Gabriel Hernández Collado

Índice

Resumen general

El objetivo es crear una aplicación de mánager de Pokémon, en la cual el usuario podrá abrir sobres y obtener Pokémon para coleccionar y hacerlos combatir contra los del resto de usuarios de la aplicación.

Resumen de las principales funcionalidades

- Registro e inicio de sesión
- Apertura de sobres
    - El usuario obtiene un sobre al día y dos al ganar un combate
- Colección
    - El usuario puede ver los Pokémon que ha obtenido abriendo sobres
- Combate
    - El usuario debe tener al menos 6 Pokémon para poder combatir
    - La aplicación enfrentará al usuario con otro al azar
    - El combate sucederá automáticamente
    - Si el usuario gana obtendrá dos sobres
- Perfil
    - El usuario podrá ver la información de su usuario
- Administrador
    - Si el usuario es administrador tendrá acceso a un panel donde podrá editar la información de los usuarios de la aplicación

Principales retos de la implementación

A continuación hablaré de las funcionalidades que me han supuesto un reto.

Sobres

Para la apertura de sobres he optado por usar un *dialog* que se abre cuando se hace click en el sobre que se quiere abrir. Al abrir el sobre, se redirige al archivo *.php* que se encarga de devolver los Pokémon que han tocado. Tuve problemas con esto, pues a veces se abría y otras no. Al final hice uso de la IA para que me ayudase y me recomendó almacenar el estado del *dialog* en *sessionStorage*:

```
if (dialogoSobre) {
    if (!sessionStorage.getItem('dialogoSobreCerrado')) {
        dialogoSobre.classList.add("blur-inverso");
        dialogoSobre.showModal();
        document.body.classList.add("blur");
        document.body.style.overflow = "hidden";
    }
    cerrarDialogoSobre?.addEventListener("click", () => {
        dialogoSobre.classList.remove("blur-inverso");
        dialogoSobre.style.display = "none";
        dialogoSobre.close();
        document.body.classList.remove("blur");
        document.body.style.overflow = "";
        sessionStorage.setItem('dialogoSobreCerrado', 'true');
    });
    dialogoSobre.addEventListener("cancel", (e) => {
        e.preventDefault();
        dialogoSobre.close();
        document.body.classList.remove("blur");
        document.body.style.overflow = "";
        sessionStorage.setItem('dialogoSobreCerrado', 'true');
    });
}
window.addEventListener('beforeunload', () => {
    sessionStorage.removeItem('dialogoSobreCerrado');
});
```

Esto por lo pronto funciona.

Colección

Además de mostrar los Pokémon que tiene el usuario, yo he querido agregar algunos detalles:

Pokémons no encontrados

En la colección se mostrarán todos los Pokémon, pero si el usuario no lo tiene se le agrega la clase *pokemon-not-found*, que aplicará un filtro quitándole todo el brillo a la imagen y subiendo el contraste al máximo.

```
if (isset($_SESSION['pokemons_usuario'])) {
                $notFoundClass = (in_array($pokemon['id'],
$_SESSION['pokemons_usuario']))
                    ? ''
                    : 'pokemon-not-found';
                $nombrePokemon = (in_array($pokemon['id'],
$_SESSION['pokemons_usuario'])) ? $pokemon['Name'] : "?";
            } else {
                $notFoundClass = 'pokemon-not-found';
            }
```

```
.pokemon-not-found {
    text-align: center;
    font-size: 1.2em;
    margin-top: 20px;
    filter:
        brightness(0)
        contrast(1000%)
        drop-shadow(0 0 8px rgba(0, 0, 0, 0.8));
}
.pokemon-card:has(.pokemon-img.pokemon-not-found) {
  pointer-events: none;
  cursor: default;
}
```

