lo que si intentas acceder al estado justo después de actualizarlo, obtendrás el valor anterior.

Hook useEffect

useEffect permite ejecutar **efectos secundarios** en un componente, tales como peticiones HTTP, suscripciones a eventos, manipulación del DOM, etc. Estos efectos se pueden ejecutar al montar, actualizar o desmontar del componente, según sus dependencias.

```
useEffect(() => {
    // Efecto
    return () => {
        // Limpieza (opcional)
    };
}, [dependencias]);
```

- **Array de dependencias vacío [ ]**: El efecto se ejecuta solo una vez, al montar el componente.
- **Array de dependencias [x, y]**: El efecto se ejecuta cada vez que cambian las variables x o y.
- **Sin array de dependencias**: El efecto se ejecuta después de cada renderizado
- **Limpieza de recursos (return)**: Si el efecto utiliza recursos externos (como eventos, timers, o conexiones), es recomendable devolver una función de limpieza para evitar problemas.

Hook useContext

useContext es un hook que permite acceder a datos de un **contexto global** desde un componente. Es útil para **compartir datos entre múltiples componentes** sin pasar props manualmente.

```
// 1. Crear contexto
const DatosContexto = createContext();

// 2. Crear Proveedor de datos (en el componente padre o raíz)
const DatosContextoProvider = ({ children }) => {
  const datos = "ContenidoDeDatos"
  return (
    <DatosContexto.Provider value={datos}>
      {children}
    </DatosContexto.Provider>
  );
};

// 3. Anidar componentes en el Proveedor de datos (en el componente padre o raíz)
<DatosContextoProvider>
  <ComponenteA />
  <ComponenteB />
</DatosContextoProvider>

// 4. Consumir datos (en cualquier componente hijo)
const datos = useContext(DatosContexto);
```

- **createContext():** Crea el contexto.
- **Provider**: Proporciona el valor compartido a los componentes descendientes.
- **useContext(Contexto):** Permite a los componentes hijos acceder al valor del contexto.

Hook useRef

useRef permite crear una **referencia mutable** que persiste entre renderizados, **sin provocar una actualización del componente**.

```
const ref = useRef(valorInicial);
```

```
const inputRef = useRef(null);

<input ref={inputRef}/>
```

ref.current contiene el valor actual de la referencia y puede modificarse sin provocar un nuevo renderizado.

Se usa para:

- Acceder a **elementos del DOM** sin necesidad de document.querySelector.
- Almacenar valores que deben persistir entre renderizados sin causar re-renderizados.

Hook useReducer

El hook **useReducer** es una alternativa a useState para **manejar estados complejos**, donde hay múltiples acciones que afectan al estado.

```
// 1. función reducer: define cómo cambia el estado
const contadorReducer = (estado, accion) => {
    switch (accion.tipo) {
        case 'incrementar':
            return { contador: estado.contador + 1 };
        case 'reiniciar':
            return { contador: 0 };
        default:
            return estado;
    }
}

const ContadorConReducer = () => {
    // 2. useReducer recibe la función reducer y el estado inicial
    const [estado, dispatch] = useReducer(contadorReducer, { contador: 0 });

    return (
        <div>
            <p>Contador: {estado.contador}</p>
            <button onClick={() => dispatch({ tipo: 'incrementar' })}>
                Incrementar
            </button>
            <button onClick={() => dispatch({ tipo: 'reiniciar' })}>
                Reiniciar
            </button>
        </div>
    );
}
```

- **estado**: representa el estado actual.
- **dispatch(accion)**: función para ejecutar una acción y actualizar el estado.
- **reducer(estado, accion)**: función que recibe el estado actual y una acción y devuelve un nuevo estado.
- **estadoInicial**: estado por defecto cuando se monta el componente.

React con TypeScript

Usar TypeScript en una aplicación React es principalmente una ayuda en **tiempo de desarrollo**, ya que proporciona:

- **Tipado estático**: Previene errores al detectar problemas antes de ejecutar el código.
- **Autocompletado y mejor documentación**: Facilita la escritura de código con mejores sugerencias en los editores como VS Code.
- **Refactorización más segura**: Permite cambiar estructuras de datos y funciones con menor riesgo de errores.

