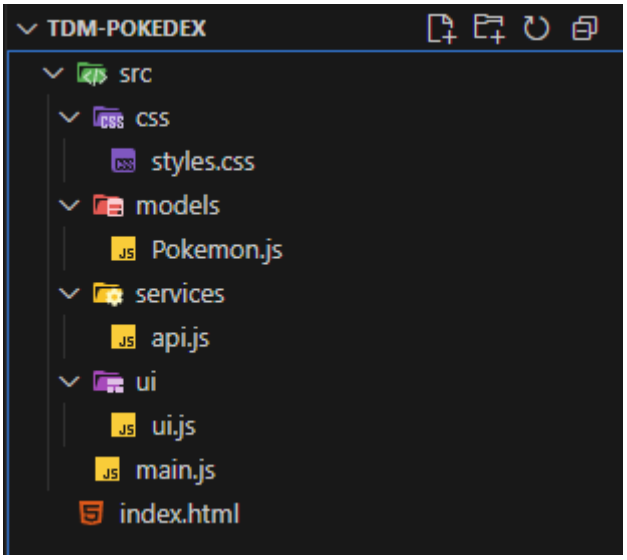
TALLER DE DISEÑO MULTIMEDIA
TALLER 3

JAVASCRIPT A FONDO

OBJETIVO: Aplicar los conceptos de JavaScript, manipulación del DOM, funciones flecha y consumo de APIs construyendo una mini Pokédex capaz de mostrar información de distintos Pokémon obtenida desde la PokéAPI.

1. Parámetros base

Primero, vamos a definir la estructura de nuestra pequeña aplicación.



Nota: Para los íconos de los archivos y carpetas, se utiliza la extensión “Material Icons”, pero no es necesaria para la realización de este taller. Ignora el archivo .gitattributes, pues este solo conecta el proyecto con Git.

Cada uno de los archivos juega un papel importante en la aplicación. A continuación se menciona cada uno de ellos:

- **src/css/styles.css:** Estilos globales de la aplicación.
- **src/models/Pokemon.js:** Exporta la CLASE base para crear los diferentes Pokémon. Esta solo contiene la información que va a tener cada Pokémon, como su ID, nombre, tipos, imagen, tamaño, peso, habilidades y estadísticas.
- **src/services/api.js:** Desde aquí vamos a hacer los métodos que se conecten a la API más adelante. Para este taller usaremos fetch() e importaremos Pokemon.js en este archivo para crear los Pokémon.
- **src/ui/ui.js:** Contiene la lógica para mostrar los Pokémon y la vista detalle. No se conecta a ningún otro archivo, solo exporta los métodos.

- **src/main.js:** Contiene la lógica principal de nuestra aplicación. Se conecta con la API para llamar el fetch y con la UI para mostrar los Pokémon. Aquí también colocaremos la lógica para navegar entre cada Pokémon.
- **index.html:** Contiene la estructura de nuestra aplicación.

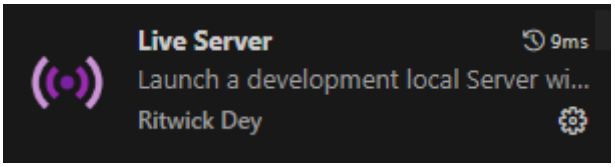
Para este taller, no hay limitaciones de diseño. Puedes estilizar tu aplicación como desees, pero ten en cuenta que debes traer y mostrar todos los datos solicitados.

Recordatorios Importantes

1. Para que puedas usar módulos (varios archivos JS que se exportan y se importan), debes declarar que el script, en el HTML, es de tipo módulo.

```
<script type="module" src="./src/main.js"></script>
```

2. Esta aplicación no funciona con elementos estáticos; no puedes abrir el archivo index.html con doble clic. Debes correr un servidor local, ya sea con la extensión de Live Server o crear tu propio servidor usando Node/Express.



3. Utiliza comentarios. En especial en la lógica (JS) y en los estilos (CSS).

2. Vista principal

En este punto, debes ser capaz de desarrollar tus propias páginas/aplicaciones usando HTML y CSS puro, por lo cual solo se darán los siguientes requerimientos que tu interfaz debe cumplir:

2.1. Estructura en HTML

Debe existir una card principal que muestre la información de un Pokémon.

La card debe incluir los siguientes elementos (cada uno con su id o class correspondiente):

- Imagen del Pokémon (img con id="pokemon-img")
- Nombre del Pokémon (h1 con id="pokemon-name")
- Número ID del Pokémon (p con id="pokemon-id")
- Tipos (contenedor con div class="types", dentro span por cada tipo)
- Botones de navegación (un botón prev y un botón next)

Toda la información se cargará de manera dinámica desde la API.

