



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TEPIC
ING. SISTEMAS COMPUTACIONALES
INTERFACES WEB



Tema 3. Desarrollo frontend.

Actividad Actividad 2: **CRUD en PWA** de Aplicación Base (**DEMO**) de clase.

GITHUB

[https://github.com/AmirPraiz/DEMO-
REACT-GADEV](https://github.com/AmirPraiz/DEMO-REACT-GADEV)

Presentado por:
Equipo 2 “GADEV”

Guizar Escobar Andrea Vianney
High Mendoza Jovan Enrique
Erik Moreno Duran
Praiz Cruz Amir Antonio
Romo Ruiz Paulina Michelle
Ruiz Murillo Denisse Monzerrat
Sánchez Leal Britney Bellanay

DOCENTE:

**Dr. Francisco Ibarra
Carlos.**

FECHA:

7 de Noviembre



Índice

Introducción.....	2
>Actividad.....	3
>Objetivos de la unidad.....	3
>Descripción.....	3
Contenido.....	5
A) Desarrollar FrontEnd y BackEnd de un CRUDs de los Formularios (paginas) web del proyecto base (demo) de clase con arquitectura SAP CDS y Vite PWA React + JavaScript.....	5
1.- Get Started with UI5 Web Components for React. (Primeros pasos con los componentes web UI5 para React.).....	6
2.- Create a Card Component. (Crea un componente de tarjeta.).....	9
3.- Integrate Charts and Conditional Rendering (Integración de gráficos y representación condicional).....	13
4.- Create an Analytical Dashboard via UI5 Web Components for React (Crea un panel de análisis mediante componentes web UI5 para React).....	28
5.- Add Routing to a UI5 Web Components for React Project (Aregar enrutamiento a un proyecto de componentes web UI5 para React).....	34
6.-Add Custom Styles and Components for UI5 Web Components for React (Agrega estilos y componentes personalizados para UI5 Web Components para React).....	37
Conclusión.....	40
Glosario.....	41
Bibliografía.....	43

Introducción.

El desarrollo de aplicaciones web modernas exige la integración de tecnologías robustas, eficientes y de fácil mantenimiento, capaces de responder ágilmente a las necesidades actuales de negocios e instituciones educativas.

El presente documento tiene como objetivo guiar a través de la construcción de una aplicación base de tipo CRUD (Crear, Leer, Actualizar y Eliminar) implementada como Progressive Web App (PWA), haciendo uso de un conjunto de herramientas y frameworks contemporáneos como SAP CDS, Vite, y React, además de integrar componentes de UI5 Web Components para React.

A lo largo de ese tutorial, se abordan aspectos esenciales del desarrollo frontend y backend, así como nociones clave sobre la infraestructura de interfaces, la gestión de estados con hooks y herramientas como Redux Toolkit, y la conexión con servicios API RESTful desarrollados en proyectos previos. El enfoque didáctico de la información permite tanto a estudiantes como a profesionales comprender los detalles técnicos y aplicar buenas prácticas en la creación de formularios, despliegue de dashboards analíticos, enrutamiento y personalización de estilos dentro de la plataforma. Además, de enfatizar la implementación de tablas interactivas, gráficos, manejo de eventos y el aprovechamiento de tecnologías en la nube tales como Azure y Azure DevOps para el despliegue, lo que brinda una visión integral del ciclo de vida de una aplicación empresarial en el entorno de la Ingeniería en Sistemas Computacionales.

>Actividad

T3-Actividad 2. CRUD en PWA de Aplicación Base (DEMO) de clase

>Objetivos de la unidad

- 3.1. Frameworks UI.
- 3.2. Arquitecturas.
- 3.3. Estilos y efectos.
- 3.4. Herramientas de maquetado.
- 3.5. Diseño basado en pruebas.
- 3.6. Interacción con el usuario.
- 3.7. Gestión de paquetes.

>Descripción

A) Desarrollar FrontEnd y BackEnd de un CRUDs de los Formularios (paginas) web del proyecto base (demo) de clase con arquitectura SAP CDS y Vite PWA React + JavaScript.

- Infraestructura de interfaces JavaScript.
- Componentes de FrontEnd de las páginas.
- Implementar en BackEnd infraestructuras de extracción de datos a través tecnologías de contenedores como colecciones, listas y arreglos de datos, así como interfaces, hooks, states, effects, dispatchers, reducers entre otros frameworks. Utilizar herramientas como Redux ToolKit.
- Implementar BackEnd de los enlaces (APIs) de la extracción de los datos y su despliegue en los componentes de pestañas (TabPages) en cada una de las paginas web (vistas del FrontEnd) a través de las colecciones y contenedores de almacenamiento de la información como resultado de los Request de las API's que fueron anteriormente desarrolladas en el RESTful del proyecto.

- Implementar tablas dentro de las pestañas y con su respectiva funcionalidad de CRUDs y APIs para la actualización de la información a través de los Formularios (Paginas).
- Despliegue y la aplicación completa.

Contenido.

3.1. Frameworks UI.

3.3. Estilos y efectos.

3.5. Diseño basado en pruebas.

3.6. Interacción con el usuario.

A) Desarrollar FrontEnd y BackEnd de un CRUDs de los Formularios (paginas)

web del proyecto base (demo) de clase con arquitectura SAP CDS y Vite PWA

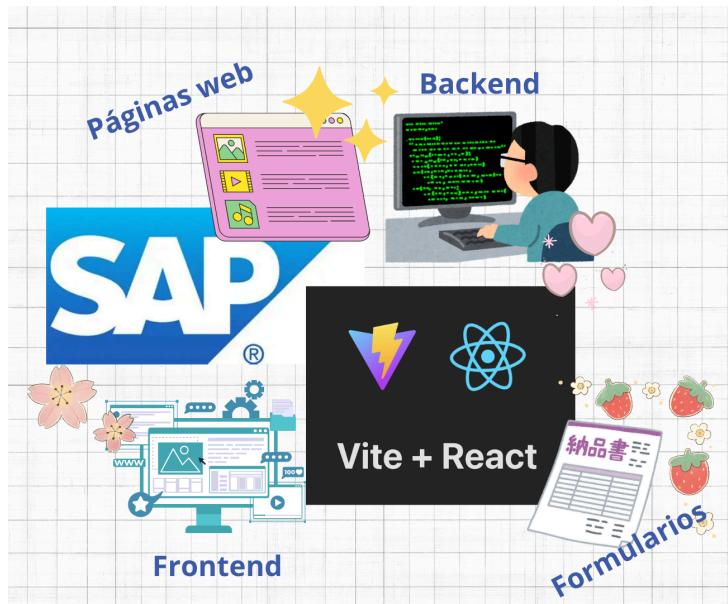
React + JavaScript.

El desarrollo integral de aplicaciones web modernas requiere la implementación coordinada de tecnologías tanto en el frontend como en el backend, para responder a las demandas de eficiencia, escalabilidad y experiencia de usuario en proyectos de ingeniería en sistemas computacionales. En este apartado se aborda el proceso de construcción de formularios y páginas web siguiendo la arquitectura base demo de clase, utilizando herramientas como SAP CDS, Vite y React, así como la integración de UI5 Web Components para enriquecer la interfaz visual.



