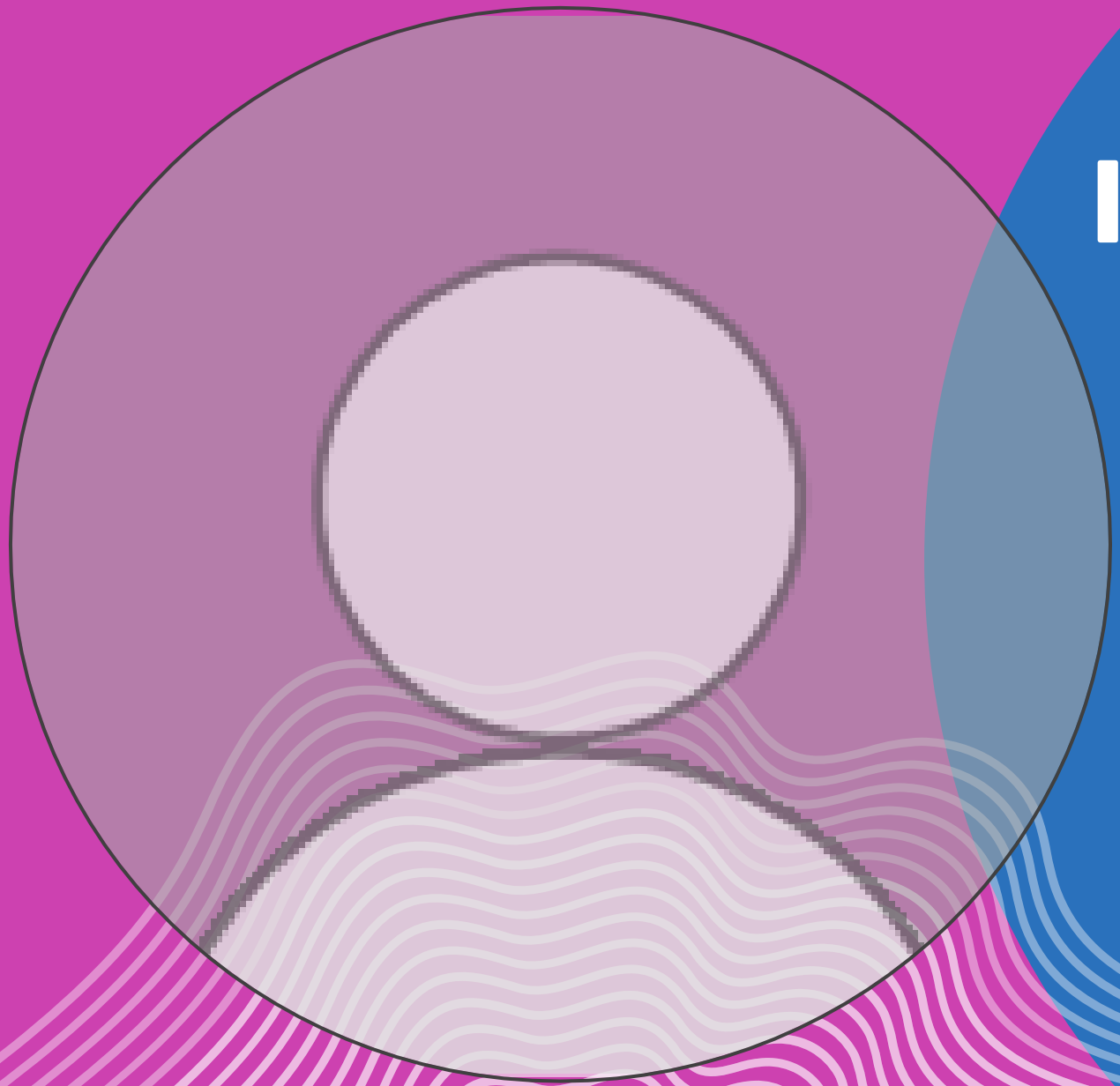
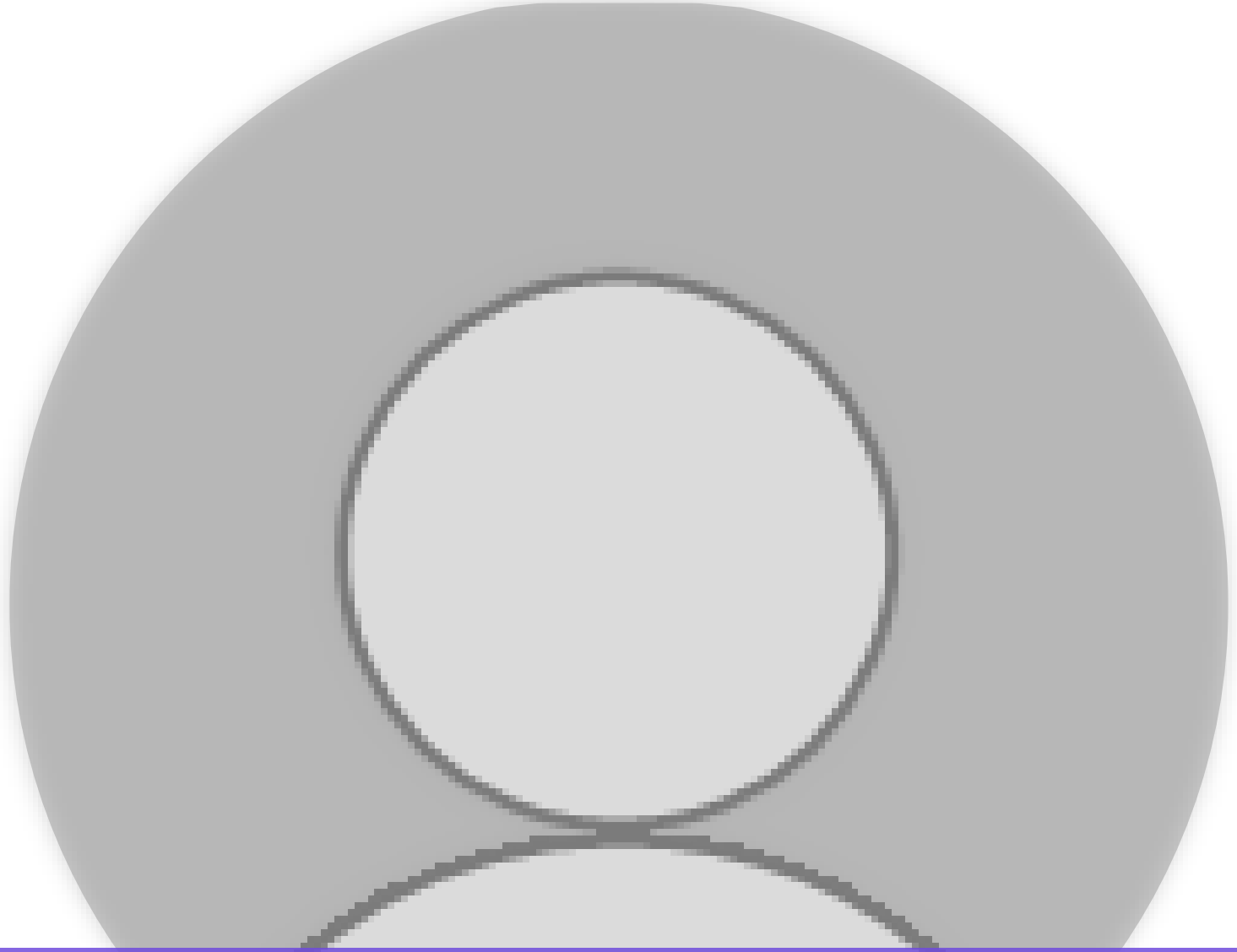