2.2. Estilos en CSS

Tipografía legible y clara (sans-serif preferiblemente).

Paleta de colores definida: al menos 3 colores consistentes (ejemplo: fondo, card y acentos).

La card debe contar con estilo propio:

- Bordes redondeados (border-radius mínimo de 8px).
- Sombra ligera (box-shadow) para dar relieve.
- Espaciado interno (padding) consistente.

Si hay cambios con hover o click, deben haber transiciones:

- Aplicar transition en hover de botones (ej. cambio de color, escala, sombra).
- Aplicar transition en hover de la card e imagen (ej. leve "zoom in").

Los botones deben ser usables y visibles:

- Contraste suficiente entre texto y fondo.
- Cambios de estado (:hover y :active).
- Cursor pointer al pasar por encima.

Los tipos Pokémon deben estar estilizados y su color debe cambiar según su tipo:

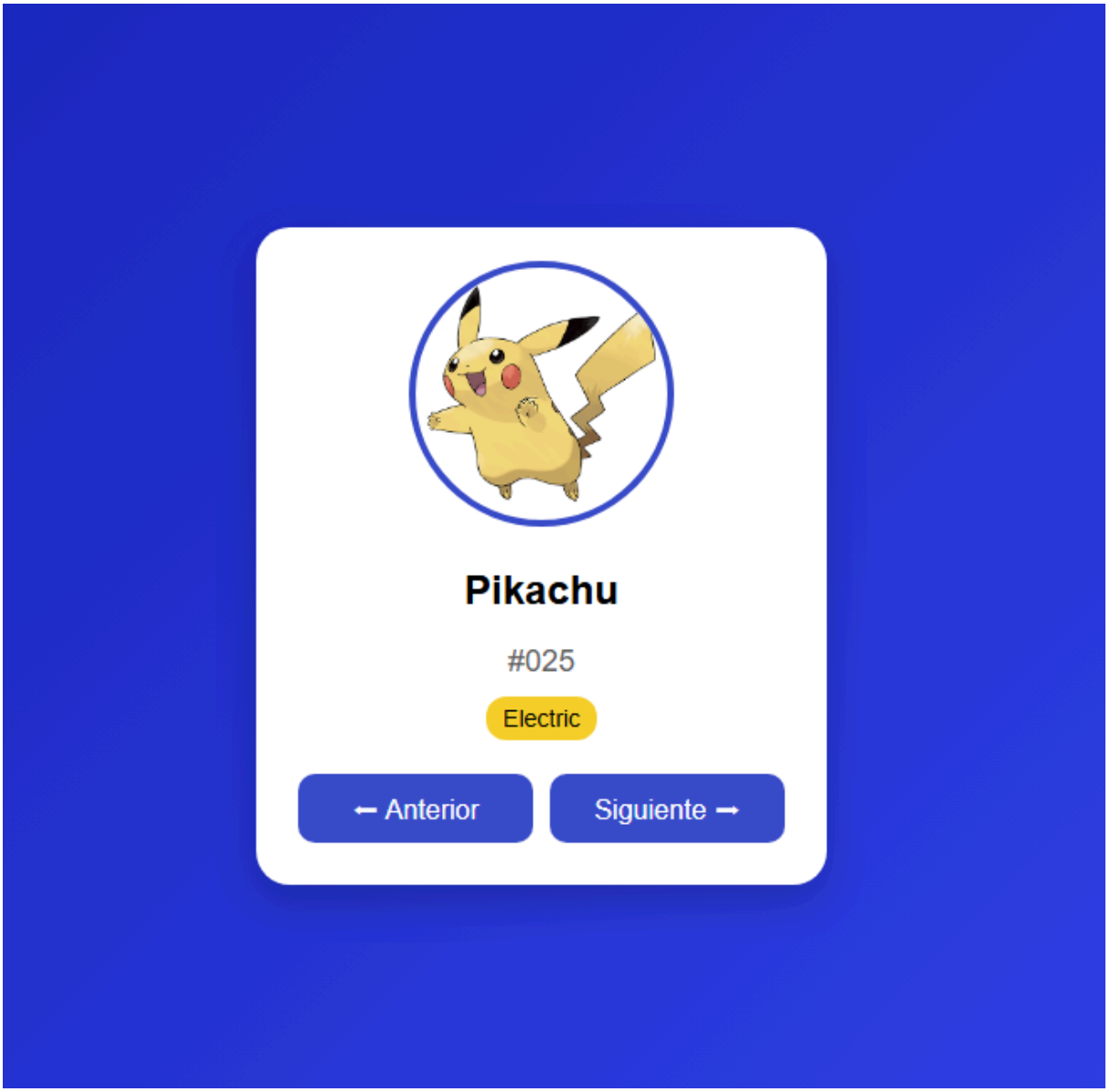
- Cada span debe tener una clase .type con estilos generales (ej: color de fondo, bordes redondeados, padding).
- Cada .type va acompañado de otra clase que determina el tipo de Pokémon que es (ej: .normal, .electric, .fire).
- Deben ser claramente legibles sobre cualquier color de card.