Esta sección es una guía a través de los principios fundamentales para estructurar componentes de frontend, gestionar estados y efectos, y conectar estos elementos con servicios backend mediante APIs RESTful, empleando patrones y herramientas modernas como Redux Toolkit y contenedores de datos.

Besides, it details the importance of organizing the information flow, taking advantage of the arrays and collections, as well as structuring the components to make the updating easier and the efficient information management inside the app, preparing the land for the domain of better practices at the web-based systems designs.



1.- Get Started with UI5 Web Components for React. (Primeros pasos con los componentes web UI5 para React.)

Es necesario contar con

- **React and React-DOM (18.0.0 o alta)**
- **Node.js - LTS**

Y enseguida vamos a instalar esta dependencia:

```
npx degit UI5/webcomponents-react/templates/vite-ts#main my-app
```

A continuación dentro de nuestra carpeta SRC vamos a crear el archivo `MyApp.tsx` y pondremos el siguiente código dentro:

```
export function MyApp() {
  return <div>My root component</div>;
}
```

Dentro de nuestro archivo `App.tsx` vamos a modificarlo para que quede de la siguiente manera:

```
import { MyApp } from "./MyApp";  
  
function App() {  
  return (  
    <>  
    <MyApp />  
    </>  
  );  
}  
  
export default App;
```

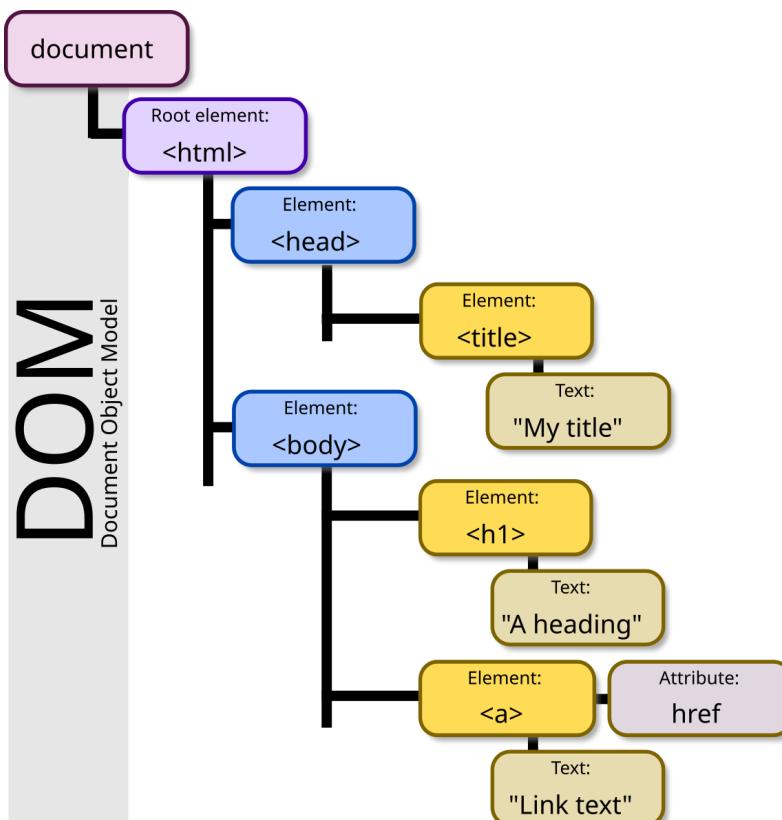
Realizamos el import de `MyApp` y lo “renderizamos”.

y para finalizar en nuestro archivo `main` lo modificamos para que quede así:

```
import '@ui5/webcomponents-react/dist/Assets.js';  
import { ThemeProvider } from '@ui5/webcomponents-react';  
import { StrictMode } from 'react';  
import { createRoot } from 'react-dom/client';  
import App from './App.tsx';  
import './index.css';  
  
createRoot(document.getElementById('root') as HTMLElement).render(  
  <StrictMode>  
    <ThemeProvider>  
      <App />  
    </ThemeProvider>  
  </StrictMode>,  
) ;
```

Nuestro código `main` se resumen en:

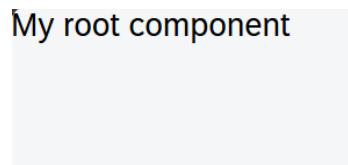
- **Assets.js**: Incluye recursos como archivos de traducción (CLDR), temas (theming), etc., de los paquetes necesarios.
- **ThemeProvider**: Entre otras cosas, este proveedor hace que tu aplicación reaccione a los cambios de tema y de idioma, e inyecta el CSS de los componentes utilizados.
- **StrictMode**: El componente `React.StrictMode` habilita comportamientos adicionales de desarrollo y advertencias para el árbol de componentes que envuelve. No es obligatorio, pero usarlo ayuda a encontrar errores comunes y problemas durante el desarrollo. Puedes encontrar más información aquí.
- **createRoot**: This function allows you to create the React root node for displaying components within the DOM in the browser (see React documentation). This node is typically added to the index.html file.



- **App**: Un componente de React.
- **index.css**: Archivo de CSS global.

Y para finalizar ejecutamos nuestro proyecto con el comando en consola:

```
npm run dev
```



My root component

2.- Create a Card Component. (Crea un componente de tarjeta.)

Para iniciar vamos a importar el componente Card dentro de nuestro archivo `MyApp.tsx`

```
import { Card } from "@ui5/webcomponents-react";
```

Usaremos el componente para hacer un “Mensaje”, el resultado del componente card:



Esta es el area de la carta

El código completo:

```
import { Card } from "@ui5/webcomponents-react";

export function MyApp() {
  return (
    <div>
      <Card>Esta es el area de la carta</Card>
    </div>
  );
}
```

El componente `card` puede tener más atributos, para que tenga más funcionalidad o para que sea vistoso.

Importaremos de ui5 2 componentes extras, text y cardheader, para hacer una demostración:

```
import { Card, CardHeader, Text } from "@ui5/webcomponents-react";
export function MyApp() {
  return (
    <div>
      <Card header={<CardHeader titleText="Carta" />}>
        <Text>Este es el area de la carta</Text>
      </Card>
    </div>
  );
}
```

Resultado:



Our card component, like any other, can have styles added using the style attribute. We will make our card 300px wide and the text spaced by 1 rem, which is written as var(--sapContent_Space_S).