En **tiempo de ejecución, solo existe JavaScript**, porque TypeScript se **transpila** a JavaScript antes de que el navegador lo ejecute.

```
interface Props {
  nombre: string;
  edad?: number;
}

const Saludo = ({ nombre, edad }: Props): JSX.Element => {
  return <p>Hola {nombre}, edad: {edad}</p>;
};
```

React con TypeScript

Estos son algunos de los tipos propios usados en el desarrollo de aplicaciones React:

```tsx
// Inputs y formularios
const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {};
const handleTextareaChange = (e: React.ChangeEvent<HTMLTextAreaElement>) => {};
const handleSelectChange = (e: React.ChangeEvent<HTMLSelectElement>) => {};
const handleFormSubmit = (e: React.FormEvent<HTMLFormElement>) => {};

// Teclado
const handleInputKeyDown = (e: React.KeyboardEvent<HTMLInputElement>) => {};
const handleDivKeyDown = (e: React.KeyboardEvent<HTMLDivElement>) => {};

// Ratón
const handleButtonClick = (e: React.MouseEvent<HTMLButtonElement>) => {};
const handleDivClick = (e: React.MouseEvent<HTMLDivElement>) => {};

// Otros eventos
const handleFocus = (e: React.FocusEvent<HTMLInputElement>) => {};
const handleDrag = (e: React.DragEvent<HTMLDivElement>) => {};

// Otros usos
interface Props { children: React.ReactNode };
const MyComponent1: React.FC<Props> = ({ children }) => <div>{children}</div>;
const MyComponent2 = ({ children }: Props): JSX.Element => <div>{children}</div>;
const inputRef = useRef<HTMLInputElement>(null);
```

React Router

La librería **react-router-dom** permite gestionar la navegación en React **sin recargar la página**.

```
function App() {
    return (
        <BrowserRouter>
            <nav>
                <Link to="/">Inicio</Link>
                <Link to="/perfil/123">Perfil</Link>
            </nav>
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/perfil/:id" element={<Perfil />} />
            </Routes>
        </BrowserRouter>
    );
}
```

- Se usa **<BrowserRouter>** para envolver la aplicación y **<Routes>** para agrupar las rutas.
- **<Link>** permite al usuario la navegación dentro del interfaz.
- **useNavigate** permite cambiar (y pasar datos) a otra ruta.
- **useParams** permite acceder a segmentos dinámicos de la URL.
- **useSearchParams** maneja parámetros de búsqueda en la URL.
- **useLocation** devuelve un objeto con detalles de la ruta actual, incluido datos pasados entre rutas.

Consola SpaDit

La Consola SpaDit muestra ejemplos de fragmentos JSX/TSX que hacen uso de React. La mayoría son componentes de la librería @dit/spad-rdit_library proporcionada por el DIT.

https://desa1.dit.aeat/spadit/SPAD-RDIT

Ejemplos de componentes de la librería @dit/spad-rdit_library:

https://desa1.dit.aeat/spadit/SPAD-RDIT/Ejemplos

Componentes visuales de la librería @dit/spad-rdit_library:

Accordion - ActionBar - Alert - Button - Card - Checkbox - ComboBox - DataGrid - DatePicker - Dropdown - InputText - List - LoadingOverlay - Modal - OutputData - ProgressBar - QueryForm - Select - Stepper - Switch - Tabs - TextArea

Componentes de infraestructura de la librería @dit/spad-rdit_library:

SpaditContexto - SpaditContextoProvider - Utilidades - axiosAPI - axiosInstance

Componentes AxiosInstance y AxiosAPI

Ambos permiten realizar peticiones GET o POST con paso de parámetros a endpoints del backend.

axiosInstance: permite añadir opciones de configuración en la petición (timeout, headers, ...) y obtener información adicional en la respuesta.

axiosAPI: versión simplificada con la configuración por defecto y devuelve directamente el "data" de la respuesta.

JSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploAxiosInstance

TSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploAxiosInstanceTSX

JSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploAxiosAPI

TSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploAxiosAPITSX

Ejemplo de Endpoint:

- https://desa1.dit.aeat/wlpl/SPAD-RDIT/EjemploRESTService

Utilidad de QuickType para obtener el tipado desde un JSON:

- https://desa1.dit.aeat/spadit/SPAD-RDIT/JsonGenerador

Backend. Serializar objetos Java a JSON

```
public class Persona {
  private String nombre;
  private int edad;

  // Constructor
  public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
  }

  // Getters y setters
}
```

Con la librería Gson (com.google.gson.Gson)

```
Persona persona = new Persona("Ana", 30);

Gson gson = new Gson();
String json = gson.toJson(persona);

//{"nombre":"Ana","edad":30}
```

Con el servicio ADHT_UTIL_Factory.getJsonSrv()

```
Persona persona = new Persona("Ana", 30);

JsonSrv jsonSrv = ADHT_UTIL_Factory.getJsonSrv();
String json = jsonSrv.toJson(persona);

//{"nombre":"Ana","edad":30}
```

Componente SpaditContexto

Permite obtener diferentes propiedades del contexto de ejecución de la SPA.

https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploSpaditContexto

Algunas de las propiedades que se pueden obtener del contexto:

- **Organismo**: AEAT
- **Entorno**: DESARROLLO
- **Plataforma**: INTRANET
- **Subplataforma**:
- **Sistema**: WLP
- **NIF**:
- **Usuario**: f0099xyz
- **NombreApellidos**: JUAN GARCIA SUAREZ
- **Dominio**: https://desa1.dit.aeat

Componente Utilidades

El método getUrl recibe como parámetro una ruta relativa y devuelve la ruta absoluta de JavaDit asociada al entorno y plataforma donde se este ejecutando la SPA.

https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploUtilidades

- Ejemplo cuando la SPA se ejecuta en Desarrollo - Intranet

  Utilidades.getUrl("/wlpl/SPAD-RDIT/EjemploRESTService")
  https://desa1.dit.aeat/wlpl/SPAD-RDIT/EjemploRESTService

- Ejemplo cuando la SPA se ejecuta en Local - Intranet

  Utilidades.getUrl("/wlpl/SPAD-RDIT/EjemploRESTService",false)
  https://desa1.dit.aeat/wlpl/SPAD-RDIT/EjemploRESTService

  Utilidades.getUrl("/wlpl/SPAD-RDIT/EjemploRESTService",true)
  https://intra.waslocal.gob.aeat:8443/wlpl/SPAD-RDIT/EjemploRESTService

- El segundo parámetro solo aplica en LOCAL

  Con el parámetro "**true**" devuelve la URL del dominio del servidor WLP LOCAL
  Con el parámetro "**false**" devuelve la URL del dominio de DESARROLLO

Componente Utilidades

En Local se puede definir el comportamiento global del método getUrl en toda la SPA, con la variable de entorno VITE_WLPL_LOCAL (ubicada en el fichero .env).

- VITE_WLPL_LOCAL = true (siempre devuelve las URLs del dominio del servidor WLP LOCAL)

  Utilidades.getUrl("/wlpl/SPAD-RDIT/EjemploRESTService")
  https://intra.waslocal.gob.aeat:8443/wlpl/SPAD-RDIT/EjemploRESTService

- VITE_WLPL_LOCAL = false (siempre devuelve las URLs del dominio de DESARROLLO)

  Utilidades.getUrl("/wlpl/SPAD-RDIT/EjemploRESTService")
  https://desa1.dit.aeat/wlpl/SPAD-RDIT/EjemploRESTService

- Si al método getUrl se le pasa el segundo parámetro (true o false), este parámetro prevalece sobre la variable de entorno VITE_WLPL_LOCAL

Componente QueryForm

Permite obtener el listado de una Query ya existente en JavaDit, sin necesidad de modificar el backend.

JSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploQueryForm

TSX > https://desa1.dit.aeat/spadit/SPAD-RDIT/EjemploQueryFormTSX

Ejemplo de Query:

- https://desa1.dit.aeat/wlpl/ADHT-AUDT/LogIntranetQuery

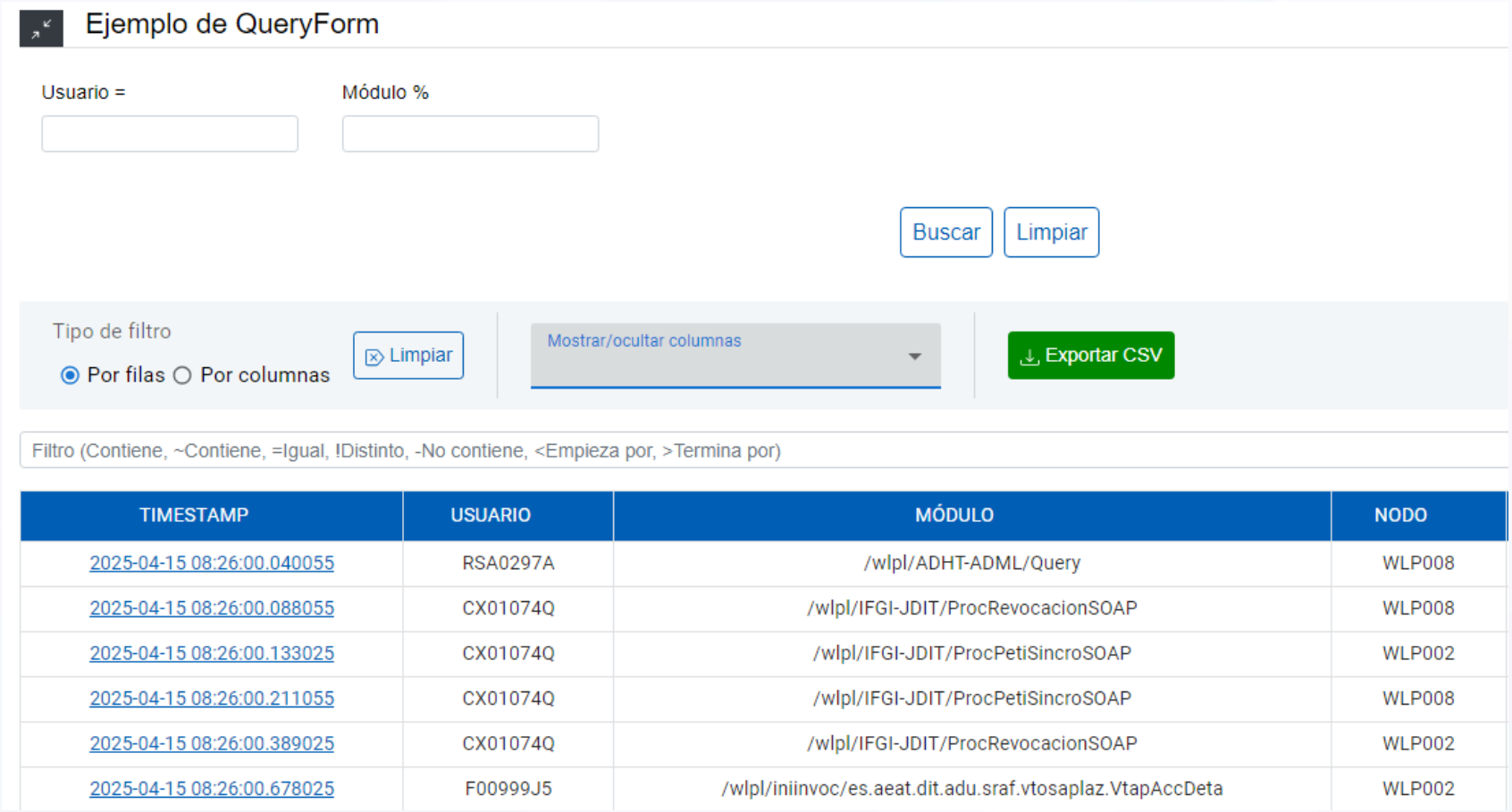
Ejemplo Endpoint REST primeros resultados:

- https://desa1.dit.aeat/wlpl/ADHT-AUDT/LogIntranetQuery?VEZ=BUSCAR1_JSON

Ejemplo Endpoint REST más resultados:

- https://desa1.dit.aeat/wlpl/ADHT-AUDT/LogIntranetQuery?VEZ=BUSCAR_JSON

Componente QueryForm



TIMESTAMP	USUARIO	MÓDULO	NODO
2025-04-15 08:26:00.040055	RSA0297A	/wlpl/ADHT-ADML/Query	WLP008
2025-04-15 08:26:00.088055	CX01074Q	/wlpl/IFGI-JDIT/ProcRevocacionSOAP	WLP008
2025-04-15 08:26:00.133025	CX01074Q	/wlpl/IFGI-JDIT/ProcPetiSincroSOAP	WLP002
2025-04-15 08:26:00.211055	CX01074Q	/wlpl/IFGI-JDIT/ProcPetiSincroSOAP	WLP008
2025-04-15 08:26:00.389025	CX01074Q	/wlpl/IFGI-JDIT/ProcRevocacionSOAP	WLP002
2025-04-15 08:26:00.678025	F00999J5	/wlpl/iniinvoc/es.aeat.dit.adu.sraf.vtosaplaz.VtapAccDeta	WLP002

