Módulo 4
Desarrollo de interfaces interactivas con React

Elementos Fundamentales de React JS

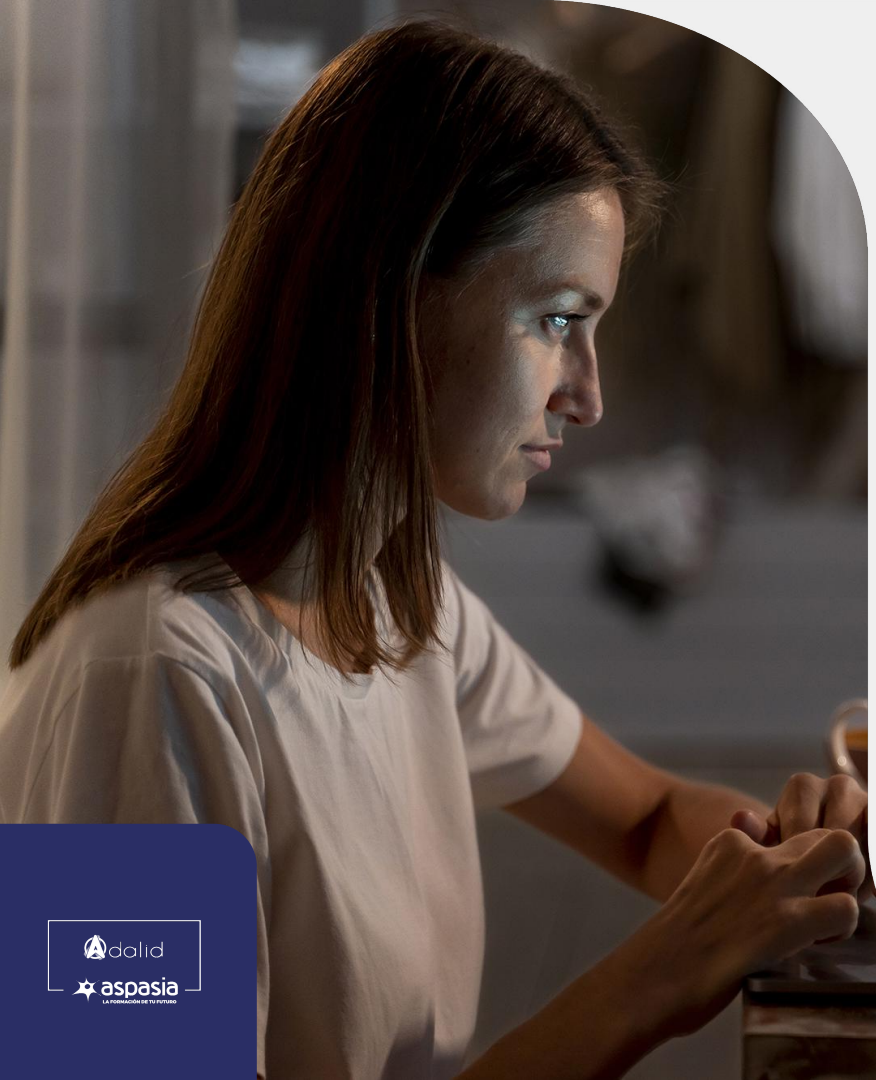
Módulo 4

AE 2

OBJETIVOS

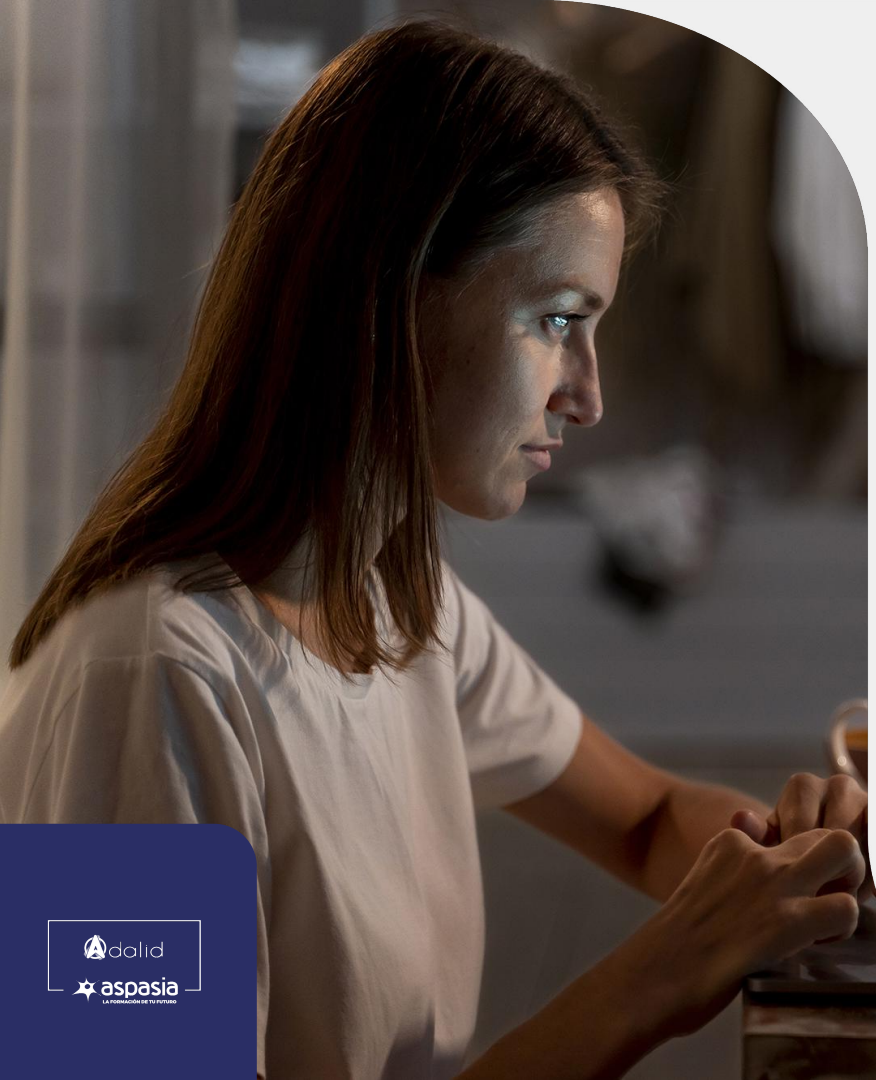
Domina los fundamentos de ReactJS: Aprende JSX, props, hooks, y cómo construir componentes reutilizables para interfaces dinámicas, eficaces y modulares.

