INTRODUCCIÓN AL DESARROLLO FRONTEND

CON REACT

2023

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

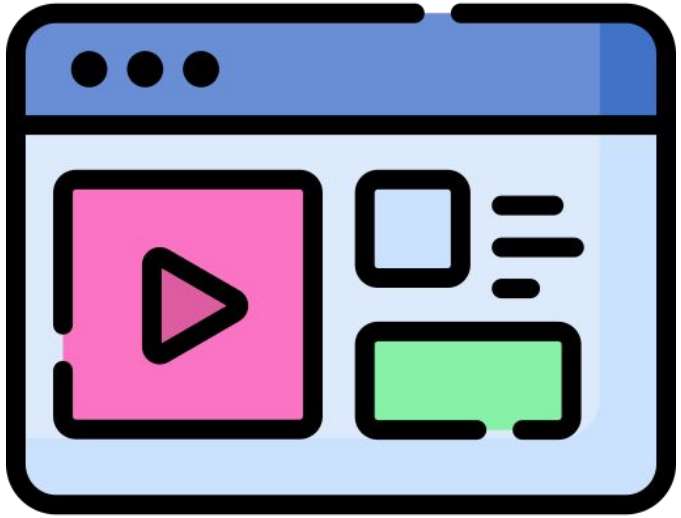
Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos

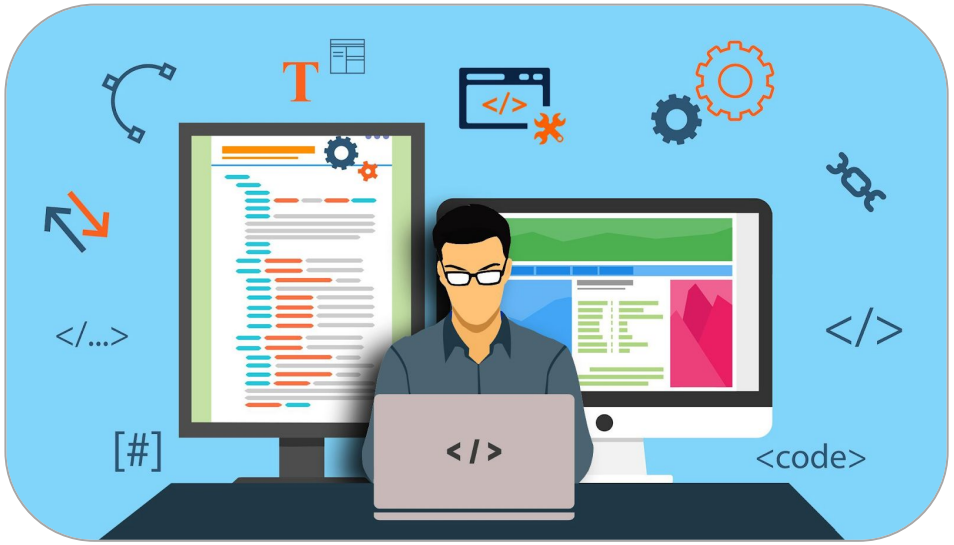
Clase 09 - Contenido

Integración con servicios de terceros

- Integración con servicios de terceros
- Uso de APIs
- Autenticación en servicios de terceros
- Ejemplos de integración con servicios de terceros
- Manejo de errores y excepciones



Objetivos de la Clase



- **Integración con Servicios de Terceros:** Comprender cómo y por qué interactuar con servicios externos y APIs en el contexto de React.
- **Manejo Robusto de Errores:** Aprender a identificar, capturar y tratar errores en React, ofreciendo un feedback adecuado al usuario.
- **Integraciones Prácticas:** Familiarizarse con ejemplos concretos de integración.
- **Uso Eficiente de APIs:** Profundizar en las técnicas y métodos para consumir APIs de forma efectiva usando React.
- **Herramientas y Debugging:** Adquirir conocimientos básicos sobre las herramientas disponibles para depurar y optimizar aplicaciones React.