Dialog para mostrar los datos de los Pokémon

Por otra parte, muestro los datos de los Pokémon en un *dialog* que se abre al pinchar en el Pokémon. Los datos del Pokémon están en los *data-set* del *div* contenedor del Pokémon y JavaScript accede a ellos (no mostraré todo el código, pues se haría muy extenso).

```
echo "<div class='pokemon-card'
                    data-id='" . $pokemon['id'] . "'
                    data-name='" . $pokemon['Name'] . "'
                    data-type1='" . $pokemon['Type 1'] . "'
                    data-type2='" . $pokemon['Type 2'] . "'
                    data-legendary='" . $pokemon['Legendary'] . "'
                    data-icon='" . $pokemon['icon_path'] . "'
                    data-hp='" . $pokemon['HP'] . "'
                    data-attack='" . $pokemon['Attack'] . "'
                    data-defense='" . $pokemon['Defense'] . "'
                    data-specialattack='" . $pokemon['Sp. Atk'] . "'
                    data-specialdefense='" . $pokemon['Sp. Def'] . "'
                    data-speed='" . $pokemon['Speed'] . "'
                    data-total='" . $pokemon['Total'] . "'
                    data-generation='" . $pokemon['Generation'] . "'
```

```
                    data-description='" . $pokemon['Description'] . "'>";
            echo "<img class='pokemon-img $notFoundClass' src='" .
$pokemon['icon_path'] . "' alt='" . $nombrePokemon . "'>";
            echo "<h3 class='pokemon-nombre'>" . $nombrePokemon . "</h3>";
            echo "</div>";
```

```
const pokemonId = card.dataset.id;
        const pokemonName = card.dataset.name;
        const pokemonType1 = card.dataset.type1;
        const pokemonType2 = card.dataset.type2;
        const pokemonTotal = card.dataset.total;
        const pokemonHP = card.dataset.hp;
        const pokemonAttack = card.dataset.attack;
        const pokemonDefense = card.dataset.defense;
        const pokemonSpecialAttack = card.dataset.specialattack;
        const pokemonSpecialDefense = card.dataset.specialdefense;
        const pokemonSpeed = card.dataset.speed;
        const pokemonGeneration = card.dataset.generation;
        const pokemonLegendary = card.dataset.legendary;
        const pokemonImage = card.dataset.icon;
        const pokemonDescription = card.dataset.description;
```

Descripciones de los Pokémon

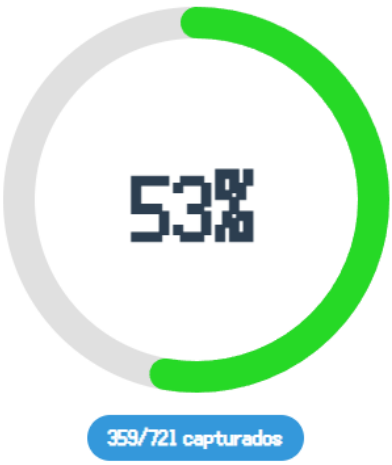
Intenté conseguir las descripciones de los 721 Pokémon con IA, pero no conseguía que me las diese todas, así que hice un script de Python que hiciese peticiones a la API de DeepSeek (pagas $2 y tienes un montón de peticiones) para conseguir todas las descripciones como updates de SQL:

```
UPDATE pokemon SET Description = 'Bulbasaur es un Pokémon pequeño y cuadrúpedo con una planta bulbosa en su lomo que crece desde su nacimiento. Este bulbos absorbe la luz solar para fortalecerse y liberar energía en combate.' WHERE id = 1;
UPDATE pokemon SET Description = 'Es un Pokémon tipo planta/veneno que evoluciona desde Bulbasaur. Tiene un bulbo más grande en su lomo que comienza a brotar cuando absorbe energía solar.' WHERE id = 2;
UPDATE pokemon SET Description = 'Venusaur es un Pokémon tipo Planta/Veneno de gran tamaño, caracterizado por su robusto cuerpo y una flor gigante en su lomo que libera un aroma calmante. Su poderosa constitución y su habilidad para realizar movimientos solares lo convierten en un formidable competidor en batalla.' WHERE id = 3;
```

Muchas no están completas o son erróneas, pero no iba a verificarlas todas, por supuesto. Podrás encontrar el script en el *.zip* del proyecto.