Nota: Para los estilos de los tipos, puedes utilizar un esquema de colores. En Pokémon existen 18 tipos de pokémon, por lo cual vamos a asignar un color a cada tipo y, en el caso donde el color sea muy claro, cambiamos el color de texto a negro.

```
.type.normal { background: #bdbdbd; color: #000; }
.type.electric { background: #f7d02c; color: #000; }
.type.fire { background: #ee8130; }
.type.ground { background: #833900; }
.type.rock { background: #e9d45f; color: #000; }
.type.water { background: #6390f0; }
.type.grass { background: #7ac74c; }
.type.bug { background: #729f3f; }
.type.flying { background: #3dc7ef; }
.type.ice { background: #9be9ff; color: #000; }
.type.poison { background: #9e0a72; }
.type.fairy { background: #ff87db; }
.type.fighting { background: #d33333; }
.type.psychic { background: #aa0080; }
.type.steel { background: #616161; }
.type.ghost { background: #313d70; }
.type.dark { background: #222222; }
.type.dragon { background: #3f23bb; }
```

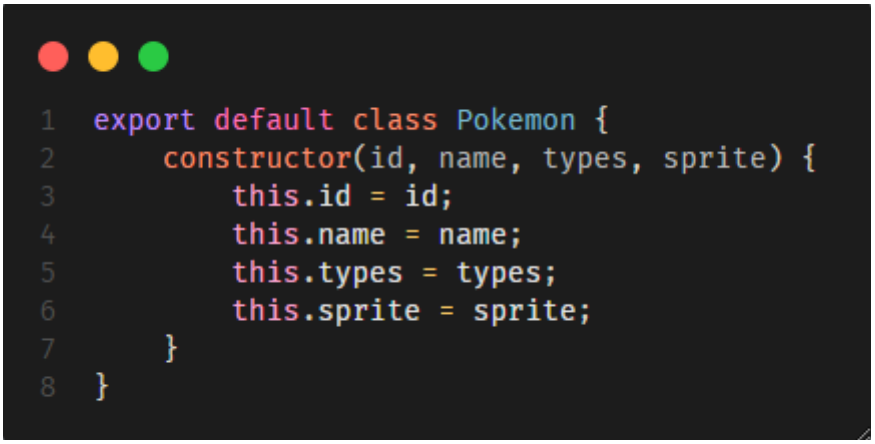
2.3. Referencia

Como referencia, puedes usar la siguiente vista, pero trata de generar un diseño propio que funcione para tu aplicación.



3. Pokemon.js

Como mencionamos anteriormente, Pokemon.js va a actuar como el modelo de nuestros Pokémon que lleguen desde la API, por lo cual debemos definir una estructura que nos sirva para “traducir” o serializar los datos.



```
export default class Pokemon {
    constructor(id, name, types, sprite) {
        this.id = id;
        this.name = name;
        this.types = types;
        this.sprite = sprite;
    }
}
```

En este caso, lo que hacemos es definir que cada Pokémon va a tener un ID, un nombre, una lista de tipos y un sprite (imagen). Posteriormente vamos a crecer esta clase, pero, de momento, estos son los datos que necesitamos de la API.

4. ui.js

Para la gestión de lo que se ve en pantalla, usamos ui.js. Este archivo se encarga de mostrar los datos del Pokémon en sus respectivos campos.

```
export function showPokemon(pokemon) {
    if (!pokemon) return;

    // Datos del Pokémon
    document.getElementById("pokemon-img").src = pokemon.sprite;
    document.getElementById("pokemon-name").textContent = capitalize(pokemon.name);
    document.getElementById("pokemon-id").textContent = "#" + pokemon.id.toString().padStart(3, "0");

    // Tipos
    const typesDiv = document.querySelector(".types");
    typesDiv.innerHTML = "";
    pokemon.types.forEach(t => {
        const span = document.createElement("span");
        span.classList.add("type", t);
        span.textContent = capitalize(t);
        typesDiv.appendChild(span);
    });
}

function capitalize(word) {
    return word.charAt(0).toUpperCase() + word.slice(1);
}

```

Podrás notar que, para obtener los elementos que ya creamos en el HTML, usamos .getElementById(), mientras que para crear los elementos de manera dinámica dentro del contenedor de tipos .types, usamos un ciclo y, en cada iteración, creamos un elemento usando .createElement y le añadimos tanto la clase .type como la clase correspondiente al tipo del Pokémon.