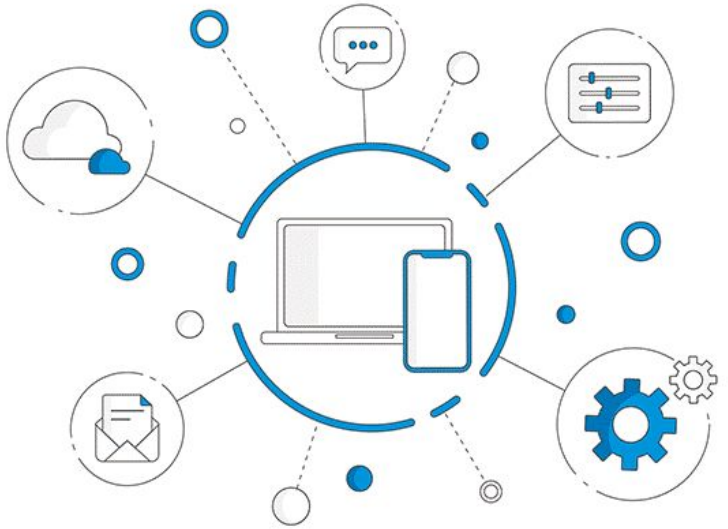
Integración con Servicios de Terceros

¿Qué significa? Integrar un servicio de terceros implica conectarse y hacer uso de capacidades externas desde nuestra aplicación. Estas capacidades pueden ir desde bases de datos, pasando por sistemas de pago, hasta redes sociales y más.

Importancia de las APIs: Las APIs (Interfaces de Programación de Aplicaciones) permiten que las aplicaciones se comuniquen entre sí. Es gracias a ellas que podemos acceder a funciones y datos de otros servicios.

Ventajas:

- **Especialización:** No necesitamos reinventar la rueda; utilizamos servicios especializados para tareas específicas.
- **Escalabilidad:** Al depender de grandes proveedores, se obtiene una infraestructura robusta y escalable.
- **Actualizaciones y Mantenimiento:** Delegamos la responsabilidad de mantener y actualizar ciertas funciones.
- **En el Contexto de React:** Gracias a la flexibilidad de React y su ecosistema, la integración con servicios externos es una tarea simplificada, permitiéndonos enfocarnos en la experiencia del usuario y la lógica de la aplicación.

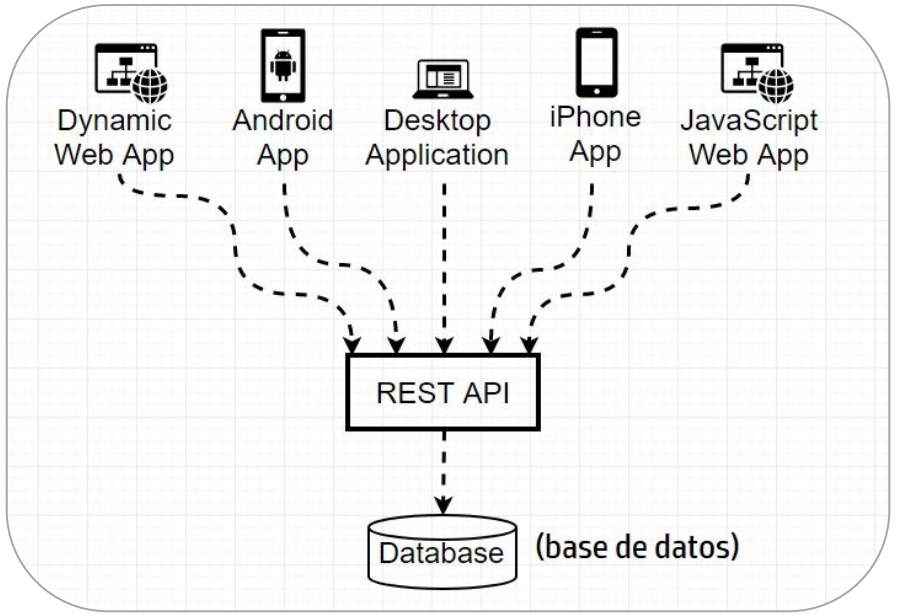


¿Qué son los Servicios de Terceros?

