

# Boletín de Ejercicios - Sprint 3

## Descripción del ejercicio

Para asentar los conocimientos mostrados en la parte teórica, se tendrá que proceder a resolver los siguientes ejercicios/problemas en el orden correcto. Para ello, primero pasamos a explicar los criterios que debemos seguir para la resolución de los ejercicios/problemas.

## Fases de la resolución de problemas

1. **Análisis del problema:** Se debe indicar en el directorio específico de la asignatura el problema que se va a resolver de una forma adecuada, es decir, no debe contener ambigüedades, debe ser simple y autocontenido.
2. **Diseño de la propuesta de solución del problema:** Como todo aquel problema que se quiere resolver, es necesario realizar el diseño de la o las soluciones que se procederá a implementar en el siguiente paso. Para esto nos debemos ayudar de las herramientas para realizar esquemas gráficos (UML, Diagramas de flujos, etc...)
3. **Implementación del diseño propuesto:** En este punto ya se procederá a implementar todo el diseño establecido en el punto anterior.
4. **Pruebas de la resolución del problema:** Es indispensable el realizar pruebas para verificar la integridad y correcto funcionamiento de la implementación realizada, para ello simplemente compararemos si el comportamiento esperado del análisis del problema se ha implementado de forma adecuada.

# Ejercicio 1: Análisis y Transformación Avanzada de Datos

Descripción:

Dado un conjunto de datos en forma de array con información sobre estudiantes, sus calificaciones en diferentes asignaturas y detalles adicionales, implementa funciones para analizar, filtrar y transformar estos datos.

## Parte 1: Estudiantes Destacados por Asignatura

Crea una función que, dada una asignatura, retorne los 3 estudiantes con las mejores notas en esa asignatura.

## Parte 2: Asignatura con Menor Rendimiento

Diseña una función que identifique la asignatura en la que los estudiantes tienen, en promedio, la menor calificación.

## Parte 3: Mejora de Notas para Estudiantes con Beca

Escribe una función que aumente todas las notas de los estudiantes con beca en un 10% (sin superar el máximo de 10).

## Parte 4: Filtrado por Ciudad y Asignatura

Crea una función que, dada una ciudad y una asignatura, retorna la lista de estudiantes de esa ciudad ordenados descendientemente por la nota de la asignatura dada.

## Parte 5: Estudiantes Sin Beca por Ciudad

Escribe una función que, dada una ciudad, retorne la cantidad de estudiantes que no tienen beca en esa ciudad.

## Parte 6: Promedio de Edad de Estudiantes con Beca

Diseña una función que calcule el promedio de edad de los estudiantes que tienen beca.

## Parte 7: Mejores Estudiantes en Total

Crea una función que devuelva un array con los 2 estudiantes que tengan el mayor promedio general entre todas las asignaturas.

## Parte 8: Estudiantes con Todas las Materias Aprobadas

Diseña una función que retorne un array con los nombres de los estudiantes que hayan aprobado todas las materias (considera aprobado con una calificación mayor o igual a 5).