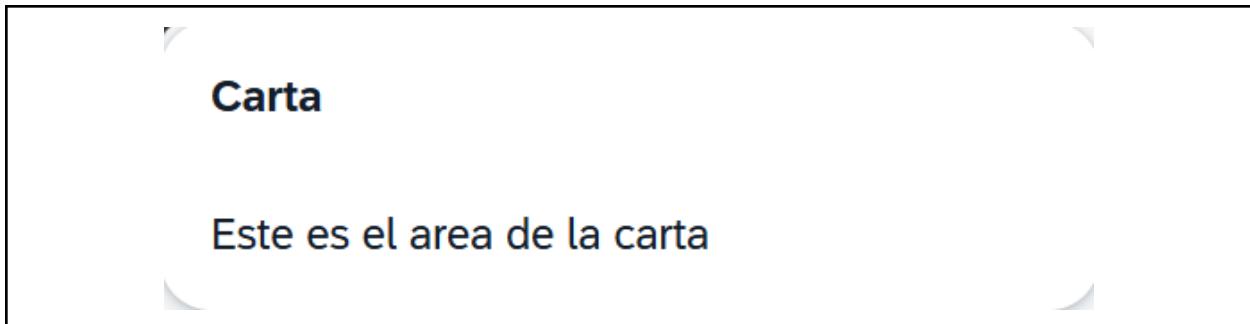
Codigo completo:

```
import { Card, CardHeader, Text } from "@ui5/webcomponents-react";

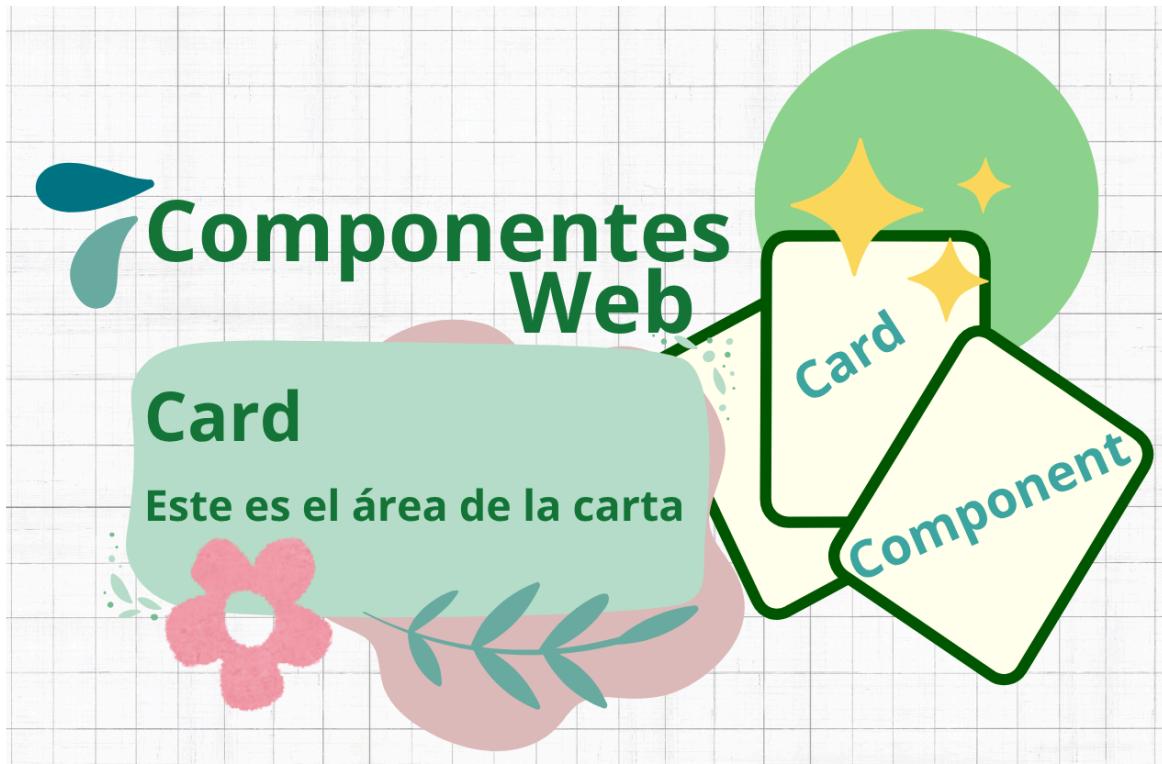
export function MyApp() {
  return (
    <div>
      <Card header={<CardHeader titleText="Carta" />} style={{ width:
        "300px" }}>
        <Text style={{ padding: "var(--sapContent_Space_S)" }}>
          Este es el area de la carta
        </Text>
      </Card>
    </div>
  );
}
```

```
) ;  
}
```

Resultado:



También podemos sumar funcionalidad a nuestro componente, agregaremos un Event handling.



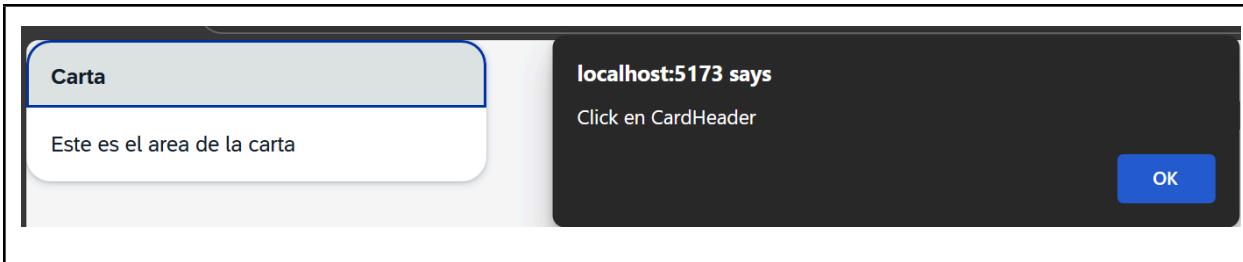
El componente `CardHeader` tiene la propiedad `interactive` y como su nombre lo indica es para hacerlo interactivo, y a su vez le añadiremos el atributo `onClick` para llamar la función `handleHeaderClick`, y esta función a su vez sera una simple ejecución de un `alert` para dar un mensaje.

Código completo:

```
import { Card, CardHeader, Text } from "@ui5/webcomponents-react";

export function MyApp() {
  const handleHeaderClick = () => {
    alert("Click en CardHeader");
  };
  return (
    <div>
      <Card
        header={
          <CardHeader
            titleText="Carta"
            interactive
            onClick={handleHeaderClick}
          />
        }
        style={{ width: "300px" }}
      >
        <Text style={{ padding: "var(--sapContent_Space_S)" }}>
          Este es el area de la carta
        </Text>
      </Card>
    </div>
  );
}
```

Resultado:



3.- Integrate Charts and Conditional Rendering (Integración de gráficos y representación condicional)

Para iniciar vamos a instalar nuevos módulos al proyecto:

```
npm install @ui5/webcomponents-react-charts
```

Y dentro de nuestro archivo `MyApp.tsx` añadiremos un nuevo import con los componentes de chart

```
import { BarChart, LineChart } from "@ui5/webcomponents-react-charts";
```

We are going to perform a price simulation over a period of months, so it is necessary to have data to perform this simulation.