- Los servicios de terceros son plataformas, herramientas o recursos externos que las aplicaciones pueden utilizar para potenciar o extender su funcionalidad sin tener que desarrollarla desde cero.
- **Tipos de Servicios:**
  - Almacenamiento: Como bases de datos o soluciones de almacenamiento en la nube.
  - Autenticación: Servicios como Auth0 o Firebase Authentication.
  - Pagos: Ejemplos son Stripe o PayPal.
  - Redes Sociales: Integraciones con Facebook, Twitter, Instagram, etc.
  - Y muchos más: Analítica, publicidad, geolocalización, entre otros.
- **Importancia en el Mundo Web Actual:**
  - Economía de Desarrollo: Ahorra tiempo y recursos al no tener que desarrollar y mantener ciertas funcionalidades.
  - Especialización: Los servicios se centran en hacer una cosa y hacerla bien, lo que a menudo resulta en un rendimiento y seguridad superiores.
  - Adaptabilidad: Facilita la adaptación a nuevas tecnologías o estándares emergentes.
  - Interoperabilidad: La naturaleza modular del desarrollo web moderno significa que las aplicaciones deben poder comunicarse e interactuar con una variedad de otros servicios y plataformas.

Uso de APIs en React

- **¿Qué es una API?:** Una API (Interfaz de Programación de Aplicaciones) permite que las aplicaciones se comuniquen entre sí. Es el "puente" para obtener datos o funciones de servicios externos.
- **React y APIs:**
  - Component Lifecycle: Utilizar métodos del ciclo de vida, como componentDidMount, para hacer peticiones cuando el componente se monta.
  - Hooks: Con los Hooks, especialmente useEffect, se puede realizar el fetch de datos de manera más intuitiva en componentes funcionales.
- **Herramientas Comunes:**
  - **Fetch API:** API nativa de JavaScript para hacer peticiones HTTP.
  - **Axios:** Biblioteca popular para hacer peticiones HTTP, maneja mejor los errores y permite cancelar peticiones, entre otras ventajas.



Uso de APIs en React

El código muestra un componente React funcional llamado App que tiene como objetivo obtener y mostrar una lista de datos desde una API externa.

- **useState:** El hook useState es utilizado para establecer el estado local del componente. En este caso, el estado data es inicializado como un array vacío y setData es la función que se utilizará para actualizar ese estado.
- **useEffect:** El hook useEffect se utiliza para ejecutar efectos secundarios en componentes funcionales. En este ejemplo, se utiliza para realizar una petición HTTP a una API cuando el componente se monta por primera vez.
- **axios.get:** Se utiliza la biblioteca Axios para hacer una petición GET a la URL 'https://api.example.com/data'.
- **.then(response => {...}):** Una vez que la petición es exitosa, se utiliza la función .then para tomar la respuesta y establecer los datos obtenidos en el estado del componente con setData(response.data).
- **Renderizado:** En el cuerpo del componente (el área de retorno), se mapean los datos obtenidos para mostrar cada ítem individualmente en un elemento <p>.

```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

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

  useEffect(() => {
    axios.get('https://api.example.com/data')
      .then(response => {
        setData(response.data);
      });
  }, []);

  return (
    <div>
      {data.map(item => (
        <p key={item.id}>{item.name}</p>
      ))}
    </div>
  );
}

export default App;
```

Función de useEffect

El hook useEffect en React se utiliza para realizar "efectos secundarios" en componentes funcionales. Estos efectos secundarios pueden incluir operaciones como:

- Fetching de datos
- Suscripciones manuales
- Cambios manuales al DOM

El useEffect toma dos argumentos:

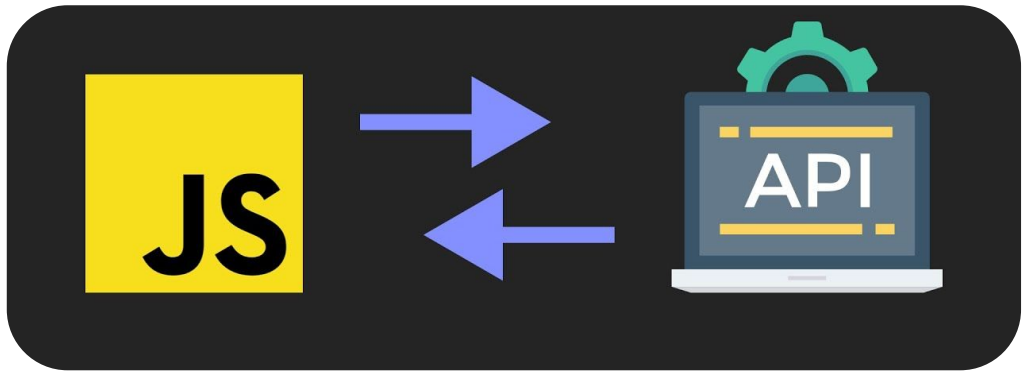
- Una función que contiene el código del efecto secundario.
- Una lista de dependencias.