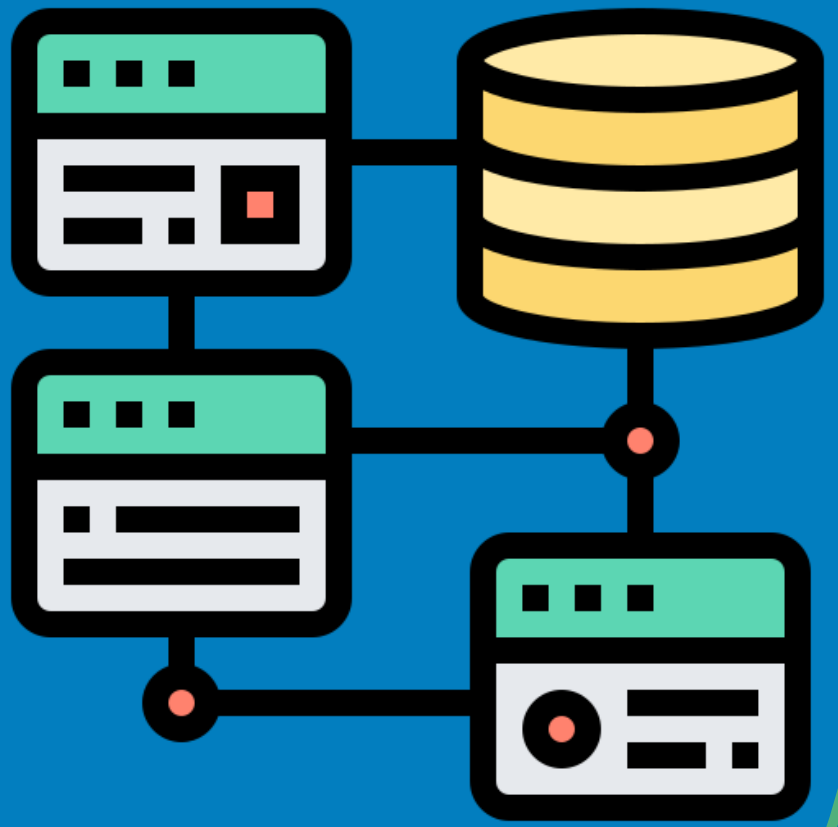