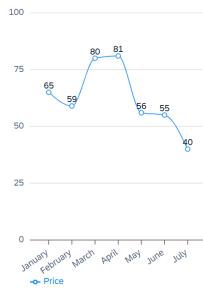
```
const dataset = [
  {
    month: "January",
    data: 65,
  },
  {
    month: "February",
    data: 59,
  },
  {
    month: "March",
    data: 80,
  },
  {
    month: "April",
    data: 81,
  },
  {
    month: "May",
    data: 56,
  },
  {
    month: "June",
    data: 55,
  },
  {
    month: "July",
  }
]
```

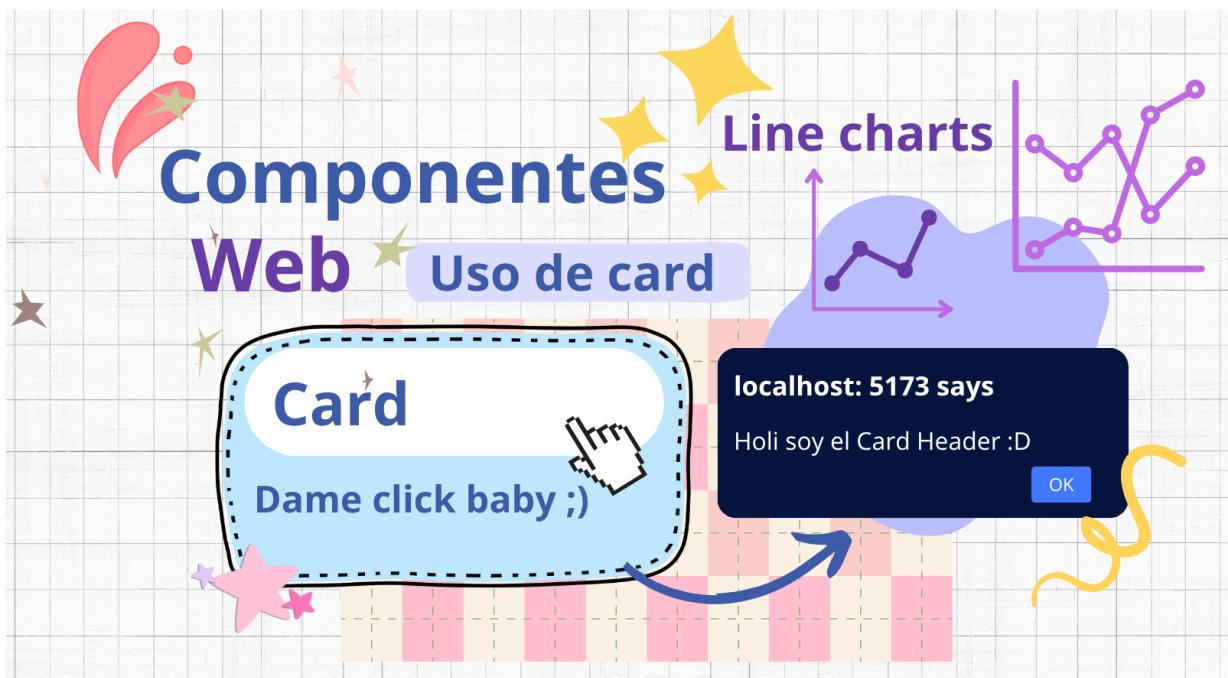
```
    data: 40,  
  },  
];
```

A continuación dentro de nuestra tarjeta vamos a crear el componente `LineChart`, los atributos serán `dimension` (que serán los meses), `measure` (que será el precio) y `dataset` será nuestro json de datos simulados:

```
<LineChart  
  dimensions={[{ accessor: "month" }]}  
  measures={[{ accessor: "data", label: "Price" }]}  
  dataset={dataset}  
/>
```

Resultado:

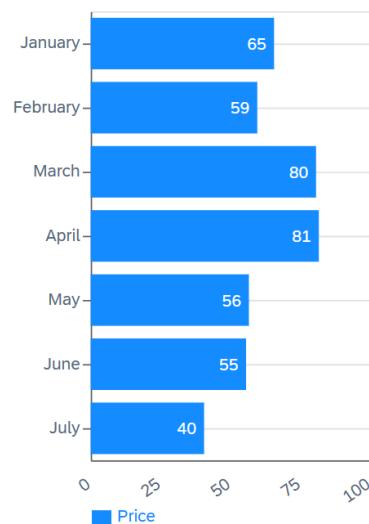




A continuación agregaremos el componente BarChart que contendrá los mismos atributos:

```
<BarChart  
    dimensions={[{ accessor: "month" }]}  
    measures={[{ accessor: "data", label: "Price" }]}  
    dataset={dataset}  
/>
```

Resultado:



Código completo:

```
import { Card, CardHeader, Text } from "@ui5/webcomponents-react";
import { BarChart, LineChart } from "@ui5/webcomponents-react-charts";

const dataset = [
  {
    month: "January",
    data: 65,
  },
  {
    month: "February",
    data: 59,
  },
  {
    month: "March",
    data: 80,
  },
  {
    month: "April",
    data: 81,
  },
  {
    month: "May",
    data: 56,
  },
  {
    month: "June",
    data: 55,
  },
  {
    month: "July",
    data: 40,
  },
]
```

```
month: "May",
  data: 56,
},
{
  month: "June",
  data: 55,
},
{
  month: "July",
  data: 40,
},
];
}

export function MyApp() {
  const handleHeaderClick = () => {
    alert("Click en CardHeader");
  };
  return (
    <div>
      <Card
        header={
          <CardHeader
            titleText="Carta"
            interactive
            onClick={handleHeaderClick}
          />
        }
        style={{ width: "300px" }}
      >
        <Text style={{ padding: "var(--sapContent_Space_S)" }}>
          Este es el area de la carta
        </Text>
        <LineChart
          dimensions={[{ accessor: "month" }]}
          measures={[{ accessor: "data", label: "Price" }]}
          dataset={dataset}
        />
        <BarChart
```

```
        dimensions={[{ accessor: "month" }]}  
        measures={[{ accessor: "data", label: "Price" }]}  
        dataset={dataset}  
    />  
  </Card>  
</div>  
) ;  
}
```

Ahora daremos la posibilidad al usuario de cambiar de gráfica.

Primero vamos a importar `useState` que nos serán muy útiles por el cambio de estado.

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

Y vamos a crear un hook para el cambio de gráfica.

```
const [toggleCharts, setToggleCharts] = useState("lineChart");
```

Ahora nuestra función `handleHeaderClick` tendrá una lógica distinta, la cual va a permitir cambiar de gráfica.

```
const handleHeaderClick = () => {  
  if (toggleCharts === "lineChart") {  
    setLoading(true);  
    setTimeout(() => {  
      setLoading(false);  
      setToggleCharts("barChart");  
    }, 2000);  
  } else {  
    setLoading(true);  
    setTimeout(() => {  
      setLoading(false);  
      setToggleCharts("lineChart");  
    }, 2000);  
  }  
};
```

```
    setLoading(false);
    setToggleCharts("lineChart");
}, 2000);
}
};
```

Y para manejar el renderizado lo haremos con un operador ternario:

```
{toggleCharts === "lineChart" ? (
  <LineChart
    dimensions={[{ accessor: "month" }]}
    measures={[{ accessor: "data", label: "Price" }]}
    dataset={dataset}
  />
) : (
  <BarChart
    dimensions={[{ accessor: "month" }]}
    measures={[{ accessor: "data", label: "Price" }]}
    dataset={dataset}
  />
)}
```