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO



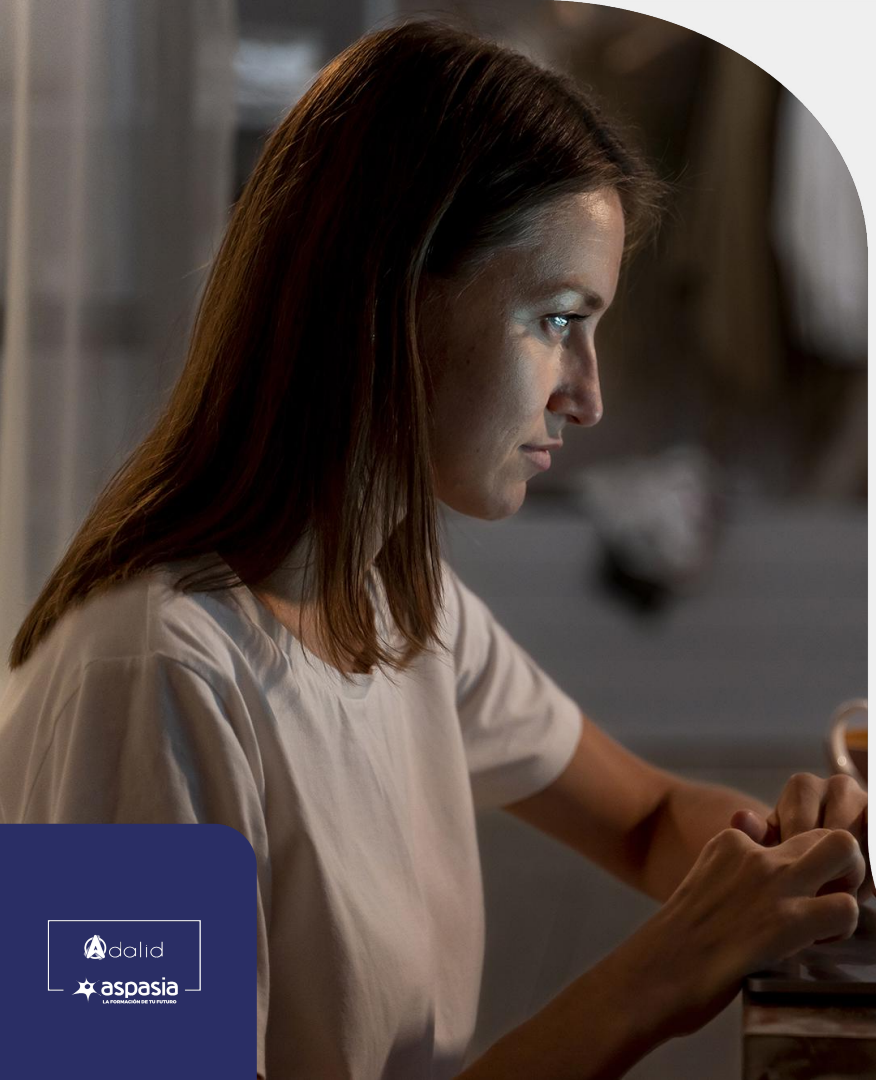
¿QUÉ VAMOS A VER?

- Elementos Fundamentales de ReactJS.
- Introducción a JSX.
- Pensando en ReactJs.
- Componentes en ReactJs.
- Composición versus herencia.
- Paso de datos con Props.
- Listas y keys.
- Formularios.
- El ciclo de vida ReactJs.
- Desplegando datos en la interfaz (Renderización).
- Uso de extensión browser “React Developer Tools”.



¿QUÉ VAMOS A VER?

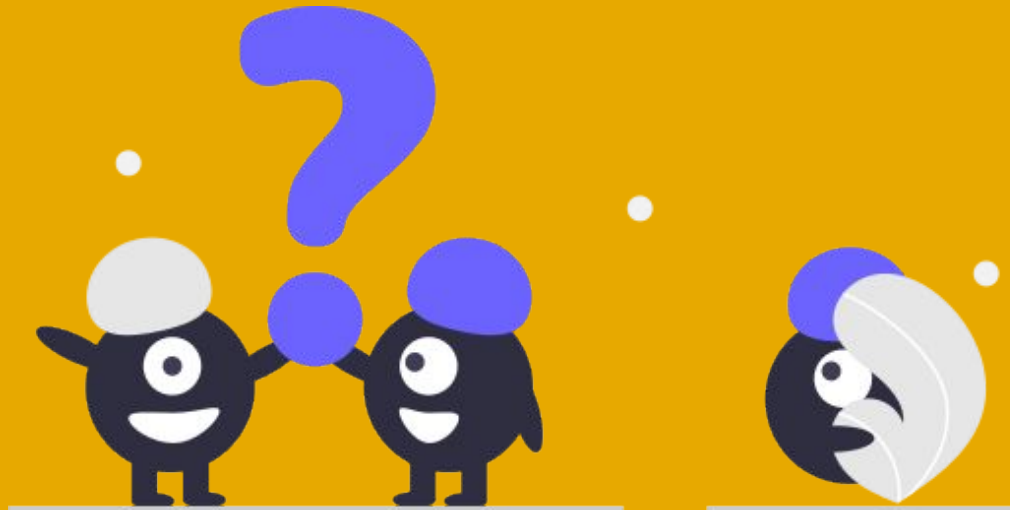
- Hojas de estilo en ReactJs.
- Introducción a Hooks.
- Introducción a uso de API’s.
- JSX, la Simplificación de ReactJS.
- Porqué usarJSX.
- Insertando expresiones en JSX.
- Notación y palabras reservadas.
- JSX también es una expresión.
- Especificando atributos con JSX.
- Especificando hijos con JSX.
- Previniendo ataques de inyección con JSX.
- Representación de objetos en JSX.