Ejemplo de Entrada:

```
const estudiantes = [  
  { nombre: "Juan", ciudad: "Madrid", beca: false, edad: 21, calificaciones: {  
    matematicas: 5, fisica: 7, historia: 6 } },  
  { nombre: "Ana", ciudad: "Barcelona", beca: true, edad: 20, calificaciones: {  
    matematicas: 9, fisica: 6, historia: 8 } },  
  { nombre: "Pedro", ciudad: "Madrid", beca: false, edad: 23, calificaciones: {  
    matematicas: 4, fisica: 5, historia: 7 } },  
  { nombre: "Maria", ciudad: "Sevilla", beca: true, edad: 19, calificaciones: {  
    matematicas: 8, fisica: 7, historia: 9 } },  
  { nombre: "Jose", ciudad: "Madrid", beca: false, edad: 22, calificaciones: {  
    matematicas: 6, fisica: 7, historia: 5 } },  
  { nombre: "Isabel", ciudad: "Valencia", beca: true, edad: 20, calificaciones: {  
    matematicas: 5, fisica: 8, historia: 7 } },  
  { nombre: "David", ciudad: "Bilbao", beca: false, edad: 24, calificaciones: {  
    matematicas: 7, fisica: 6, historia: 8 } },  
  { nombre: "Laura", ciudad: "Barcelona", beca: true, edad: 19, calificaciones: {  
    matematicas: 6, fisica: 8, historia: 7 } },  
  { nombre: "Miguel", ciudad: "Sevilla", beca: false, edad: 21, calificaciones: {  
    matematicas: 7, fisica: 7, historia: 8 } },  
  { nombre: "Sara", ciudad: "Madrid", beca: true, edad: 20, calificaciones: {  
    matematicas: 6, fisica: 5, historia: 9 } },  
  { nombre: "Daniela", ciudad: "Valencia", beca: false, edad: 22, calificaciones: {  
    matematicas: 8, fisica: 9, historia: 6 } },  
  { nombre: "Alberto", ciudad: "Bilbao", beca: true, edad: 23, calificaciones: {  
    matematicas: 5, fisica: 8, historia: 6 } },
```

```

    { nombre: "Gabriel", ciudad: "Sevilla", beca: false, edad: 19, calificaciones: {
matematicas: 8, fisica: 5, historia: 7 } }},

    { nombre: "Carmen", ciudad: "Barcelona", beca: true, edad: 24, calificaciones: {
matematicas: 9, fisica: 9, historia: 9 } }},

    { nombre: "Roberto", ciudad: "Madrid", beca: false, edad: 20, calificaciones: {
matematicas: 4, fisica: 5, historia: 5 } }},

    { nombre: "Carolina", ciudad: "Valencia", beca: true, edad: 22, calificaciones: {
matematicas: 5, fisica: 7, historia: 6 } }},

    { nombre: "Alejandro", ciudad: "Bilbao", beca: false, edad: 23, calificaciones: {
matematicas: 9, fisica: 8, historia: 8 } }},

    { nombre: "Lucia", ciudad: "Barcelona", beca: true, edad: 21, calificaciones: {
matematicas: 7, fisica: 7, historia: 7 } }},

    { nombre: "Ricardo", ciudad: "Sevilla", beca: false, edad: 19, calificaciones: {
matematicas: 6, fisica: 5, historia: 6 } }},

    { nombre: "Marina", ciudad: "Madrid", beca: true, edad: 20, calificaciones: {
matematicas: 5, fisica: 9, historia: 8 } }

];

```

Funciones a Implementar:

```

function estudiantesDestacadosPorAsignatura(estudiantes, asignatura) {

    // Devuelve un array con los 3 estudiantes con las mejores notas en la asignatura
    dada

}

function asignaturaMenorRendimiento(estudiantes) {

    // Identifica la asignatura con el promedio de calificación más bajo

}

function mejoraNotasBeca(estudiantes) {

    // Aumenta todas las notas de los estudiantes con beca en un 10% (máximo 10)

}

```

```
function filtrarPorCiudadYAsignatura(estudiantes, ciudad, asignatura) {  
  
    // Devuelve la lista de estudiantes de una ciudad ordenados descendientemente  
    por la nota de la asignatura dada  
  
}  
  
function estudiantesSinBecaPorCiudad(estudiantes, ciudad) {  
  
    // Devuelve la cantidad de estudiantes sin beca en la ciudad dada  
  
}  
  
function promedioEdadEstudiantesConBeca(estudiantes) {  
  
    // Devuelve el promedio de edad de los estudiantes con beca  
  
}  
  
function mejoresEstudiantes(estudiantes) {  
  
    // Devuelve un array con los 2 estudiantes con el mejor promedio general  
  
}  
  
function estudiantesAprobados(estudiantes) {  
  
    // Devuelve un array con los nombres de los estudiantes que hayan aprobado todas  
    las materias  
  
}
```

Punto a destacar para la prueba:

- Poned que las funciones se aplican cuando se pulsan botones diferentes, para que así podáis realizar una única prueba para probar cada una de las funciones en cada paso, es decir, paso 1, hacer click en el botón de la función “estudiantesDestacadosPorAsignatura()”, resultado esperado, lo que devuelve ese método y así sucesivamente con el resto de funciones.