En el ejemplo proporcionado, useEffect se utiliza para hacer una petición a una API cuando el componente se monta. La lista de dependencias, representada por el array vacío [], significa que el efecto se ejecutará sólo una vez, similar a componentDidMount en componentes de clase. Si esa lista de dependencias se omite, el efecto se ejecutaría después de cada renderizado. Si se incluyen variables específicas en la lista, el efecto se ejecutará sólo cuando esas variables cambien.

```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

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

  useEffect(() => {
    axios.get('https://api.example.com/data')
      .then(response => {
        setData(response.data);
      });
  }, []);

  return (
    <div>
      {data.map(item => (
        <p key={item.id}>{item.name}</p>
      ))}
    </div>
  );
}

export default App;
```

Métodos para Consumir APIs en React

- **Fetch API:**
  - Herramienta nativa de JavaScript para realizar peticiones HTTP.
  - Usa Promesas para manejar respuestas y errores.
- **Axios:**
  - Biblioteca de JavaScript diseñada para consumir APIs.
  - Maneja mejor los errores y tiene una sintaxis más limpia comparada con Fetch.
- **jQuery AJAX:**
  - Aunque no se recomienda en aplicaciones modernas de React, es posible usar jQuery para hacer peticiones AJAX.
- **GraphQL y Apollo Client:**
  - Si tu API usa GraphQL, Apollo Client es una excelente herramienta para consumir la API en React.
  - Permite especificar exactamente qué datos se desean obtener.
- **WebSockets y Socket.io:**
  - Para aplicaciones en tiempo real o cuando se necesita una comunicación bidireccional.
  - No es una petición HTTP tradicional, sino una conexión persistente.



Fetch API:

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:',
error));
```

Axios

```
import axios from 'axios';

axios.get('https://api.example.com/data')
  .then(response => console.log(response.data))
  .catch(error => console.error('Error:', error));
```

jQuery AJAX

```
$.ajax({
  url: 'https://api.example.com/data',
  type: 'GET',
  success: data => console.log(data),
  error: error => console.error('Error:', error)
});
```

GraphQL y Apollo Client

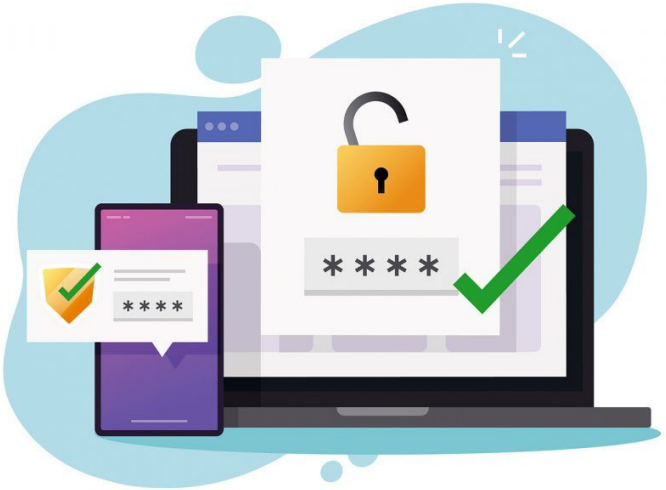
```
import { useQuery } from '@apollo/client';