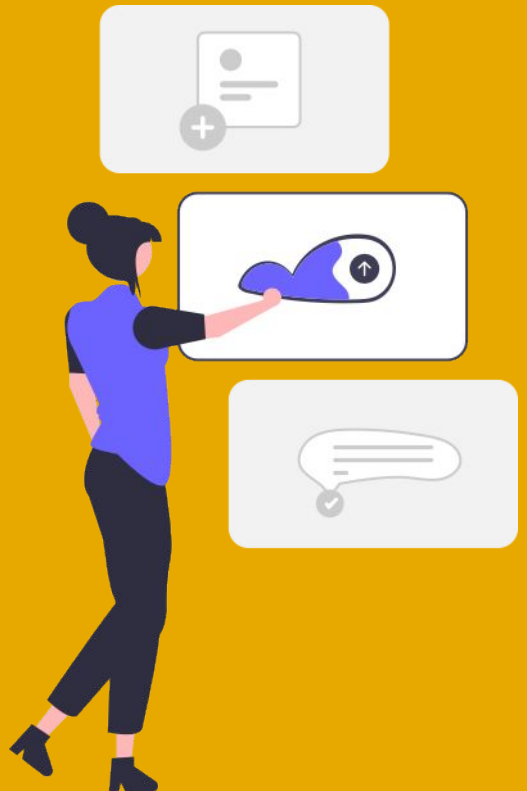


¿QUÉ VAMOS A VER?

- El rol de Babel en la conversión de objetos.
- Comentarios en JSX.
- Implementar componentes reutilizables.
- Describir los aspectos fundamentales y beneficios de la componentización en el desarrollo de aplicaciones front-end.
- Reconocerlos elementos y sintaxis requerida para la especificación de un componente.
- Utilizar componentes previamente construidos utilizando sus propiedades para su personalización

¿Que puede hacer React?





Elementos Fundamentales de ReactJS

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Elementos Fundamentales de ReactJS

Para entender cómo funciona React vamos a desarrollar una aplicación **desde cero**, cubriendo los conceptos fundamentales mientras avanzamos en la construcción de nuestro proyecto.

Alista tu terminal 😁

Configuración Inicial del Proyecto

Para comenzar a trabajar con React, configuraremos un proyecto utilizando **Vite**. Esto nos permitirá crear un entorno moderno y rápido para desarrollar.

Crear un proyecto React con Vite:

```
npm create vite@latest my-todo-app -- --template react
cd my-todo-app
npm install
npm run dev
```

npm run dev: Comando para iniciar el servidor de desarrollo.

```
my-todo-app/
├── node_modules/
├── public/
│   └── vite.svg
├── src/
│   ├── App.css
│   ├── App.jsx
│   ├── index.css
│   ├── main.jsx
│   └── assets/
│       └── react.svg
├── .gitignore
├── eslint.config.js
├── index.html
├── package-lock.json
├── README.md
└── vite.config.js
```

src/App.jsx: Archivo principal donde se define el componente raíz de la aplicación.

Vite.config.js: Configuración de Vite.

Introducción a JSX

JSX (JavaScript XML) es una extensión de JavaScript que permite escribir código HTML dentro de un archivo JavaScript. JSX es más **legible** y se convierte en JavaScript utilizando Babel.

Edita App.jsx:

```
export default function App() {
  const title = 'Welcome to My Todo App';
  return (
    <div>
      <h1>{title}</h1>
    </div>
  );
}
```

- **{title}:** Las llaves permiten insertar expresiones de JavaScript.
- JSX es más legible y elimina la necesidad de usar funciones como React.createElement.

Pensando en ReactJS

ReactJS se basa en la **división de interfaces** en **componentes reutilizables**. Identificamos las partes de la UI y las implementamos como pequeños bloques independientes.

Crea un componente Header.jsx:

```
export default function Header() {
  return (
    <header>
      <h1>My Todo App</h1>
    </header>
  );
}
```

Cada sección de la UI es un componente independiente.

Agregalo en App.jsx:

```
import Header from './Header';

export default function App() {
  return (
    <div>
      <Header />
    </div>
  );
}
```

Los componentes promueven la reutilización de código y la separación de responsabilidades.

Componentes en ReactJS

Los componentes son funciones que devuelven un árbol de elementos JSX. Pueden ser **funcionales** (como hemos visto) o basados en clases (menos comunes hoy).

Crea un componente TaskList.jsx:

```
export default function TaskList() {
  return (
    <ul>
      <li>Learn React</li>
      <li>Build a project</li>
      <li>Explore Hooks</li>
    </ul>
  );
}
```

Modifica App.jsx agregando TaskList:

```
import Header from './Header';
import TaskList from './TaskList';

export default function App() {
  return (
    <div>
      <Header />
      <TaskList />
    </div>
  );
}
```

Paso de Datos con Props

Las props permiten pasar datos a los componentes. Son inmutables y se utilizan para personalizar componentes.