Código completo:

```
import lineChartIcon from "@ui5/webcomponents-icons/dist/line-chart.js";
import barChartIcon from "@ui5/webcomponents-icons/dist/vertical-bar-chart.js";
import { useState } from "react";
import { Card, CardHeader, Text, Icon } from "@ui5/webcomponents-react";
import { BarChart, LineChart } from "@ui5/webcomponents-react-charts";

const dataset = [
{
  month: "January",
  data: 65,
},
{
  month: "February",
  data: 70,
},
{
  month: "March",
  data: 75,
},
{
  month: "April",
  data: 80,
},
{
  month: "May",
  data: 85,
},
{
  month: "June",
  data: 90,
},
{
  month: "July",
  data: 95,
},
{
  month: "August",
  data: 100,
},
{
  month: "September",
  data: 105,
},
{
  month: "October",
  data: 110,
},
{
  month: "November",
  data: 115,
},
{
  month: "December",
  data: 120,
}];
```

```
month: "February",
  data: 59,
},
{
  month: "March",
  data: 80,
},
{
  month: "April",
  data: 81,
},
{
  month: "May",
  data: 56,
},
{
  month: "June",
  data: 55,
},
{
  month: "July",
  data: 40,
},
];
;

export function MyApp() {
  const [toggleCharts, setToggleCharts] = useState("lineChart");
  const [loading, setLoading] = useState(false);

  const handleHeaderClick = () => {
    if (toggleCharts === "lineChart") {
      setLoading(true);
      setTimeout(() => {
        setLoading(false);
        setToggleCharts("barChart");
      }, 2000);
    } else {
      setLoading(true);
    }
  };
}
```

```
setTimeout(() => {
    setLoading(false);
    setToggleCharts("lineChart");
}, 2000);
}

};

return (
<div>
<Card
header={
<CardHeader
titleText="Graficas"
interactive
avatar={
<Icon
name={
toggleCharts === "lineChart" ? lineChartIcon :
barChartIcon
}
/>
}
onClick={handleHeaderClick}
/>
}
style={{ width: "300px" }}
>
<Text style={{ padding: "var(--sapContent_Space_S)" }}>
    Graficas simuladas
</Text>
{toggleCharts === "lineChart" ? (
<LineChart
dimensions={{ [{} accessor: "month" ]}}
measures={{ [{} accessor: "data", label: "Price" ]}}
dataset={dataset}
/>
) : (
<BarChart
```

```
        dimensions={[{ accessor: "month" }]}  
        measures={[{ accessor: "data", label: "Price" }]}  
        dataset={dataset}  
    />  
 )}  
</Card>  
</div>  
);  
}
```

Ahora añadiremos la lógica de un nuevo hook.

```
const [loading, setLoading] = useState(false);
```

In our handleHeaderClick function, we have a setTimeout and in this case we use it to simulate how long a request to an API can take, so it is useful for the user to understand that the component is loading.

Antes de entrar a los setTimeout indicamos que nuestro hook loading es true, y una vez dentro lo hacemos false y en nuestros componentes LineChart y BarChart añadimos el atributo loading y le asignamos nuestro hook loading.

Codigo completo:

```
import lineChartIcon from "@ui5/webcomponents-icons/dist/line-chart.js";  
import barChartIcon from "@ui5/webcomponents-icons/dist/horizontal-bar-chart.js";  
import { useState } from "react";  
import { Card, CardHeader, Text, Icon } from "@ui5/webcomponents-react";  
import { BarChart, LineChart } from "@ui5/webcomponents-react-charts";  
  
const dataset = [  
  {  
    month: "January",  
    data: 65,  
  },  
  {  
    month: "February",  
    data: 70,  
  },  
  {  
    month: "March",  
    data: 75,  
  },  
  {  
    month: "April",  
    data: 80,  
  },  
  {  
    month: "May",  
    data: 85,  
  },  
  {  
    month: "June",  
    data: 90,  
  },  
  {  
    month: "July",  
    data: 95,  
  },  
  {  
    month: "August",  
    data: 100,  
  },  
  {  
    month: "September",  
    data: 95,  
  },  
  {  
    month: "October",  
    data: 90,  
  },  
  {  
    month: "November",  
    data: 85,  
  },  
  {  
    month: "December",  
    data: 80,  
  },  
];
```

```
        } ,
        {
            month: "February",
            data: 59,
        } ,
        {
            month: "March",
            data: 80,
        } ,
        {
            month: "April",
            data: 81,
        } ,
        {
            month: "May",
            data: 56,
        } ,
        {
            month: "June",
            data: 55,
        } ,
        {
            month: "July",
            data: 40,
        } ,
    ] ;

export function MyApp() {
    const [toggleCharts, setToggleCharts] = useState("lineChart");
    const [loading, setLoading] = useState(false);

    const handleHeaderClick = () => {
        if (toggleCharts === "lineChart") {
            setLoading(true);
            setTimeout(() => {
                setLoading(false);
                setToggleCharts("barChart");
            }, 2000);
        }
    }
}
```

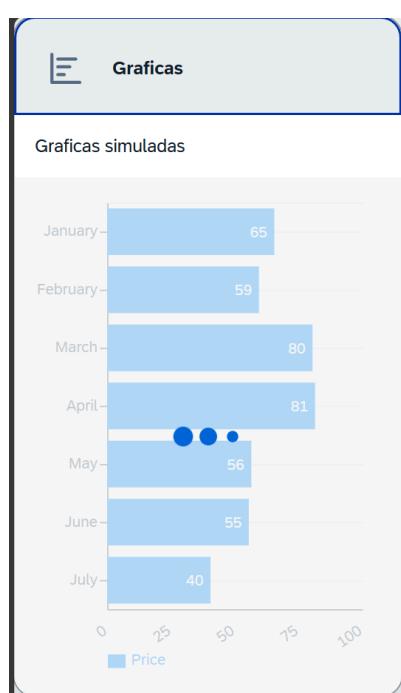
```
    } else {
      setLoading(true);
      setTimeout(() => {
        setLoading(false);
        setToggleCharts("lineChart");
      }, 2000);
    }
  };

  return (
    <div>
      <Card
        header={
          <CardHeader
            titleText="Graficas"
            interactive
            avatar={
              <Icon
                name={
                  toggleCharts === "lineChart" ? lineChartIcon :
barChartIcon
                }
              />
            }
          }
          onClick={handleHeaderClick}
        />
      }
      style={{ width: "300px" }}
    >
      <Text style={{ padding: "var(--sapContent_Space_S)" }}>
        Graficas simuladas
      </Text>
      {toggleCharts === "lineChart" ? (
        <LineChart
          dimensions={{ accessor: "month" }}}
          measures={{ accessor: "data", label: "Price" }}}
          dataset={dataset}
          loading={loading}
        </LineChart>
      ) : (
        <BarChart
          dimensions={{ accessor: "month" }}}
          measures={{ accessor: "data", label: "Price" }}}
          dataset={dataset}
          loading={loading}
        </BarChart>
      )}
    </div>
  );
}

export default App;
```

```
        />
      ) : (
    <BarChart
      dimensions={[{ accessor: "month" }]}
      measures={[{ accessor: "data", label: "Price" }]}
      dataset={dataset}
      loading={loading}
    />
  )
</Card>
</div>
);
}
```

Resultado:



Para finalizar, para hacerlo aún más dinámico, vamos a crear dos constantes que están anidadas al nuestro hook para determinar el valor, esto con el fin de dar un mensaje claro de cual grafica estamos viendo y a cual vamos a cambiar.