## Pruebas mínima para cada función

### Parte 1: Estudiantes Destacados por Asignatura

```
console.log(estudiantesDestacadosPorAsignatura(estudiantes, 'matematicas'));  
  
// Esperado: [{ nombre: "Carmen", ... }, { nombre: "Alejandro", ... }, { nombre: "Ana", ... }]
```

### Parte 2: Asignatura con Menor Rendimiento

```
console.log(asignaturaMenorRendimiento(estudiantes));  
  
// Esperado: 'matematicas' (promedio más bajo entre todas las asignaturas).
```

### Parte 3: Mejora de Notas para Estudiantes con Beca

```
console.log(mejoraNotasBeca(estudiantes));  
  
// Esperado: Las notas de estudiantes con beca aumentan un 10%, ejemplo: { nombre: "Ana",  
matematicas: 9.9, fisica: 6.6, historia: 8.8 } sin superar 10.
```

### Parte 4: Filtrado por Ciudad y Asignatura

```
console.log(filtrarPorCiudadYAsignatura(estudiantes, 'Madrid', 'fisica'));  
  
// Esperado: [{ nombre: "Juan", ... }, { nombre: "Jose", ... }, { nombre: "Sara", ... }] ordenados de  
mayor a menor en 'fisica'.
```

### Parte 5: Estudiantes Sin Beca por Ciudad

```
console.log(estudiantesSinBecaPorCiudad(estudiantes, 'Madrid'));  
  
// Esperado: 4 (Juan, Pedro, Jose y Roberto no tienen beca en Madrid).
```

### Parte 6: Promedio de Edad de Estudiantes con Beca

```
console.log(promedioEdadEstudiantesConBeca(estudiantes));  
  
// Esperado: 21 (promedio de la edad de los estudiantes con beca).
```

### Parte 7: Mejores Estudiantes en Total

```
console.log(mejoresEstudiantes(estudiantes));  
  
// Esperado: [{ nombre: "Carmen", ... }, { nombre: "Alejandro", ... }] (mejores promedios  
generales).
```

## Parte 8: Estudiantes con Todas las Materias Aprobadas

```
console.log(estudiantesAprobados(estudiantes));
```

```
// Esperado: ["Juan", "Ana", "Maria", "Jose", "Isabel", "David", "Laura", "Miguel", "Sara", "Daniela",  
"Alberto", "Gabriel", "Carmen", "Carolina", "Alejandro", "Lucia", "Marina"] (estudiantes que  
aprobaron todas las materias).
```

## Ejercicio 2: Uso de PokeAPI

### Parte 1: Información Básica del Pokémon

- Realizar una petición a la PokeAPI para obtener información básica de un Pokémon por su nombre.
- Mostrar el nombre, id, tipos, y una imagen del Pokémon.
- Gestionar errores de manera adecuada si el Pokémon no existe.

#### PRUEBAS

```
function obtenerInfoBasicaPokemon(nombrePokemon) {  
  
    // Lógica para realizar la petición a la PokeAPI y gestionar la respuesta  
  
    }  
  
// Prueba 1: Obtener información de "pikachu"  
  
    obtenerInfoBasicaPokemon("pikachu");  
  
// Esperado: {nombre: "Pikachu", id: 25, tipos: ["Electric"], imagen: "URL de imagen de Pikachu"}  
  
// Prueba 2: Obtener información de "charizard"  
  
    obtenerInfoBasicaPokemon("charizard");  
  
// Esperado: {nombre: "Charizard", id: 6, tipos: ["Fire", "Flying"], imagen: "URL de imagen de Charizard"}  
  
// Prueba 3: Manejo de error al buscar "fakepokemon"  
  
    obtenerInfoBasicaPokemon("fakepokemon");  
  
// Esperado: Error adecuado mostrando "Pokémon no encontrado".
```



## Parte 2: Comparativa de Pokémon

- Obtener datos de dos Pokémon elegidos por el usuario.
- Comparar sus estadísticas base (stats) y determinar cuál de ellos tiene mejores estadísticas generales.
- Presentar los resultados en una tabla comparativa de fácil lectura.