const { loading, error, data } = useQuery(MY_QUERY);
```

WebSockets y Socket.io

```
import { io } from 'socket.io-client';
const socket = io('https://api.example.com');
```

Autenticación en Servicios de Terceros

- **¿Qué es la Autenticación?:**
  - Es el proceso de verificar la identidad de un usuario o sistema.
  - Asegura que el actor que intenta acceder a un recurso es quien dice ser.
- **¿Por qué es importante?:**
  - Seguridad: Proteger información sensible y recursos de accesos no autorizados.
  - Personalización: Proporcionar una experiencia de usuario personalizada basada en su identidad.
  - Control: Limitar el acceso a ciertas áreas o funciones de una aplicación basadas en roles o permisos.
- **Servicios Externos y Autenticación:**
  - Al interactuar con servicios de terceros (APIs, bases de datos, etc.), a menudo necesitamos autenticarnos para acceder a datos o realizar ciertas operaciones.
  - La autenticación garantiza que sólo los usuarios o sistemas autorizados puedan interactuar con estos servicios externos.
- **Métodos Comunes:**
  - Autenticación Básica: Usuario y contraseña codificados en base64.
  - Tokens (como JWT): Tokens generados que representan la identidad del usuario y, a menudo, incluyen detalles como roles o permisos.
  - OAuth: Un estándar de autorización que permite a las aplicaciones obtener acceso limitado a cuentas de usuario en un servicio HTTP, como Facebook o Google.



Métodos de Autenticación y su Implementación en React

- **Tokens (JWT - JSON Web Tokens):**
  - Descripción: Es una cadena codificada que representa reclamaciones o afirmaciones sobre un usuario. Es seguro y puede ser firmado para garantizar su integridad.
  - Uso en React: Se suele almacenar en el localStorage o sessionStorage y se adjunta en el header de las peticiones HTTP.
- **OAuth 2.0:**
  - Descripción: Protocolo de autorización que permite a las aplicaciones obtener un acceso limitado a cuentas de usuario en un servicio externo.
  - Uso en React: Se redirige al usuario al proveedor de OAuth, y una vez autenticado, se redirige de vuelta con un código o token.

```
const token = localStorage.getItem('token');
axios.get('https://api.example.com/data', {
  headers: { 'Authorization': `Bearer ${token}` }
});
```

Integración con JSONPlaceholder

- **JSONPlaceholder:**
  - API REST gratuita que simula datos falsos para pruebas y prototipos.
  - Provee recursos como posts, comments, users, etc.
- **Instalación de Axios:**
  Para facilitar las peticiones HTTP, utilizaremos Axios:

```
npm install axios
```

- **Ejemplo de Consumo de la API --->**

```jsx
import React, { useState, useEffect } from 'react';
import axios from 'axios';

function Posts() {
  const [posts, setPosts] = useState([]);

  useEffect(() => {
    axios.get('https://jsonplaceholder.typicode.com/posts')
      .then(response => {
        setPosts(response.data);
      })
      .catch(error => {
        console.error("Error fetching posts:", error);
      });
  }, []);

  return (
    <div>
      <h2>Posts from JSONPlaceholder</h2>
      <ul>
        {posts.map(post => (
          <li key={post.id}>
            <h3>{post.title}</h3>
            <p>{post.body}</p>
          </li>
        ))}
      </ul>
    </div>
  );
}

export default Posts;
```

Manejo de Errores y Excepciones en React

- Introducción:
  - El manejo adecuado de errores es crucial para crear aplicaciones robustas y user-friendly.
  - Permite identificar y corregir problemas, además de proporcionar feedback útil a los usuarios y desarrolladores.
- Error Boundaries:
  - Proporcionan una manera de capturar y manejar errores en componentes descendentes.
- React-error-boundary:
  - Librería alternativa, moderna y más simple para implementar Error Boundaries en React.

```
import { ErrorBoundary } from
'react-error-boundary'

function MyApp() {
  return (
    <ErrorBoundary
      FallbackComponent={({ error }) =>
<div>Algo salió mal: {error.message}</div>}
    >
      <MyComponent />
    </ErrorBoundary>
  )
}
```

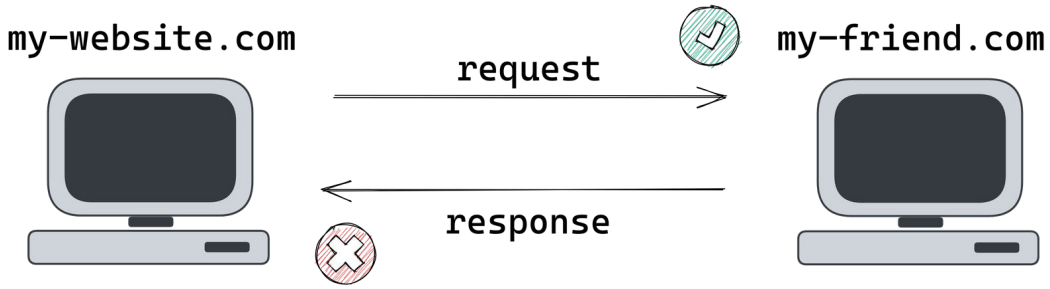
Try-Catch

- La estructura try-catch permite capturar excepciones y manejarlas de manera adecuada, evitando que los errores no atrapados provoquen fallos en la aplicación.
- Uso básico de Try-Catch:
  - Un bloque try envuelve el código que podría lanzar una excepción, mientras que un bloque catch captura y maneja cualquier error que ocurra.