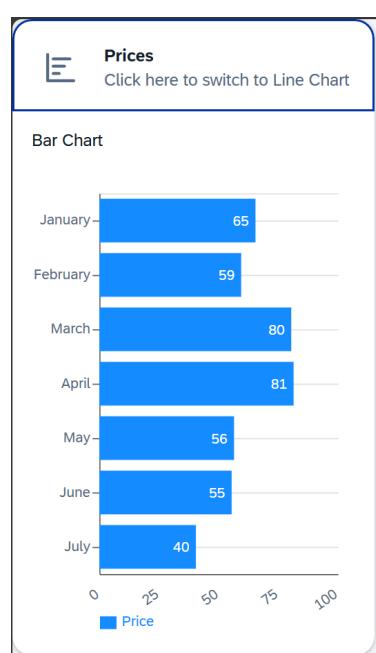
```
const contentTitle =  
  toggleCharts === "lineChart" ? "Line Chart" : "Bar Chart";  
const switchToChart =  
  toggleCharts === "lineChart" ? "Bar Chart" : "Line Chart";
```

Y el texto de nuestra Card como los iconos serán dinámicos.

```
<Card  
  header={  
    <CardHeader  
      titleText="Prices"  
      subtitleText={`Click here to switch to ${switchToChart}`}  
      interactive  
      avatar={  
        <Icon  
          name={  
            toggleCharts === "lineChart" ? lineChartIcon :  
            barChartIcon  
          }  
          accessibleName={contentTitle}  
        />  
      }  
      onClick={handleHeaderClick}  
    />  
  }  
  style={{ width: "300px" }}  
>  
<Text style={{ padding: "var(--sapContent_Space_S)" }}>  
  {contentTitle}  
</Text>  
{toggleCharts === "lineChart" ? (  
  <LineChart
```

```
        dimensions={[{ accessor: "month" }]}  
        measures={[{ accessor: "data", label: "Price" }]}  
        dataset={dataset}  
        loading={loading}  
    />  
  ) : (  
    <BarChart  
      dimensions={[{ accessor: "month" }]}  
      measures={[{ accessor: "data", label: "Price" }]}  
      dataset={dataset}  
      loading={loading}  
    />  
  )  
</Card>
```

Resultado:



4.- Create an Analytical Dashboard via UI5 Web Components for React (Crea un panel de análisis mediante componentes web UI5 para React)

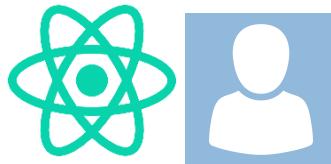


Para iniciar vamos a importar todo lo siguiente dentro de `MyApp.tsx`

```
import {
  Avatar,
  Card,
  CardHeader,
  Text,
  ShellBar,
  ShellBarItem,
  List,
  ListItemStandard,
  ListItemCustom,
  ProgressIndicator,
  FlexBox,
  FlexBoxJustifyContent,
  FlexBoxWrap,
  FlexBoxDirection,
  AnalyticalTable,
  Icon,
} from "@ui5/webcomponents-react";
```

Vamos iniciando creando un shellBar

Para iniciar vamos a importar dos iconos para hacerlo más vistoso.



Creamos una carpeta `assets` y colocamos ahí las imágenes y hacemos el `import` de ellas en nuestro archivo `MyApp.tsx`

```
import reactLogo from "./assets/reactLogo.png";
import profilePictureExample from "./assets/profilePictureExample.png";
```

y dentro de nuestro `return` (el final del código, la llamada de lo que se renderiza) añadiremos el componente `shellbar`

```
<ShellBar
  logo={<img src={reactLogo} alt="Company Logo" />}
```

```
profile={  
  <Avatar>  
    <img src={profilePictureExample} alt="User Avatar" />  
  </Avatar>  
}  
primaryTitle="My App"  
>  
  <ShellBarItem icon={activateIcon} text="Activate" />  
</ShellBar>
```

Ahora vamos a hacer una lista, para iniciar vamos a agregar un par de imports, para el manejo de los elementos gráficos.

```
import ValueState from "@ui5/webcomponents-base/dist/types/ValueState.js";  
import listIcon from "@ui5/webcomponents-icons/dist/list.js";
```

Let's create a new card below the one that was created.

```
<Card  
  header={  
    <CardHeader  
      titleText="Progress"  
      subtitleText="List"  
      avatar={<Icon name={listIcon} />}  
    />  
  }  
  style={{ width: "300px" }}  
>
```

y vamos a crear dos componentes (dentro de card) ListItemStandard, con los atributos additionalText y additionalTextState (que es para indicar con un color el estado del texto)

```
<ListItemStandard  
  additionalText="finished"
```

```
        additionalTextState={ValueState.Positive}  
>  
    Activity 1  
</ListItemStandard>  
<ListItemStandard  
    additionalText="failed"  
    additionalTextState={ValueState.Negative}  
>  
    Activity 2  
</ListItemStandard>
```

y también agregaremos dos componentes más que serán `ListItemCustom` que sirve para crear elementos personalizados dentro de una lista y te permite *poner cualquier contenido React* dentro del ítem, sin limitarte a un diseño predefinido.

```
<ListItemCustom>  
    <FlexBox  
        direction={FlexBoxDirection.Column}  
        fitContainer  
        style={{ paddingBlock: "var(--sapContent_Space_S)" }}  
>  
        <FlexBox  
            justifyContent={FlexBoxJustifyContent.SpaceBetween}>  
                <Text style={{ fontSize: "var(--sapFontLargeSize)" }}>  
                    Activity 3  
                </Text>  
                <Text style={{ color: "var(--sapCriticalTextColor)" }}>  
                    in progress  
                </Text>  
            </FlexBox>  
            <ProgressIndicator  
                value={89}  
                valueState={ValueState.Positive}  
                style={{ marginBlockStart: "0.5rem" }}  
>  
        </FlexBox>  
</ListItemCustom>
```

```
<ListItemCustom>
    <ProgressIndicator value={5} valueState={ValueState.Negative}>
/>
</ListItemCustom>
```

A continuación vamos a añadir un componente llamado AnalyticalTable, para iniciar vamos a crear datos de prueba:

```
const tableData = new Array(500).fill(null).map(_, index) => {
  return {
    name: `name${index}`,
    age: Math.floor(Math.random() * 100),
    friend: {
      name: `friend.Name${index}`,
      age: Math.floor(Math.random() * 100),
    },
  };
};

const tableColumns = [
  {
    Header: "Name",
    accessor: "name", // String-based value accessors!
  },
  {
    Header: "Age",
    accessor: "age",
  },
  {
    Header: "Friend Name",
    accessor: "friend.name",
  },
  {
    Header: "Friend Age",
    accessor: "friend.age",
  },
];
```

Para después crear un card con nuestra AnalyticalTable

```
<Card
  header={
    <CardHeader
      titleText="AnalyticalTable"
      avatar={<Icon name={tableViewIcon} />}
    />
  }
  style={{ maxWidth: "900px" }}
>
<AnalyticalTable
  data={tableData}
  columns={tableColumns}
  visibleRows={5}
/>
</Card>
```

AnalyticalTable contendrá los siguientes atributos

- Data = con los datos creados en la función TableData
- columns = con los datos del JSON tableColumns
- visibleRows = para limitar los datos visibles.