Crea un componente Task.jsx:

```jsx
export default function Task({ task }) {
  return <li>{task}</li>;
}
```

Modifica TaskList.jsx:

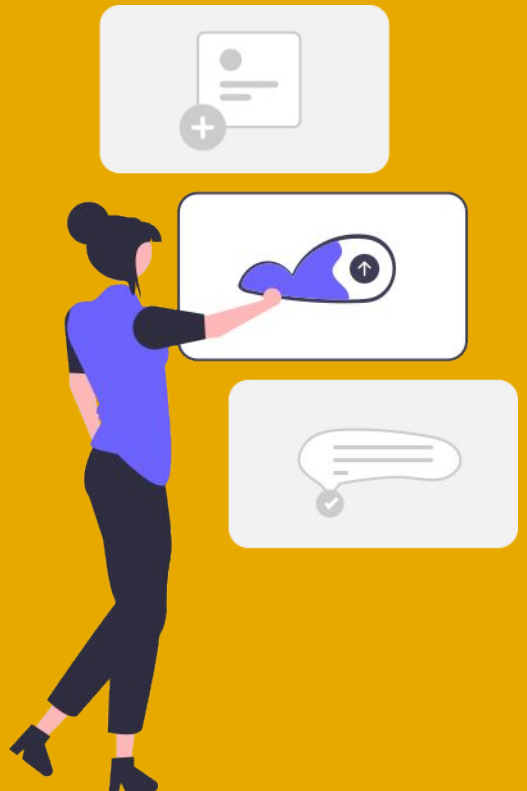
```jsx
import Task from './Task';

export default function TaskList() {
  const tasks = ['Learn React', 'Build a project', 'Explore Hooks'];
  return (
    <ul>
      {tasks.map((task, index) => (
        <Task key={index} task={task} />
      ))}
    </ul>
  );
}
```

Introducción a Hooks

Los Hooks permiten gestionar estados y efectos secundarios en los componentes funcionales.

Agrega estados al componente TaskList.jsx:

```jsx
import { useState } from 'react';
import Task from './Task';

export default function TaskList() {
  const [tasks, setTasks] = useState(['Learn React', 'Build a project', 'Explore Hooks']);

  const addTask = () => {
    setTasks([...tasks, `Task ${tasks.length + 1}`]);
  };

  return (
    <div>
      <button onClick={addTask}>Add Task</button>
      <ul>
        {tasks.map((task, index) => (
          <Task key={index} task={task} />
        ))}
      </ul>
    </div>
  );
}
```



JSX, la Simplificación de ReactJS

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Porqué usar JSX

JSX es una extensión de sintaxis para JavaScript que **facilita la creación de interfaces de usuario en React**. Su similitud con HTML lo hace intuitivo para los desarrolladores.

Actualiza el Header.jsx:

```jsx
export default function Header() {
  const appName = 'My Todo App';
  const currentDate = new Date().toLocaleDateString();

  return (
    <header>
      <h1>{appName}</h1>
      <p>{`Today's date: ${currentDate}`}</p>
    </header>
  );
}
```

Las llaves {} permiten incluir expresiones de JavaScript.

La interpolación de cadenas como Today's date: ${currentDate} combina HTML y JavaScript de forma natural.

JSX también es una Expresión

JSX puede asignarse a variables, pasarse como parámetros o devolverse desde funciones.

Actualiza App.jsx:

```
import Header from './Header';
import TaskList from './TaskList';

export default function App() {
  const footer = <footer>(c) 2024 My Todo App</footer>;

  return (
    <div>
      <Header />
      <TaskList />
      {footer}
    </div>
  );
}
```

JSX como una expresión puede almacenarse en variables (footer) y reutilizarse.

Especificando Atributos con JSX

En JSX, los atributos son similares a HTML, pero con convenciones de nombres de JavaScript.

Agrega estilos en línea a Header.jsx:

```jsx
export default function Header() {
  const appName = 'My Todo App';
  const headerStyle = {
    textAlign: 'center',
    color: 'blue',
  };

  return (
    <header style={headerStyle}>
      <h1>{appName}</h1>
    </header>
  );
}
```

Atributos como style usan objetos JavaScript en lugar de cadenas.

Las claves del objeto siguen la convención camelCase (textAlign).

Previniendo Ataques de Inyección con JSX

JSX escapa automáticamente cualquier contenido peligroso.

Actualiza TaskList.jsx:

```
export default function TaskList() {
  const tasks = ['<script>alert("Hacked!")</script>', 'Learn React', 'Explore Hooks'];

  return (
    <ul>
      {tasks.map((task, index) => (
        <li key={index}>{task}</li>
      ))}
    </ul>
  );
}
```

Aunque tasks contiene HTML malicioso, JSX lo muestra como texto plano, previniendo inyecciones de código.

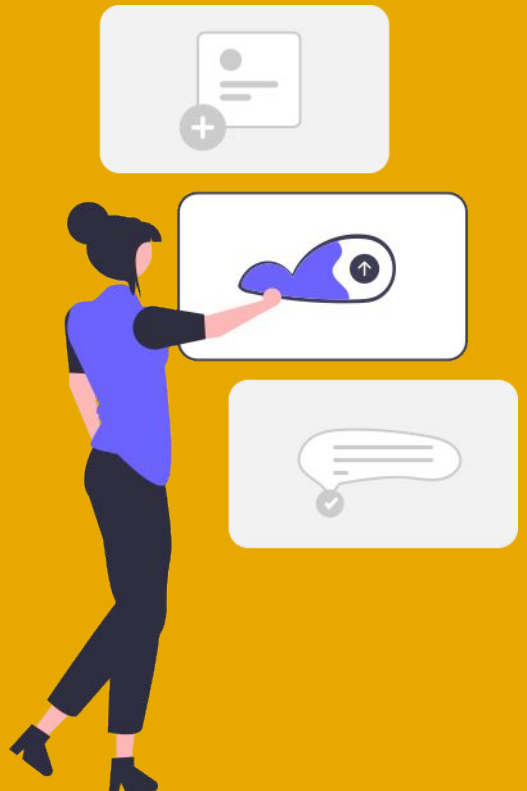
Comentarios en JSX

Los comentarios dentro de JSX se colocan dentro de llaves usando sintaxis específica.