```
import axios from 'axios';

async function fetchData() {
  try {
    const response = await
axios.get('https://api.example.com/data');
    console.log(response.data);
  } catch (error) {
    console.error('An error occurred:',
error.message);
  }
}
```

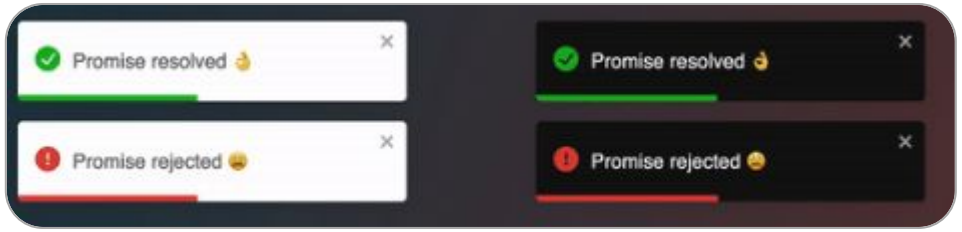
Entendiendo CORS (Cross-Origin Resource Sharing)

- Mecanismo que permite solicitudes de origen cruzado seguras entre diferentes dominios.
- Necesidad: Prevenir accesos no autorizados, permitiendo a los servidores especificar quién puede acceder a sus recursos.
- Headers Clave:
  - Access-Control-Allow-Origin: Especifica dominios permitidos.
  - Access-Control-Allow-Methods: Métodos HTTP permitidos.
  - Access-Control-Allow-Headers: Headers HTTP permitidos.
- Implementación:
  - Servidor: Configurar headers de respuesta.
  - Cliente: Manejar errores de CORS adecuadamente.
- Herramientas y Recursos: Plugins, middleware y documentación para manejar e implementar CORS.



Importancia del Feedback

Es crucial proporcionar feedback al usuario para indicar el estado actual de la aplicación, especialmente cuando ocurren errores.



Tipos de Feedback:

- **Mensajes de Error:** Informan sobre problemas específicos.
- **Indicadores de Carga:** Muestran que una operación está en curso.
- **Confirmaciones:** Confirman que una acción se completó exitosamente.

Bibliotecas Útiles:

- react-toastify para mostrar notificaciones.
- react-spinners para indicadores de carga.

Importancia del Feedback: Implementación en React

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

function MiComponente() {
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState(null);

  const fetchData = async () => {
    setLoading(true);
    setError(null);
    try {
      // ... código para obtener datos
      setLoading(false);
    } catch (error) {
      setLoading(false);
      setError(error);
    }
  };

  return (
    <div>
      {loading && <div>Loading...</div>}
      {error && <div>Error: {error.message}</div>}
      {/* ... renderizar contenido normal */}
    </div>
  );
}
```

Debugging

Debugging es esencial para identificar y solucionar problemas en tu aplicación. Afortunadamente, React tiene una variedad de herramientas y técnicas disponibles para ayudar en esta tarea.

Herramientas de Debugging:

- **React DevTools:**
  - Extensión para navegadores que permite inspeccionar el árbol de componentes, revisar el estado y las props, y entender mejor el rendimiento de tu aplicación.
- **Console.log:**
  - Una técnica básica pero poderosa para rastrear el flujo de datos y entender cómo y cuándo cambian los valores.