### PRUEBAS

```
function compararPokemon(pokemon1, pokemon2) {  
  
    // Lógica para realizar la comparación de las estadísticas de los dos Pokémon  
  
}  
  
// Prueba 1: Comparar "bulbasaur" y "squirtle"  
  
compararPokemon("bulbasaur", "squirtle");  
  
// Esperado: Tabla comparativa con las estadísticas de ambos Pokémon, mostrando quién tiene  
mejores estadísticas generales.  
  
// Prueba 2: Comparar "gengar" y "alakazam"  
  
compararPokemon("gengar", "alakazam");  
  
// Esperado: Tabla comparativa con las estadísticas de ambos Pokémon, indicando el Pokémon con  
mejores estadísticas.
```

### Parte 3: Evoluciones y Habilidades

- Dado un Pokémon específico, buscar su cadena de evolución completa.
- Listar cada una de las formas evolutivas y sus habilidades.
- Incluir un botón que permita al usuario ver más detalles de cualquier habilidad (usando un modal o una nueva vista).
- **Pistas:** `pokemon/charmander -> pokemon-species/4/ -> evolution-chain/2/`

#### PRUEBAS

```
function obtenerCadenaEvolutiva(pokemon) {
```

```
    // Lógica para obtener la cadena evolutiva y habilidades del Pokémon
```

```
}
```

```
// Prueba 1: Obtener cadena evolutiva de "charmander"
```

```
obtenerCadenaEvolutiva("charmander");
```

```
// Esperado: Lista de evoluciones { Charmander, Charmeleon, Charizard }, junto con sus habilidades.
```

```
// Prueba 2: Manejar Pokémon sin cadena evolutiva como "tauros"
```

```
obtenerCadenaEvolutiva("tauros");
```

```
// Esperado: Mostrar mensaje indicando que no tiene cadena evolutiva.
```

## Ejercicio 3: Implementación de CRUD con Modal para "Add New Member" en una Guild

### Contexto:

Vas a implementar una funcionalidad CRUD (Create, Read, Update, Delete) para gestionar miembros de una guild en un sistema de administración. Debes crear una interfaz que permita visualizar los miembros actuales en una tabla, añadir nuevos miembros mediante un modal y editar o eliminar miembros existentes.

### Requerimientos Funcionales:

#### 1. Visualización de Miembros:

- Crea una tabla que muestre todos los miembros de la guild.
- La tabla debe incluir las siguientes columnas:
  - User ID
  - Username
  - Level
  - Item Level
  - Character Role (enum: TANK, HEALER, DAMAGE, SUPPORT)
  - Guild Role (enum: LIDER, GERENTE SENIOR, GERENTE, GERENTE A2, ALPHA 2, MEMBER)
- Cada fila debe tener botones de **Editar** y **Eliminar**.

#### 2. Añadir Miembro (Create):

- Implementa un botón **"Add New Member"** que abra un **modal**.
- El modal debe contener un formulario con los siguientes campos obligatorios:
  - `user_id` (string)
  - `username` (string)
  - `level` (integer)
  - `ilvl` (integer)
  - `character_role` (enum: TANK, HEALER, DAMAGE, SUPPORT)
  - `guild_role` (enum: LIDER, GERENTE SENIOR, GERENTE, GERENTE A2, ALPHA 2, MEMBER)
  - `main_archetype` (enum: BARD, CLERIC, FIGHTER, MAGE, RANGER, ROGUE, SUMMONER, TANK)
  - `secondary_archetype` (enum: BARD, CLERIC, FIGHTER, MAGE, RANGER, ROGUE, SUMMONER, TANK)