Actualiza TaskList.jsx:

```jsx
export default function TaskList() {
  const tasks = ['Learn React', 'Build a project', 'Explore Hooks'];

  return (
    <div>
      {/* Mostrar lista de tareas */}
      <ul>
        {tasks.map((task, index) => (
          <li key={index}>{task}</li>
        ))}
      </ul>
    </div>
  );
}
```

Los comentarios deben colocarse dentro de {/* */}.



Implementar componentes reutilizables

Aspectos fundamentales de la Componentización

La componentización en React permite dividir la UI en pequeñas piezas reutilizables, lo que mejora la organización, mantenibilidad y escalabilidad del código.

Beneficios:

- Reutilización de código.
- Separación de preocupaciones.
- Escalabilidad y consistencia.

Creamos un componente reutilizable Button.jsx:

```
export default function Button({ label, onClick }) {
  return <button onClick={onClick}>{label}</button>;
}
```

Button recibe label y onClick como propiedades, lo que permite personalización.

Reconocer los Elementos y Sintaxis para Componentes

Los componentes en React son funciones que devuelven elementos JSX. Usan props para recibir datos y comportamientos.

Actualizar TaskList.jsx para usar Button:

```jsx
import Button from './Button';

export default function TaskList({ tasks, onDelete }) {
  return (
    <ul>
      {tasks.map((task, index) => (
        <li key={index}>
          {task}
          <Button label="Delete" onClick={() => onDelete(index)} />
        </li>
      ))}
    </ul>
  );
}
```

Button se usa para eliminar tareas. onClick ejecuta onDelete con el índice de la tarea. Calma si se rompe el código ;), aun no terminamos

Usar Componentes Previamente Construidos

Los componentes construidos pueden reutilizarse con diferentes configuraciones mediante props.

Crea TaskForm.jsx:

```jsx
import { useState } from 'react';

export default function TaskForm({ onAdd }) {
  const [task, setTask] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    if (task.trim()) {
      onAdd(task);
      setTask('');
    }
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Add a task"
        value={task}
        onChange={(e) => setTask(e.target.value)}
      />
      <button type="submit">Add</button>
    </form>
  );
}
```

Usar Componentes Previamente Construidos

Integra en App.jsx:

```
import { useState } from 'react';
import Header from './Header';
import TaskList from './TaskList';
import TaskForm from './TaskForm';

export default function App() {
  const [tasks, setTasks] = useState([]);

  const addTask = (task) => {
    setTasks([...tasks, task]);
  };

  const deleteTask = (index) => {
    setTasks(tasks.filter((_, i) => i !==
index));
  };
```

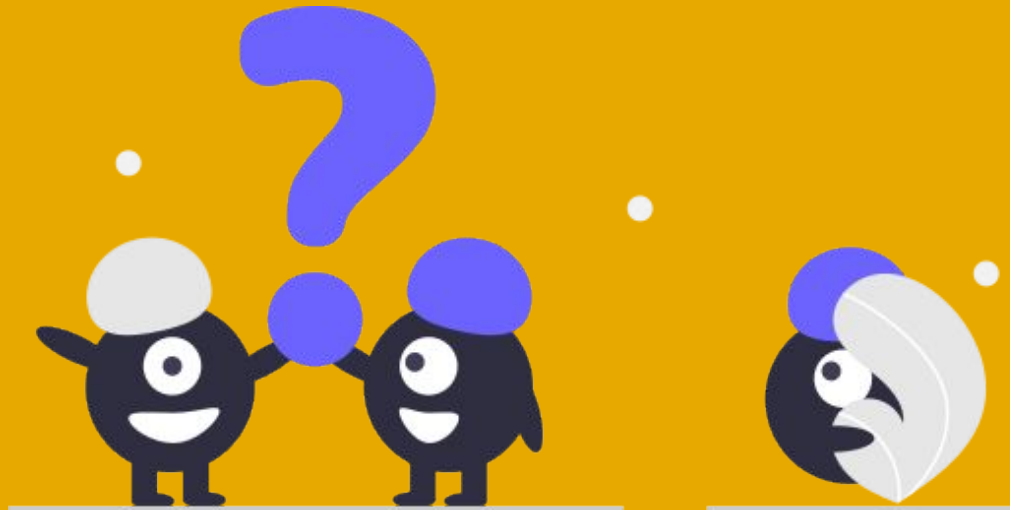
```
  return (
    <div>
      <Header />
      <TaskForm onAdd={addTask} />
      <TaskList tasks={tasks}
onDelete={deleteTask} />
    </div>
  );
}
```

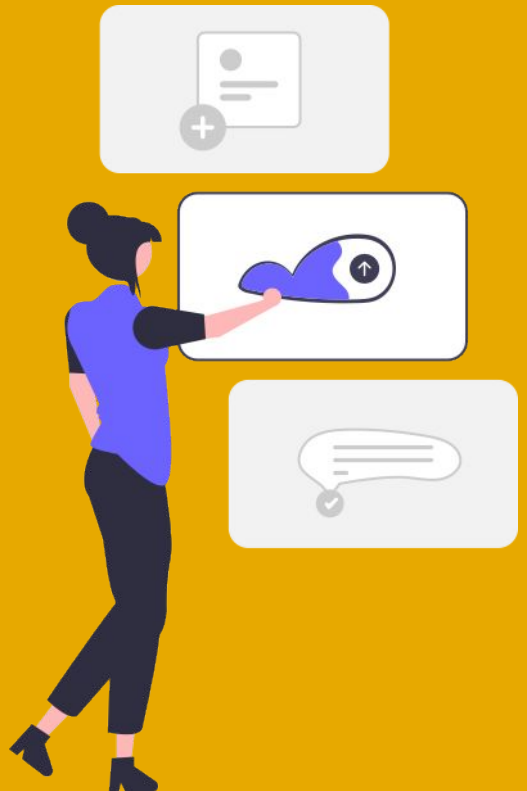
TaskForm permite agregar nuevas tareas.