Porcentaje de colección completada

También he querido agregar barras de progreso para toda la colección, además de por generaciones.
Para el progreso general quise hacer una barra de progreso circular:

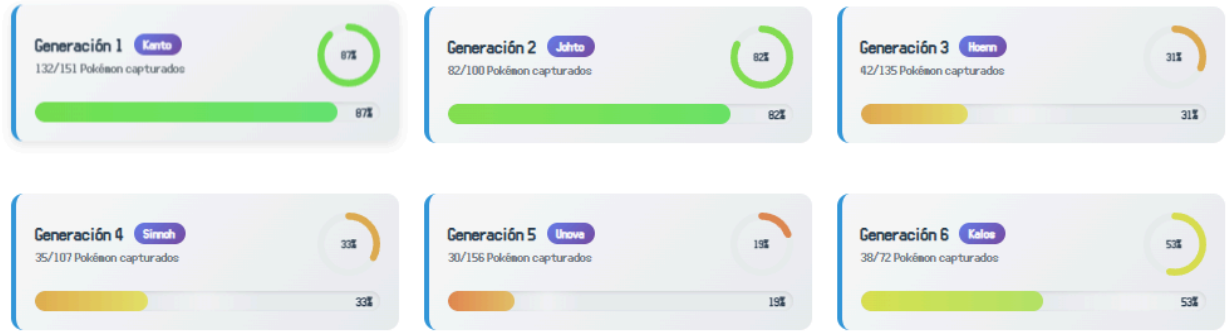


Esto lo he conseguido creando svg: uno para la barra y otro para la parte completada así:

```
<div class="circular-progress">
                            <svg viewBox="0 0 36 36">
                                <circle class="circular-bg"
cx="18"cy="18"r="16" />
                                <circle class="circular-progress-bar"
cx="18" cy="18" r="16" />
                            </svg>
                            <div class="circular-text">0%</div>
                        </div>
```

Y luego se maneja en el JavaScript (de nuevo no muestro el código).

Para las generaciones he optado por una barra horizontal, además de otra circular para cada generación.



No te voy a engañar, los estilos los ha hecho en su mayoría la IA.

El funcionamiento es sencillo. Tan solo hay que ver cuántos Pokémon tiene cada generación, generar un div por cada generación y calcular el porcentaje de Pokémon conseguidos de cada generación. Después tratas esos datos en JavaScript para modificar los atributos de las barras de progreso.

Filtrar por generaciones

Para hacer el filtro tan solo agrego un *eventListener* a cada ítem de generación. Cuando se hace click, toma el número de la generación del data-set del ítem y cambia el *display* de los Pokémon que no son de esa generación por *none*. Si se vuelve a hacer click en ese mismo ítem, vuelve a cambiar el display de todos los pokémon a *block*.

```
function initGenerationFilter() {
    const generacionItems = document.querySelectorAll('.generacion-item');
    const pokemonCards = document.querySelectorAll('.pokemon-card');
    let currentFilter = 'all';
    generacionItems.forEach(item => {
        item.addEventListener('click', () => {
            const selectedGen = item.dataset.gen;
            if (currentFilter === selectedGen) {
                showAllPokemon();
                removeActiveClass();
                currentFilter = 'all';
            } else {
                filterByGeneration(selectedGen);
                setActiveGeneration(item);
                currentFilter = selectedGen;
            }
        });
    });
    function filterByGeneration(generation) {
        pokemonCards.forEach(card => {
            const pokemonGen = card.dataset.generation;
            if (pokemonGen === generation) {
                card.style.display = 'block';
            } else {
                card.style.display = 'none';
            }
        });
    }
    function showAllPokemon() {
        pokemonCards.forEach(card => {
            card.style.display = 'block';
        });
    }
    function setActiveGeneration(activeItem) {
```

```
        generacionItems.forEach(item => {
            item.classList.remove('generation-active');
        });
        activeItem.classList.add('generation-active');
    }
    function removeActiveClass() {
        generacionItems.forEach(item => {
            item.classList.remove('generation-active');
        });
    }
}
```

Combate

Para el combate, quise hacer algo lo más parecido a los combates de los juegos originales de GameBoy.