¿Qué es un foreach?: Es un bucle for simplificado, en el cual le decimos que recorra una lista, en este caso, la lista types que se encuentra dentro de pokemon.

¿Y capitalize()?: Es una función que creamos para que el nombre y el tipo se vea usando el formato gramatical correcto de mayúsculas. En este caso, agarra la letra en la posición 0 usando charAt(0) y luego la pone en mayúsculas usando toUpperCase(). Luego, utilizamos slice(1) para separar la palabra de la primera letra y unirla a esa nueva letra en mayúscula, esencialmente reemplazando la minúscula por la mayúscula.

5. api.js

Para la conexión con la API usaremos el archivo api.js. Lo más importante es conectarlo con la URL correcta. Como se mencionó al inicio de este taller, estaremos usando la API de PokéAPI, la cual puedes encontrar en el siguiente enlace:

PokéAPI: https://pokeapi.co/

Tal como menciona la documentación, usaremos la siguiente URL para conectarnos a la API:

URL: https://pokeapi.co/api/v2/pokemon/

Con esto en mente, vamos a desarrollar nuestro api.js.

```
import Pokemon from "../models/Pokemon.js";

const API_URL = "https://pokeapi.co/api/v2/pokemon/";

export async function fetchPokemon(id) {
    try {
        const res = await fetch(API_URL + id);
        if (!res.ok) throw new Error("No se encontró el Pokémon");
        const data = await res.json();

        // Extraer los tipos
        const types = data.types.map(t => t.type.name);

        // Crear instancia de Pokemon
        return new Pokemon(
            data.id,
            data.name,
            types,
            data.sprites.other["official-artwork"].front_default
        );
    } catch (error) {
        console.error(error);
        return null;
    }
}
```

6. main.js

Con todos los elementos, lo único que resta es unirlo todo en nuestro main.js, que actuará como el cerebro de toda la aplicación. Aquí queremos evitar lo más posible definir nuevos métodos y solo vamos a añadir la funcionalidad de carga de Pokémon usando showPokemon() de ui.js y trayendo los datos de fetchPokemon() de api.js.

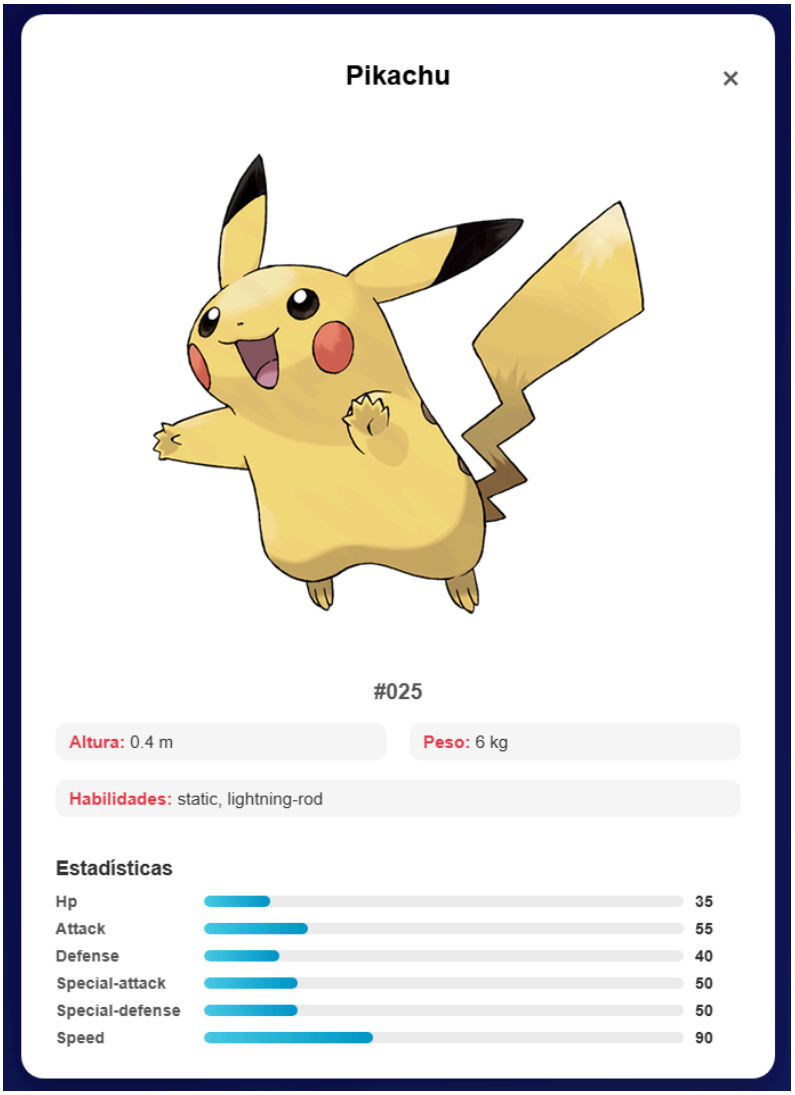
```
import { fetchPokemon } from "./services/api.js";
import { showPokemon } from "./ui/ui.js";

let current = 25; // 25 es Pikachu

async function loadPokemon(id) {
    const pokemon = await fetchPokemon(id);
    showPokemon(pokemon);
}

// Inicial
loadPokemon(current);

// Navegación
document.querySelector(".next").addEventListener("click", () => {
    current++;
    loadPokemon(current);
});

document.querySelector(".prev").addEventListener("click", () => {
    if (current > 1) current--;
    loadPokemon(current);
});