TaskList muestra las tareas y permite eliminarlas.

Los métodos **addTask** y **deleteTask** gestionan el estado.

¿Qué te pareció tu primer proyecto con React?





Extras 😁!!!

El ciclo de vida ReactJS

El ciclo de vida en React describe las etapas por las que pasa un componente: montaje, actualización y desmontaje. Comprender estas fases permite **manipular** datos, realizar **peticiones** o **limpiar** recursos adecuadamente.

- **Montaje:** componentDidMount se usa para inicializar datos.
- **Actualización:** Métodos como componentDidUpdate responden a cambios en props o state.
- **Desmontaje:** componentWillUnmount permite limpiar recursos como listeners o intervalos.

Desplegando datos en la interfaz (Renderización)

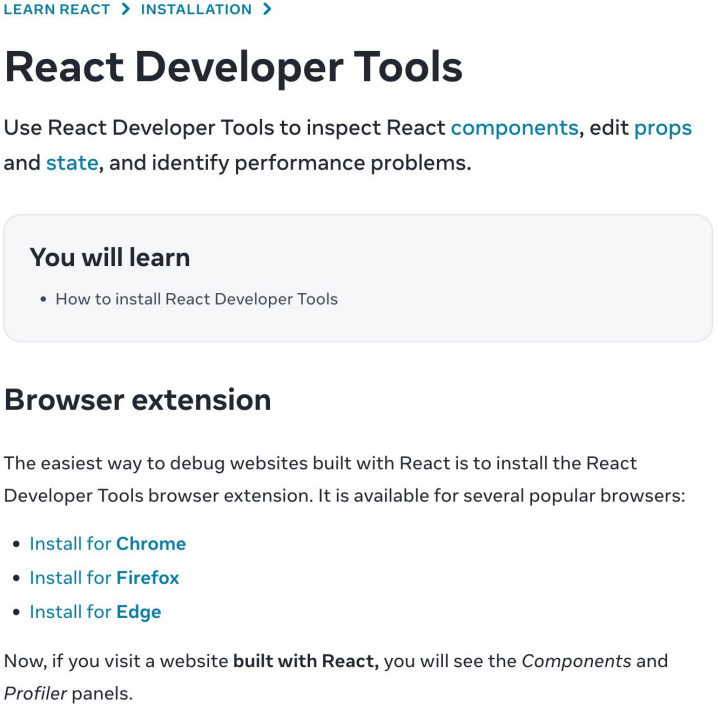
React utiliza el método **render()** para actualizar la interfaz de usuario basándose en los cambios en el estado o las propiedades del componente.

Aquí, el componente UserProfile despliega dinámicamente los datos proporcionados a través de las props.

```
function UserProfile({ name, age }) {
  return (
    <div>
      <h1>{name}</h1>
      <p>Edad: {age}</p>
    </div>
  );
}
```

Uso de extensión browser “React Developer Tools”

React Developer Tools es una **extensión** para inspeccionar jerarquías de componentes, sus props y estados, facilitando el debugging.
https://react.dev/learn/react-developer-tools



LEARN REACT › INSTALLATION ›

React Developer Tools

Use React Developer Tools to inspect React components, edit props and state, and identify performance problems.

You will learn

- How to install React Developer Tools

Browser extension

The easiest way to debug websites built with React is to install the React Developer Tools browser extension. It is available for several popular browsers:

- Install for **Chrome**
- Install for **Firefox**
- Install for **Edge**

Now, if you visit a website **built with React,** you will see the *Components* and *Profiler* panels.

Introducción a uso de API’s

React permite consumir **APIs REST** utilizando funciones como **fetch()** o librerías como **Axios** para manejar datos externos.

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

function FetchData() {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/todos")
      .then((response) => response.json())
      .then((json) => setData(json));
  }, []);

  return (
    <ul>
      {data.map((item) => (
        <li key={item.id}>{item.title}</li>
      ))}
    </ul>
  );
}
```

Notación y palabras reservadas en JSX

JSX utiliza una combinación de JavaScript y HTML con **notaciones específicas**. Palabras reservadas como **class** se sustituyen por **className**.

```
function Button() {
  return <button className="btn-primary">Click Me!</button>;
}
```

Especificando hijos con JSX

JSX permite incluir elementos hijos **dentro de un componente**.

```
function Card({ children }) {
  return <div className="card">{children}</div>;
}

function App() {
  return (
    <Card>
      <h2>Título</h2>
      <p>Contenido</p>
    </Card>
  );
}
```

Representación de objetos en JSX

JSX permite usar expresiones de JavaScript, incluidas **representaciones de objetos**.

```
const user = { name: "John", age: 30 };

function DisplayUser() {
  return <p>{`Nombre: ${user.name}, Edad: ${user.age}`}</p>;
}
```

El rol de Babel en la conversión de objetos

Babel convierte el código JSX en JavaScript estándar, permitiendo la **compatibilidad con navegadores**.

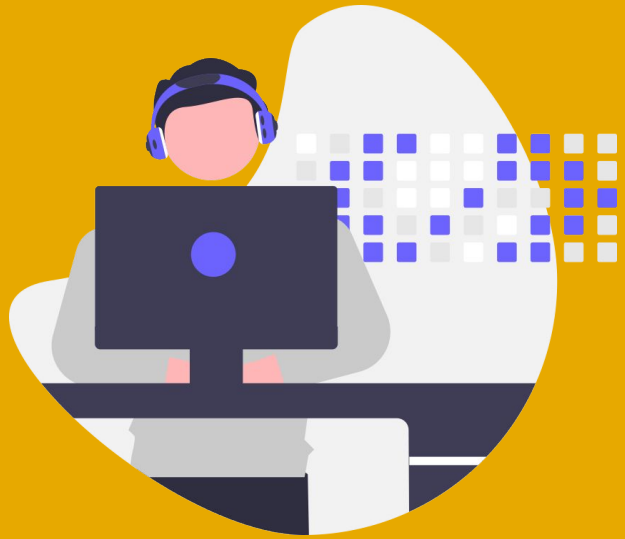
Ejemplo de Código Pre-Babel:

```
const element = <h1>Hello, world!</h1>;
```

Ejemplo convertido por Babel:

```
const element = React.createElement("h1", null,
"Hello, world!");
```

Pongamos a prueba lo aprendido 😊!!!



Adalid aspasia

Ejercicio Guiado: Plataforma Inmobiliaria

En este ejercicio, desarrollaremos una aplicación básica de una plataforma inmobiliaria utilizando los elementos fundamentales de ReactJS. Implementaremos **componentes reutilizables** para listar propiedades, mostrar detalles de una propiedad y realizar solicitudes de contacto. Usaremos **JSX**, **props**, **hooks** y formularios para manejar la interacción del usuario y desplegar datos dinámicos.

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 1: Preparar el Entorno de Trabajo

```
npm create vite@latest real-estate-app --template react
cd real-estate-app
npm install
npm run dev
```

```
real-estate-app/
├── node_modules/
├── public/
│   └── vite.svg
├── src/
│   ├── App.css
│   ├── App.jsx
│   ├── index.css
│   ├── main.jsx
│   └── assets/
│       └── react.svg
├── .gitignore
├── eslint.config.js
├── index.html
├── package-lock.json
├── README.md
└── vite.config.js
```

Esta será nuestra estructura inicial.

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 2: Elimina los estilos que vienen por defecto en Main.jsx

- Asi deberia verse src/Main.jsx al inicio

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

- Asi deberia verse src/Main.jsx al final

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 3: Agrega Bootstrap

En la carpeta raíz de tu proyecto se encuentra el archivo **index.html** en el agregaremos el CDN de Bootstrap

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Vite + React</title>
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
      crossorigin="anonymous"
    />
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 4: Modifica el archivo App.js

- Agrega los componentes **PropertyList** y **ContactForm.**
- Simula la implementación de una **API.**

```
import { useState, useEffect } from "react";
import PropertyList from "./components/PropertyList";
import ContactForm from "./components/ContactForm";

function App() {
  const [properties, setProperties] = useState([]);

  useEffect(() => {
    // Simular una API
    setProperties([
      { id: 1, name: "Casa Moderna", location: "Ciudad A", price: 120000 },
      { id: 2, name: "Departamento", location: "Ciudad B", price: 95000 },
      { id: 3, name: "Casa de Campo", location: "Ciudad C", price: 150000 },
    ]);
  }, []);
```

```
  return (
    <div className="container my-5">
      <h1 className="text-center">Plataforma
Inmobiliaria</h1>
      <div className="row">
        <div className="col">
          <PropertyList properties={properties} />
          <ContactForm />
        </div>
      </div>
    </div>
  );
}

export default App;
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 5: Crear Componentes Reutilizables

- Genera una carpeta llamada components y agrega:
  - **Componente PropertyCard:** Muestra información básica de una propiedad.
  - **Componente PropertyList:** Lista todas las propiedades disponibles.
  - **Componente ContactForm:** Permite enviar una solicitud de contacto.

```
real-estate-app/
├── node_modules/
├── public/
│   └── vite.svg
├── src/
│   ├── App.css
│   ├── App.jsx
│   ├── index.css
│   ├── main.jsx
│   ├── components/
│   │   ├── ContactForm.jsx
│   │   ├── PropertyCard.jsx
│   │   └── PropertyList.jsx
│   └── assets/
│       └── react.svg
├── .gitignore
├── eslint.config.js
├── index.html
├── package-lock.json
├── README.md
└── vite.config.js
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 5: Crear Componentes Reutilizables

- **Ubicación:** src/components/PropertyCard.jsx

```jsx
import PropTypes from 'prop-types';

function PropertyCard({ name, location, price }) {
  return (
    <div className="col">
      <div className="card">
        <div className="card-body">
          <h5 className='card-title'>{name}</h5>
          <p className='card-text'>{location}</p>
          <p className='card-text'>Precio: ${price}</p>
        </div>
      </div>
    </div>
  );
}
```

```jsx
PropertyCard.propTypes = {
  name: PropTypes.string.isRequired,
  location: PropTypes.string.isRequired,
  price: PropTypes.number.isRequired,
};

export default PropertyCard;
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 5: Crear Componentes Reutilizables

- **Ubicación:** src/components/PropertyList.jsx

```jsx
import PropTypes from 'prop-types';
import PropertyCard from "./PropertyCard";

function PropertyList({ properties }) {
  return (
    <div className="row row-cols-1 row-cols-md-3 g-3 my-2">
      {properties.map((property) => (
        <PropertyCard
          key={property.id}
          name={property.name}
          location={property.location}
          price={property.price}
        />
      ))}
    </div>
  );
}
```

```jsx
PropertyList.propTypes = {
  properties: PropTypes.arrayOf(
    PropTypes.shape({
      id: PropTypes.number.isRequired,
      name: PropTypes.string.isRequired,
      location: PropTypes.string.isRequired,
      price: PropTypes.number.isRequired,
    })
  ).isRequired,
};

export default PropertyList;
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 5: Crear Componentes Reutilizables

- **Ubicación:** src/components/ContactForm.jsx

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

function ContactForm() {
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    message: "",
  });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log("Formulario enviado:", formData);
  };
```