Animaciones

He implementado las animaciones poniendo y quitando clases que contienen la animación a los elementos.
Lo más complicado es hacer que las animaciones sucedan a tiempo. El truco es utilizar *timeouts* y secuenciarlos correctamente. Por poner un ejemplo:

```
setTimeout(() => {
                    this.currentPlayerPokemon = this.playerPokemons[0];
                    this.log.push(`¡Ve!
¡${this.currentPlayerPokemon.name}!`);
                    this.updateUI();

                    this.showPokeballAnimation(true);

                    setTimeout(() => {
                        this.showPokemonAppearAnimation(true);

                        setTimeout(() => {
                            this.setAttackPriority();
                            this.isAnimating = false;
```

```
                        }, 1000);
                    }, 1200);
                }, 1800);
```

Anidando los *timeouts* se consigue que las animaciones se hagan sucesivamente y con los tiempos adecuados.

Scroll en el log

Algo que me ha dado problemas ha sido el scroll automático en el log, pero era por cómo lo tenía estructurado. Al final lo conseguí utilizando esta función:

```
scrollToBottom(element) {
        element.scrollTop = element.scrollHeight;

        requestAnimationFrame(() => {
            element.scrollTop = element.scrollHeight;
            const lastItem = element.lastElementChild;
            if (lastItem) {
                lastItem.scroll({
                    behavior: 'smooth',
                });
            }
        });
    }
```

Así conseguimos que haga scroll automático al último hijo de la lista de logs.

Administrador

A la hora de hacer el panel de administrador, para poder editar la información del usuario me he decantado por un atributo que desconocía: *contenteditable*.

```
editarBtns.forEach(btn => {
        btn.addEventListener('click', () => {
            const fila = btn.closest('tr');
            const celdasEditables =
fila.querySelectorAll('td:nth-child(n+4):nth-child(-n+9)');

            celdasEditables.forEach(td => {
                td.setAttribute('contenteditable', 'true');
                td.dataset.original = td.innerText.trim();
                td.classList.add('editable-activa');
            });

            btn.style.display = 'none';
            fila.querySelector('.btn-guardar-usuario').style.display =
'inline-block';
            fila.querySelector('.btn-cancelar-edicion').style.display =
'inline-block';
        });
    });
```

Cuando se hace click en el botón editar, se cambia el atributo a *true*, lo que hace que los campos afectados puedan ser editados.

Al hacer clic en guardar, se envía la información haciendo *fetch*:

```
guardarBtns.forEach(btn => {
        btn.addEventListener('click', () => {
            const fila = btn.closest('tr');
            const id = btn.dataset.id;
            const celdas = fila.querySelectorAll('td');

            const datos = {
                id: id,
                nombre: celdas[3].innerText.trim(),
                email: celdas[4].innerText.trim(),
                fecha_nacimiento: celdas[5].innerText.trim(),
                creado: celdas[6].innerText.trim(),
                ultimo_login: celdas[7].innerText.trim(),
                sobres: celdas[8].innerText.trim()
            };

            fetch('includes/admin/editar_usuario.inc.php', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(datos)
            })
                .then(res => res.text())
                .then(response => {
                    location.reload();
                });
        });
    });
```