Resultado:

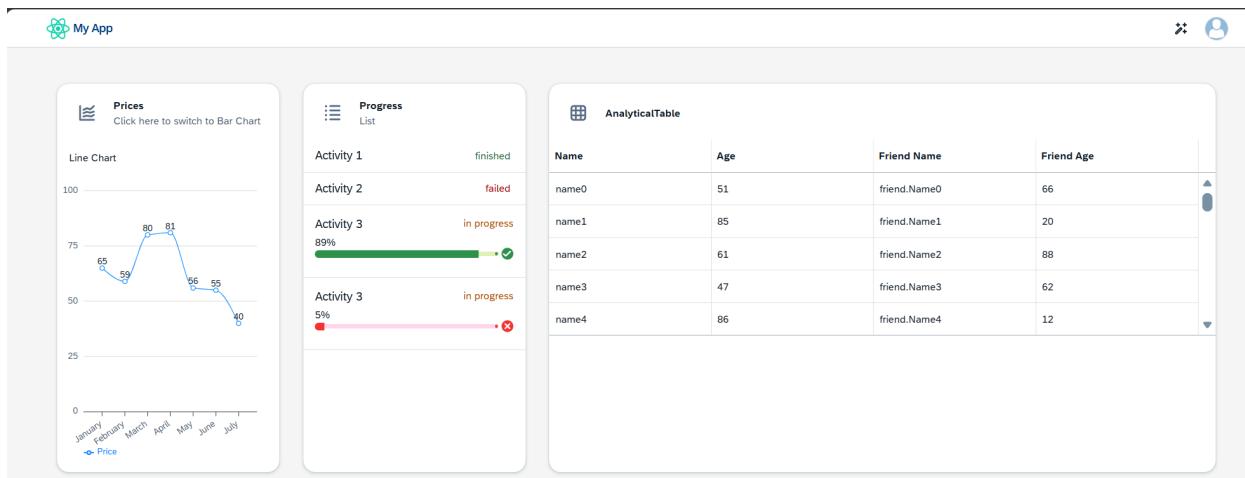
Name	Age	Friend Name	Friend Age
name0	53	friend.Name0	23
name1	19	friend.Name1	16
name2	76	friend.Name2	61
name3	20	friend.Name3	72
name4	61	friend.Name4	28

Para continuar añadiremos layouts para que nuestra vista se parezca más a un dashboard clásico. En cada card meteremos un atributo Style para moldear cada componente mas a nuestro agrado, como ejemplo:

```
<Card
    header={
        <CardHeader
            titleText="AnalyticalTable"
            avatar={<Icon name={tableViewIcon} />}
        />
    }
    style={{{
        maxWidth: "900px",
        margin: "var(--sapContent_Margin_Small)",
    }}}>
```

Aquí nuestro style hace que nuestra card tenga una anchura máxima de 900px y que el margin aplica el margen pequeño definido en el tema de SAP para espacios entre elemento (0.5 rem aproximadamente)

Resultado:



5.- Add Routing to a UI5 Web Components for React Project (Aregar enrutamiento a un proyecto de componentes web UI5 para React)

Primero vamos a instalar un nuevo componente a nuestro proyecto, el cual es:

```
npm install react-router-dom
```

En seguida vamos a crear dentro de src un nuevo archivo llamado Detail.tsx con este código:

```
import { Title } from "@ui5/webcomponents-react";

export function Detail() {
  return <Title>Hello World!</Title>;
}
```

Y dentro de App.tsx vamos a añadir un nuevo import y vamos a “envolver” la llamada de MyApp con ese nuevo import

```
import { MyApp } from "./MyApp";
import { HashRouter } from "react-router-dom"; // <--- Nuevo import agregado

function App() {
  return (
    <HashRouter> {/* <--- Componente HashRouter envolviendo MyApp */}
      <MyApp />
    </HashRouter>
  );
}

export default App;
```

We're going to create a new TSX file called Home.tsx that will contain everything from myApp, except for the ShellBar we created earlier.

Inside myApp, we'll import the following components and views:

```
import { Routes, Route, Navigate } from "react-router-dom";
import { Home } from "./Home";
import { Detail } from "./Detail";
```

Así podremos usar Routes de react dom como también podremos realizar la navegación a las vistas que creamos.

El código completo de MyApp queda de la siguiente manera:

```
import activateIcon from "@ui5/webcomponents-icons/dist/activate.js";
import { Avatar, ShellBar, ShellBarItem } from "@ui5/webcomponents-react";
import profilePictureExample from "./assets/profilePictureExample.png";
import reactLogo from "./assets/reactLogo.png";
import { Routes, Route, Navigate } from "react-router-dom";
import { Home } from "./Home";
import { Detail } from "./Detail";

export function MyApp() {
  return (
    <>
      <ShellBar
        logo={<img src={reactLogo} alt="Company Logo" />}
        profile={
          <Avatar>
            <img src={profilePictureExample} alt="User Avatar" />
          </Avatar>
        }
        primaryTitle="My App"
      >
        <ShellBarItem icon={activateIcon} text="Activate" />
      </ShellBar>
      <Routes>
        <Route path="/home" element={<Home />} />
        <Route path="/detail" element={<Detail />} />
        <Route path="*" element={<Navigate replace to="/home" />} />
      </Routes>
    </>
  ) ;
}
```

Dentro de Home, en específico en la segunda Card vamos a añadir un evento onClick llamando a la función handleProgressHeaderClick que esta a su vez tendrá el siguiente código:

```
const handleProgressHeaderClick = () => {
  navigate("/detail");
};
```

Así podremos hacer la navegación a la vista detail solo haciendo un click sobre el headear de la segunda card.

```
import { useNavigate } from "react-router-dom";
const navigate = useNavigate();
```

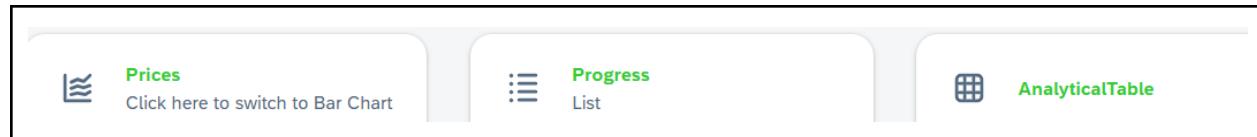
No olvidar hacer el import de useNavigate como la declaración de este dentro de la función principal de Home.

6.-Add Custom Styles and Components for UI5 Web Components for React (Agrega estilos y componentes personalizados para UI5 Web Components para React)

Dentro de nuestro archivo index.css podemos cambiar el estilo de nuestros componentes, por ejemplo:

```
* {
  --sapTile_TitleTextColor: limegreen;
}
```

Esto hace que todos nuestros componentes con la propiedad --sapTile_TitleTextColor se tornen de un color verde.