- `grandmaster_profession_one` (enum: FISHING, HERBALISM, HUNTING, LUMBERJACKING, MINING, ALCHEMY, ANIMALHUSBANDRY, COOKING, FARMING, LUMBERMILLING, METALWORKING, STONECUTTING, TANNING, WEAVING, ARCANENGINEERING, ARMORSMITHING, CARPENTRY, JEWELCUTTING, LEATHERWORKING, SCRIBE, TAILORING, WEAPONSMITHING)
- `grandmaster_profession_two` (enum: FISHING, HERBALISM, HUNTING, LUMBERJACKING, MINING, ALCHEMY, ANIMALHUSBANDRY, COOKING, FARMING, LUMBERMILLING, METALWORKING, STONECUTTING, TANNING, WEAVING, ARCANENGINEERING, ARMORSMITHING, CARPENTRY, JEWELCUTTING, LEATHERWORKING, SCRIBE, TAILORING, WEAPONSMITHING)
- `email` (string)
- `notify_email` (boolean)

- Los datos ingresados deben ser enviados a la API para añadir el nuevo miembro utilizando el endpoint **indicado en el YML**.

### 3. Editar Miembro (Update):

- Implementa la funcionalidad de editar un miembro utilizando el botón "**Edit**" en cada fila de la tabla.
- Al hacer click en **Editar**, se debe abrir el mismo modal que para añadir un nuevo miembro, pero precargando los datos del miembro seleccionado.
- Envía los cambios utilizando el endpoint **indicado en el YML** para actualizar la información.

### 4. Eliminar Miembro (Delete):

- Implementa la funcionalidad de eliminar un miembro utilizando el botón "**Delete**" en cada fila de la tabla.
- Al hacer click en **Eliminar**, debe realizarse una confirmación.
- Envía la solicitud al endpoint **indicado en el YML**

### 5. Validaciones:

- Todos los campos en el modal deben ser obligatorios.
- El campo `email` debe ser validado con un formato de correo electrónico.
- `user_id` debe ser único. Si el `user_id` ya existe en la guild, no se debe permitir su duplicación.

## Pruebas a Realizar:

### Prueba 1: Visualización de Miembros:

- Abre la pantalla de gestión de miembros.
- Verifica que la tabla lista correctamente a los miembros con todos los campos mencionados.

### Prueba 2: Añadir Nuevo Miembro:

- Haz click en el botón **"Add New Member"**.
- Llena el formulario con datos válidos y haz click en **"Add Member"**.
- Verifica que el nuevo miembro se muestra en la tabla y que los datos se han guardado correctamente en la API.

### Prueba 3: Valoraciones erróneas de Añadir Miembro:

- Intenta añadir un miembro dejando el campo **email** vacío o con un formato no válido. Debe mostrar un error.
- Intenta añadir un miembro con un **user\_id** que ya existe. Debe mostrar un error indicando que el **user\_id** es único.

### Prueba 4: Editar Miembro:

- Haz click en **"Edit"** para un miembro específico.
- Modifica los datos del miembro (por ejemplo, cambia el **level** y el **lvl**).
- Guarda los cambios y verifica que se actualizan en la tabla y en la API.

### Prueba 5: Eliminar Miembro:

- Haz click en **"Delete"** para un miembro específico.
- Confirma la acción de eliminación.
- Verifica que el miembro se ha eliminado de la tabla y de la API.

### Prueba 6: Comportamiento del Modal:

- Abre el modal para **añadir** un miembro y cierra el modal sin realizar ninguna acción. Verifica que los campos del formulario se limpian correctamente al reabrir el modal.
- Haz click en **"Edit"** para un miembro, edita algunos campos, y cierra el modal sin guardar. Verifica que los cambios no se guardan y los datos del miembro no se han modificado.