Figuras

Diagrama de uso caso

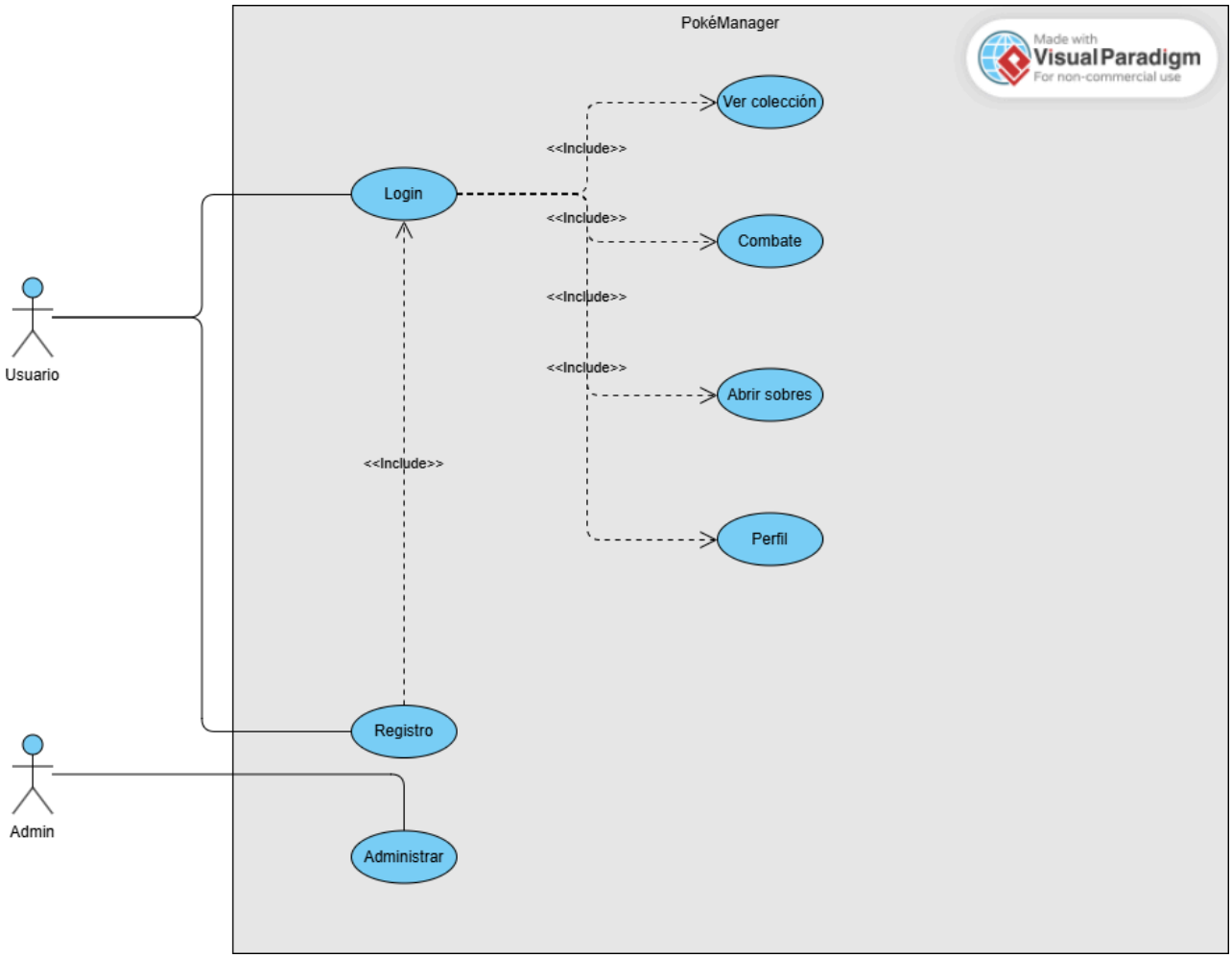
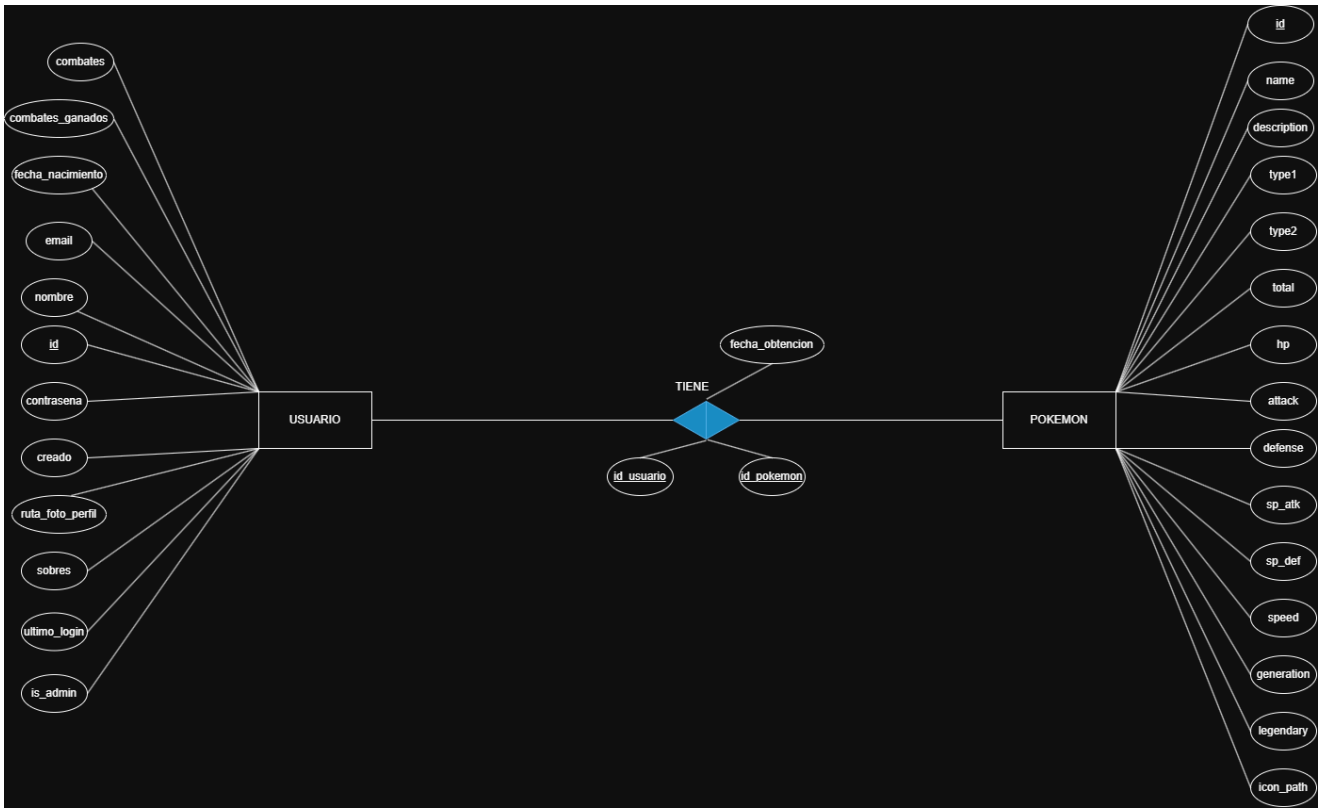


Diagrama de entidad-relación



Conclusión

Esta práctica es mucho más divertida si quieres ir más allá de lo que se plantea en el enunciado. Hemos tocado todos (o casi todos) los aspectos que se requieren para hacer una aplicación completa y funcional, desde el HTML básico pasando por los scripts y creación de APIs para manejar datos entre JavaScript y PHP y conexión a base de datos y modificación de los mismos.

Mi parte favorita es el combate, sin duda. Creo que me ha quedado bastante bien.
Si bien es cierto que estoy contento con el diseño general de la aplicación, me habría gustado pulirlo más.

En conclusión, me ha parecido una práctica divertida.