Caso Práctico

- Crear formulario básico y **manejo de eventos** de botones
- Añadir estados para los campos del formulario con **useState**
- Implementar llamada HTTP con **axios**
- **useEffect** para cargar datos iniciales
- Uso de map para **renderizar listado** en una tabla
- **Renderizado condicional** cuando no hay resultados
- Añadir **validación** básica y mostrar **alerta de errores**
- Enfoque automático en campos con errores con **useRef**
- Implementar persistencia con **localStorage**
- Añadir navegación a otras vistas con **useNavigate**
- Añadir **TypeScript** (interfaces, tipos)

Ejemplo de formulario responsivo con Bootstrap (aeat.07.css)

```
<div class="container">
  <form>
    <div class="row">
      <div class="col-md-3 p-3">
        <label class="form-label">NIF</label>
        <input type="text" class="form-control form-control-sm" value=""/>
      </div>
      <div class="col-md-3 p-3">
        <label class="form-label">Grupo</label>
        <select class="form-control form-control-sm">
          <option value=""/>
          <option value="A1">A1</option>
          <option value="A2">A2</option>
          <option value="C1">C1</option>
          <option value="C2">C2</option>
        </select>
      </div>
      <div class="col-md-3 p-3">
        <label class="form-label">Nivel</label>
        <input type="number" class="form-control form-control-sm" value=""/>
      </div>
      <div class="col-md-3 p-3">
        <label class="form-label">Nombre</label>
        <input type="text" class="form-control form-control-sm" value=""/>
      </div>
    </div>

    <div class="d-flex flex-column flex-sm-row justify-content-center">
      <button type="button" class="btn btn-outline-primary m-3">Buscar</button>
      <button type="button" class="btn btn-outline-primary m-3">Limpiar</button>
      <button type="button" class="btn btn-outline-primary m-3">Nuevo Empleado</button>
    </div>
  </form>
</div>
```

Ejemplo de formulario responsivo con Bootstrap (aeat.07.css)

Clase	Función / Responsividad
.container	Centra y limita el ancho del contenido. Contenedor base del sistema grid en Bootstrap
.row	Activa el sistema de columnas de Bootstrap
.col-md-3	Ocupa 3 columnas en pantallas ≥768px (4 campos por fila); se apilan en <768px
.p-3	Agrega padding interno (espacio alrededor del campo)
.form-control	Estira el input/select al 100% del ancho de la columna
.form-control-sm	Aplica versión más pequeña del input/select (menos altura y padding)
.d-flex	Activa Flexbox en el contenedor de los botones
.flex-column	En vista móvil, los botones se apilan verticalmente
.flex-sm-row	Desde sm (≥576px), los botones se alinean horizontalmente
.justify-content-center	Centra horizontalmente los elementos dentro del d-flex
.m-3	Aplica margen en los 4 lados a cada botón, separándolos entre sí

https://getbootstrap.com/docs/4.3/layout/grid

https://getbootstrap.com/docs/4.3/components/forms

https://getbootstrap.com/docs/4.3/utilities/flex

Ejemplo de formulario responsivo con Bootstrap (aeat.07.css)

Alertas

Nivel inválido. Introduzca un nivel correcto (10-30).

Consulta de Empleados

NIF | Grupo | Nivel | Nombre

C1 | 7

Buscar | Limpiar | Nuevo Empleado

NIF	Grupo	Nivel	Nombre
59875389H	A2	20	María López A
58444864R	C2	17	Carmen Moraga
92021666T	A2	21	Pedro Fernández
89707885L	C1	20	Laura Rodríguez

Alertas

Nivel inválido. Introduzca un nivel correcto (10-30).

Consulta de Empleados

NIF

Grupo

C1

Nivel

7

Nombre

Buscar

Limpiar

Nuevo Empleado

NIF	Grupo	Nivel	Nombre
89707885L	C1	20	Laura Rodríguez

Buzon SpaDit

BUZON SPADIT E INFRAESTRUCTURAS EXTERNAS/AEAT/ES@AEAT

spadit@correo.aeat.es