Pero para dar un mejor ejemplo vamos a crear el archivo `MyCustomElement.tsx` y `MyCustomElement.module.css` con el siguiente código:

MyCustomElement.module.css

```
.container {  
    background-color: var(--sapBackgroundColor);  
    font-family: var(--sapFontFamily);  
    height: 50px;  
    display: flex;  
    justify-content: center;  
    align-items: center;  
}
```

MyCustomElement.tsx

```
import { ThemingParameters } from '@ui5/webcomponents-react-base';  
import classes from './MyCustomElement.module.css';  
  
export const MyCustomElement = () => {  
    return (  
        <div className={classes.container}>  
            <span  
                style={{  
                    color: ThemingParameters.sapNegativeColor,  
                    fontSize: ThemingParameters.sapFontHeader1Size,  
                }}>  
                >  
                Mi elemento personalizado  
            </span>  
        </div>  
    );  
};
```

Debemos de instalar la siguiente librería

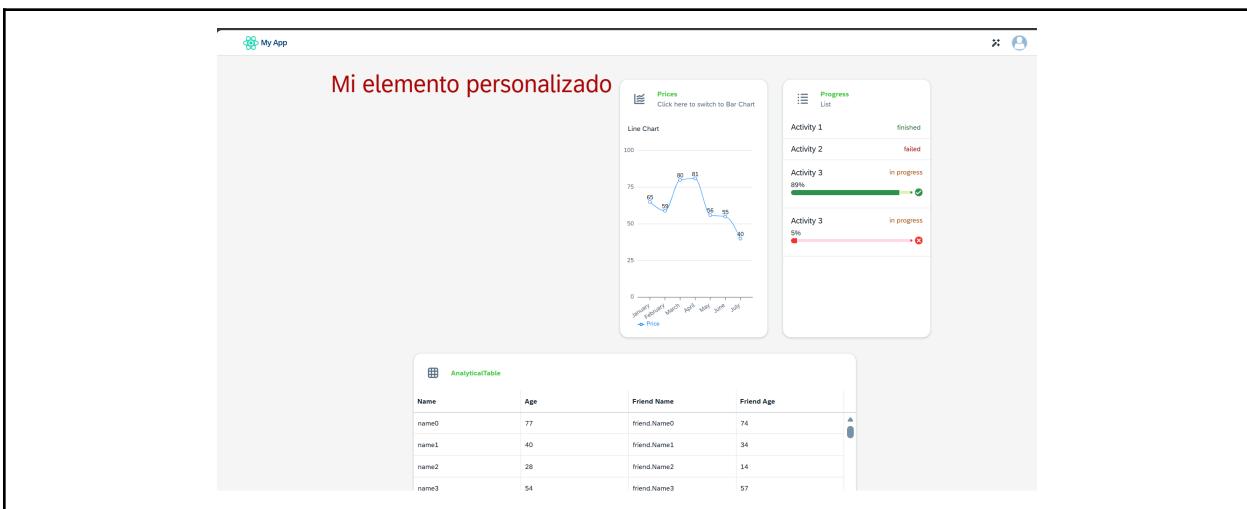
```
npm install @ui5/webcomponents-react-base
```

Resultado:

Interfaces Web. -Tema 3. Desarrollo frontend

Dr. Francisco Ibarra Carlos.

Instituto Tecnológico de Tepic || Ingeniería en Sistemas Computacionales



Conclusión.

Para finalizar, el desarrollo de CRUD en aplicaciones PWA mediante arquitecturas SAP CDS y frameworks modernos como React y Vite representa una estrategia sólida para el diseño de soluciones web escalables, interactivas y alineadas a los estándares de la industria actual. Este documento expuso paso a paso cómo estructurar tanto el frontend como el backend, integrando la gestión de estados, estilos personalizados, dashboards analíticos y la visualización de datos a través de componentes reutilizables. Al mismo tiempo, resalta la importancia de vincular estas aplicaciones a servicios en la nube, facilitando su despliegue y administración continua. Esta aproximación fomenta el aprendizaje activo, la experimentación y la capacidad de adaptación de los desarrolladores a entornos cambiantes, preparándolos para enfrentar retos reales de la ingeniería de software.

Finalmente, el material proporciona un recurso fundamental no solo para dominar una tecnología específica, sino para adquirir una perspectiva integral sobre buenas prácticas y metodologías en proyectos web empresariales, contribuyendo al crecimiento profesional y académico de los estudiantes y futuros ingenieros en sistemas computacionales.

Glosario.

API RESTful

Es un estilo de arquitectura para diseñar servicios web que usan los métodos HTTP (GET, POST, PUT, DELETE) para operar recursos en formato JSON o XML, facilitando la comunicación entre frontend y backend.

DOM

DOM significa Document Object Model (Modelo de Objetos del Documento), una interfaz de programación que representa documentos HTML y XML como una estructura de árbol de objetos, permitiendo que lenguajes como JavaScript manipulen la estructura, el estilo y el contenido de una página web.

Hook

En React, es una función especial que permite “enganchar” características de React como el estado (state) y el ciclo de vida en componentes funcionales, evitando la necesidad de clases

PWA

Progressive Web App, aplicación web que usa tecnologías modernas para ofrecer una experiencia similar a la de una app nativa, con funcionalidades como trabajo offline, notificaciones push y acceso desde el escritorio o móvil.

SAP CDS

Is a technology and framework for defining and consuming data models in the database layer, rather than the application layer. It's a data-centric approach that uses an extension of SQL to build semantically rich data models that are optimized for performance.

Redux Toolkit

Redux Toolkit (RTK) es la biblioteca oficial para simplificar la gestión del estado con Redux. Su objetivo es reducir el código repetitivo (boilerplate) y hacer que el uso de Redux sea más fácil y eficiente.

Renderizar

Es el proceso de generar una imagen final o una escena visible a partir de datos o modelos digitales

UI5

UI5, o SAPUI5, es un framework de JavaScript que SAP desarrolló para crear aplicaciones web empresariales responsivas y de interfaz de usuario atractivas, utilizando tecnologías web como HTML5, CSS y JavaScript.

Bibliografía.

de Alejandro, V. M. C. (s/f). *Qué son los hooks y cómo utilizarlos en tu proyecto con React*. Paradigmadigital.com. Recuperado el 8 de noviembre de 2025, de <https://www.paradigmadigital.com/dev/hooks-como-utilizarlos-react/>

Strategic Platform. (2024, 1 de octubre). SAPUI5: concepto, funciones, ventajas, tutoriales y tipos de uso. Recuperado de <https://strategicplatform.com/articulos/sapui5> Estrategia y Tecnología

MDN Web Docs. (2024, 17 de diciembre). DOM – Glosario de MDN Web Docs. Recuperado de <https://developer.mozilla.org/es/docs/Glossary/DOM> MDN Web Docs

SAP SE. (s. f.). SAPUI5 SDK – Demo Kit. Recuperado de <https://sapui5.hana.ondemand.com/> SAPUI5 SDK+1

Seobility Wiki. (s. f.). *¿Qué es Renderizar o Rendering?*. Recuperado de <https://www.seobility.net/es/wiki/Renderizar> Seobility

Vidal, M. (2022). *¿Qué son las Progressive Web Apps? ¿Por qué son tan importantes?* *Thinking for Innovation*. <https://www.iebschool.com/hub/progressive-web-apps-analitica-usabilidad/>

What is REST? (2018, mayo 29). REST API Tutorial. <https://restfulapi.net/>

Barragán, A. (2024a, septiembre 12). Redux Toolkit: Simplifica la gestión del estado.

Openwebinars.net.

<https://openwebinars.net/blog/redux-toolkit-simplifica-gestion-estado/>