Ejercicio Guiado: Plataforma Inmobiliaria

Paso 5: Crear Componentes Reutilizables

- **Ubicación:** src/components/ContactForm.jsx

```
return (
  <div className="card mt-5">
    <div className="card-body">
      <form className="contact-form" onSubmit={handleSubmit}>
        <div className="mb-3">
          <input
            type="text"
            name="name"
            className="form-control"
            placeholder="Tu nombre"
            value={formData.name}
            onChange={handleChange}
            required
          />
        </div>
```

```
        <div className="mb-3">
          <input
            type="email"
            name="email"
            className="form-control"
            placeholder="Tu correo"
            value={formData.email}
            onChange={handleChange}
            required
          />
        </div>
        <div className="mb-3">
        <textarea
          name="message"
          rows="4"
          placeholder="Tu mensaje"
          className="form-control"
          value={formData.message}
          onChange={handleChange}
          required
        ></textarea>
        </div>
```

```
          <div className="d-grid gap-2">
            <button type="submit" className="btn
btn-primary">Enviar</button>
          </div>
        </form>
      </div>
    </div>
  );
}

export default ContactForm;
```

Ejercicio Guiado: Plataforma Inmobiliaria

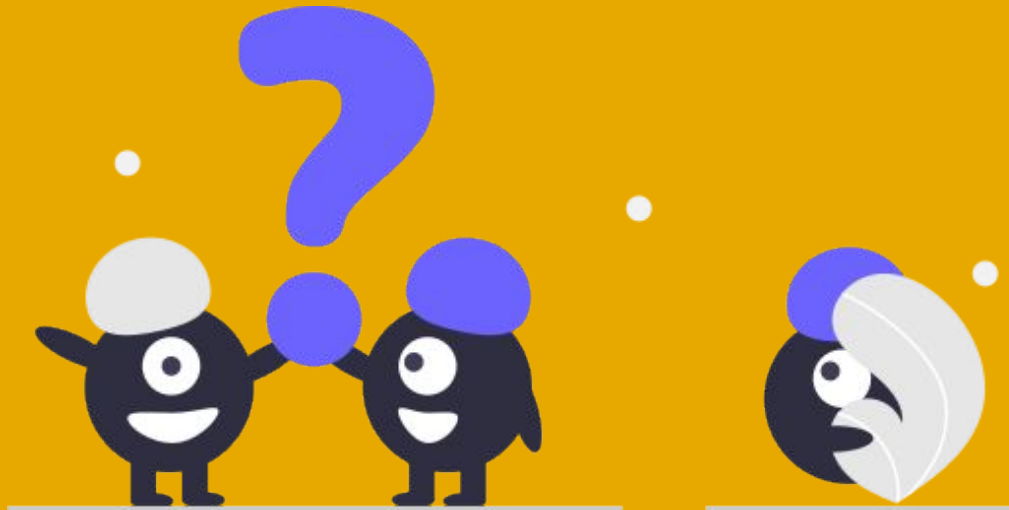
Paso 6: Ejecuta el proyecto

- Usa el comando **npm run dev**

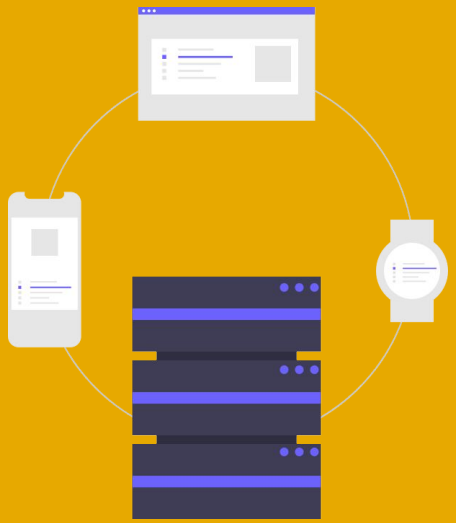
```
npm run dev

 VITE v6.0.2  ready in 84 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Asi se vera la web, al usar el formulario asegúrate de revisar la **consola** con el inspector de elementos

¿Como te fue con el Ejercicio Guiado?



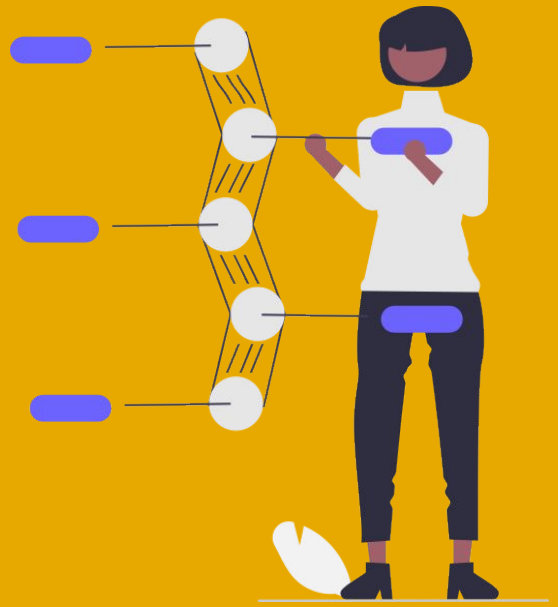
Adalid aspasia



Visita el Repositorio de GitHub para ver ejemplos

Adalid
aspasia
LA FORMACIÓN DE TU FUTURO

Resumen de lo aprendido 🧑‍🎓



Adalid aspasia

Resumen de lo aprendido

- **Fundamentos de React y JSX:** Aprendimos cómo JSX simplifica la creación de interfaces mediante una sintaxis declarativa, permitiendo insertar expresiones, manejar atributos.
- **Componentización y Props:** Desarrollamos componentes reutilizables para estructurar aplicaciones, entendiendo cómo pasar datos con props, gestionar listas con keys únicas y construir formularios interactivos.
- **Ciclo de vida y Hooks:** Exploramos el ciclo de vida de los componentes en React y la introducción a Hooks como useState y useEffect para manejar estados y efectos secundarios.
- **Estilos y API's:** Implementamos hojas de estilo en React y aprendimos a consumir datos desde API's, integrando dinámicamente contenido en nuestras aplicaciones y personalizando componentes reutilizables.

GRACIAS POR TU ATENCIÓN

Nos vemos en la próxima clase

Chile
Adalid
aspasia
LA FORMACIÓN DE TU FUTURO