```

7. Reto: Modal con más información

El objetivo de este reto es simple: debes crear un modal que se abra al hacer clic sobre la imagen de cada Pokémon. Este modal va a contener la siguiente información:

- Nombre
- Imagen
- ID Pokémon
- Altura
- Peso
- Habilidades (Lista)
- Estadísticas (Lista)

Esta información debe mostrarse de la siguiente manera al pulsar la imagen de cualquier Pokémon:



Para el modal, puedes guiarte usando el siguiente HTML:

```html
<div id="pokemon-modal" class="modal hidden">
    <div class="modal-content">
        <span id="close-modal">&times;</span>
        <h2 id="modal-name"></h2>
        <img id="modal-img" src="" alt="pokemon grande">

        <p id="modal-id" class="pokemon-id"></p>

        <div class="modal-info">
            <p><strong>Altura:</strong> <span id="modal-height"></span> m</p>
            <p><strong>Peso:</strong> <span id="modal-weight"></span> kg</p>
            <p><strong>Habilidades:</strong> <span id="modal-abilities"></span></p>
        </div>

        <div id="modal-stats" class="modal-stats"></div>
    </div>
</div>
```

Por otro lado, dentro de api.js, puedes utilizar la siguiente estructura para extraer los datos de la habilidades y estadísticas de cada Pokémon:

```javascript
// Extraer las habilidades
const abilities = data.abilities.map(a => a.ability.name);

// Extraer las estadísticas
const stats = data.stats.map(s => ({
    stat: s.stat.name,
    base: s.base_stat
}));
```

Dentro de ui.js, específicamente dentro de la función de showPokemon(), añade la siguiente línea para agregar la funcionalidad de botón a la imagen:

```javascript
// Mostrar modal
document.querySelector(".pokemon-img").onclick = () => showModal(pokemon);
```

Para cumplir con este reto, debes asegurarte de realizar los siguientes pasos:

1. Asegúrate de que la clase Pokemon dentro de Pokemon.js tenga las nuevas variables que vamos a utilizar (tamaño, peso, habilidades y estadísticas).
2. De igual forma, asegúrate de que en api.js retornes correctamente los datos del Pokémon.
3. Crear una función showModal() en ui.js que se encargue de mostrar el modal y cargar los datos. Esta función toma como parámetro al Pokémon que obtienes en showPokemon().
4. Añade la funcionalidad de cierre y apertura del modal añadiendo o quitando la clase .hidden.

Tip: Para agregar de manera dinámica las estadísticas, fíjate mucho en cómo se cargan los tipos de manera dinámica dentro de showPokemon(). Solo que en vez de iterar sobre pokemon.types, debes iterar sobre pokemon.stats:

```
// Tipos
const typesDiv = document.querySelector(".types");
typesDiv.innerHTML = "";
pokemon.types.forEach(t => {
    // Iterar
});
```

Iteración de types ya realizada

```
// Estadísticas
const statsDiv = document.getElementById("modal-stats");
statsDiv.innerHTML = "<h3>Estadísticas</h3>";
pokemon.stats.forEach(s => {
    // Iterar
});
```

Iteración de estadísticas por realizar