INPLEMENTANDO USO DE APIS EN PAGINAS WEB



Fabian Florian

ING. SISTEMAS - MENTOR BOOTCAMP DWFS



IMPLEMENTANDO APIS EN PAGINAS WEB

CRUD

Create-Read-Update-Delete

CODIGOS HTTP

1XX	Códigos informativos	El servidor ha recibido la petición y procederá con ella.
2XX	Códigos de éxito	El servidor ha recibido, entendido y procesado la solicitud correctamente.
3XX	Códigos de redirección	El servidor ha recibido la solicitud, pero hay una redirección a alguna otra parte (o, en raras ocasiones, alguna acción adicional que debe completarse).
4XX	Códigos de error de cliente	El servidor no puede encontrar (o alcanzar) la página o la web. Se trata de un error del lado de la web.
5XX	Códigos de error de servidor	El cliente ha realizado una solicitud válida, pero el servidor ha fallado al completarla.

CREANDO EL SCRIPT PARA UN CRUD

Crear un script que gestione un CRUD de "Personas" mediante interacción con una API usando fetch. Al cargar la página, se listan todas las personas, y se ofrece la posibilidad de crear, editar y eliminar entradas. También incluye la lógica para cambiar entre el modo de creación y edición, así como para actualizar la tabla dinámicamente después de cada operación.

IMPLEMENTANDO SCRIPT CON “FETCH”

El término **fetch** se refiere a una función de JavaScript utilizada para hacer solicitudes HTTP (como GET, POST, PUT y DELETE) a servidores remotos. Fue introducida en el estándar de JavaScript como una alternativa moderna a XMLHttpRequest y es ampliamente utilizada para interactuar con APIs y obtener o enviar datos a través de la web.

PORQUE UTILIZAR FETCH

¿Por qué se utiliza fetch en el código?
En el código que desarrollaremos, **fetch** permite realizar las siguientes operaciones CRUD (Crear, Leer, Actualizar y Eliminar) de forma asíncrona en una API:

1. **Obtener personas (fetchPersonas)**: Se realiza una solicitud GET para traer todas las personas y mostrarlas en la tabla.
2. **Crear persona (createPersona)**: Se usa un POST para enviar datos de una nueva persona al servidor.
3. **Editar persona (editPersona)**: Con un GET, se obtienen los datos de una persona específica, y luego con updatePersona, un PUT para actualizar sus datos.
4. **Eliminar persona (deletePersona)**: Mediante un DELETE, se elimina la persona identificada por su cédula.

VENTAJAS DE FETCH

Ventajas de utilizar fetch

•**Promesas**: fetch devuelve una Promesa, lo que permite manejar operaciones asíncronas usando .then() y .catch(), facilitando la lectura y manejo de errores.
•**Compatibilidad con JSON**: Es sencillo trabajar con datos JSON, ya que fetch permite convertir respuestas en JSON fácilmente, como se hace con .json().
•**Sintaxis limpia**: Comparado con XMLHttpRequest, fetch tiene una sintaxis más sencilla y directa, mejorando la legibilidad y mantenimiento del código.

QUE ES UNA PROMESA.?

En JavaScript, una **Promesa** es un objeto que representa un valor que puede estar disponible en algún momento en el futuro, después de que una operación asíncrona se complete. Una promesa puede tener uno de estos tres estados:

1. **Pendiente (Pending)**: La promesa aún no ha sido resuelta ni rechazada.
2. **Resuelta (Fulfilled)**: La operación se completó con éxito, y la promesa contiene el resultado.
3. **Rechazada (Rejected)**: La operación falló, y la promesa contiene un motivo de error.

Las promesas son fundamentales para manejar la programación asíncrona, ya que permiten continuar ejecutando el código mientras se espera la respuesta de una operación (como una solicitud HTTP) sin bloquear la ejecución del programa.

.THEN() Y .CATCH..?

¿Qué son .then() y .catch()?

Son métodos que se utilizan para trabajar con promesas y manejar los resultados o errores de operaciones asíncronas:

- **.then()**: Se usa para manejar el caso exitoso (cuando la promesa se resuelve). Este método recibe una función como argumento, la cual se ejecuta cuando la promesa cambia a estado "resuelta". La función recibe el valor que la promesa ha devuelto al resolverse, lo que permite manipular o utilizar ese valor.

```
fetch('/api/datos')
    .then(response => response.json()) // Convierte la respuesta a JSON cuando el fetch
    .then(data => console.log(data)) // Muestra los datos en consola
```

.THEN() Y .CATCH()..?

.catch(): Se utiliza para manejar el caso de error (cuando la promesa es rechazada). Este método recibe una función que se ejecuta si ocurre un error, y la función recibe el error como argumento. Así, .catch() permite manejar los errores de forma centralizada.

```
fetch('/api/datos')
    .then(response => response.json())
    .then(data => console.log(data))
    .catch(error => console.error('Error en la solicitud:', error)); // Maneja cualquier
```

PREGUNTAS Y RESPUESTAS