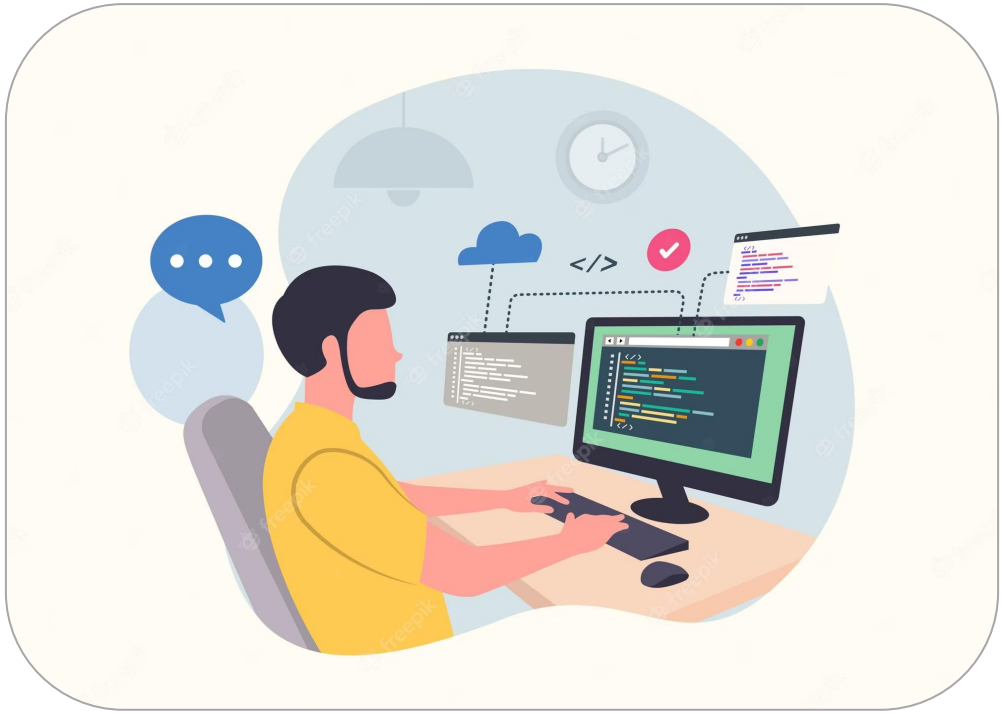
```
console.log('Value of state:', this.state.value);
```

- **Breakpoints en el navegador:**
  - Establecer puntos de interrupción en el código para pausar la ejecución y examinar el estado de la aplicación en un punto específico.

Bueno, Vamo a Codea!!!!

Actividad 9: Integración API de Terceros

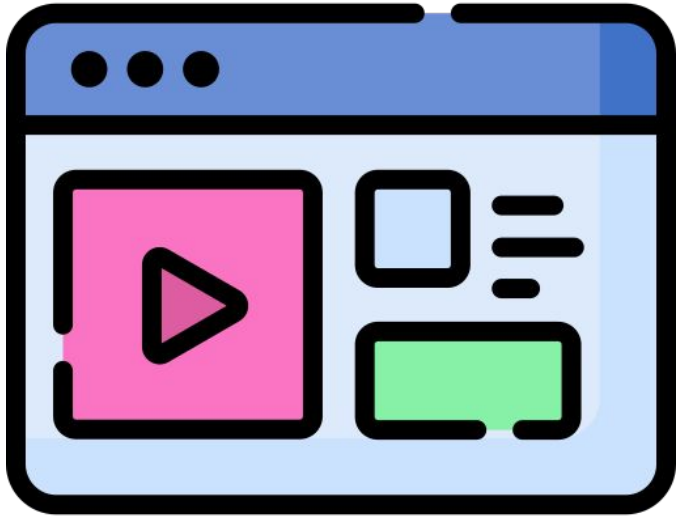
- Seguir las instrucciones de la actividad publicada en la UVE.



Próxima Clase

Desarrollo avanzado en React

- Técnicas avanzadas de React
- Optimización de rendimiento
- Mejoras en la experiencia del usuario



MUCHAS GRACIAS

ANDÉN Centro de Innovación y Emprendimientos Tecnológicos

SECRETARÍA DE EXTENSIÓN UNIVERSITARIA UTN - FRC SEU

UTN Facultad Regional Córdoba

Agencia CÓRDOBA JOVEN

CÓRDOBA entre todos