## Ejercicio 4: Implementación de un Formulario de Creación de Party en "Party Finder"

### Contexto:

En este ejercicio, vas a implementar un formulario que permita la creación de una Party en el sistema "**Party Finder**". El formulario debe permitir ingresar los detalles de la party y crearla utilizando la API correspondiente.

### Requerimientos Funcionales:

#### 1. Formulario de Creación de Party:

- Debe tener los siguientes campos:
  - **Party Size** (enum): Tamaño de la party (3, 5, 8).
  - **Creator ID** (string): ID del miembro creador de la party que debe existir en la tabla guildmembers.
  - **Level Cap** (integer): Nivel mínimo permitido en la party.
  - **Item Level Cap** (integer): Nivel de ítem mínimo permitido en la party.
  - **Party Role** (enum): Rol del creador en la party (TANK, HEALER, DAMAGE, SUPPORT).
  - **Planned Start** (string): Fecha y hora planeadas para el inicio de la party en formato **DD/MM/YYYY\_HH:mm**.
- El formulario debe validar que:
  - Todos los campos obligatorios están completos.
  - El formato de la fecha/hora sea correcto (**DD/MM/YYYY\_HH:mm**).

#### 2. Interacciones con la API:

- Al enviar el formulario, se debe realizar una solicitud **indicado en el YML** para crear la party.
- Si la solicitud es exitosa, la party debe aparecer listada en la interfaz de usuario.

#### 3. Validaciones:

- El campo **Planned Start** debe ser una fecha y hora futura.
- Los campos **Level Cap** y **Item Level Cap** deben ser números enteros positivos.
- El **Creator ID** debe ser un identificador válido.

## Pruebas a Realizar:

### Prueba 1: Crear una Party Correctamente:

- Llena todos los campos del formulario con datos válidos, incluyendo un **Planned Start** con una fecha y hora futuras. El campo Creator ID debe existir en la tabla guildmembers.
- Envía el formulario.
- Verifica que la party se ha creado correctamente y que aparece en la lista de parties.

### Prueba 2: Validación de Campos Vacíos:

- Intenta enviar el formulario dejando algunos de los campos obligatorios vacíos.
- **Resultado esperado:** El sistema debe evitar la creación de la party y mostrar un mensaje de error indicando que los campos son obligatorios.

### Prueba 3: Validación de Fecha Inválida:

- Intenta ingresar una fecha en el pasado en el campo **Planned Start**.
- **Resultado esperado:** El sistema debe evitar la creación de la party y mostrar un mensaje de error indicando que la fecha y hora deben ser futuras.

### Prueba 4: Validación de Números Negativos en Caps:

- Ingresa números negativos o cero en los campos **Level Cap** o **Item Level Cap**.
- **Resultado esperado:** El sistema debe evitar la creación de la party y mostrar un mensaje de error indicando que los valores deben ser positivos.

### Prueba 5: Comportamiento del Formulario:

- Abre el formulario para crear una party, ingresa datos en algunos campos y cierra el formulario sin enviarlo.
- Reabre el formulario y verifica que los campos están limpios.
- **Resultado esperado:** Los campos deben estar vacíos cuando se vuelve a abrir el formulario.

# Ejercicio 5: Gestión Avanzada de Parties con Añadir/Remover Miembros

## Contexto:

Este ejercicio extiende la funcionalidad del sistema **"Party Finder"** para permitir la gestión avanzada de parties. Implementarás una pantalla que permita no solo crear una party, sino también visualizar los detalles de las parties creadas, añadir nuevos miembros, y remover miembros de las parties.

## Requerimientos Funcionales:

### 1. Visualización de Parties Creadas:

- Se debe mostrar una lista de todas las parties creadas. La lista debe incluir los siguientes campos:
  - **Party ID:** Identificador único de la party.
  - **Creator ID:** ID del creador de la party que debe existir en la tabla `guildmembers`.
  - **Planned Start:** Fecha y hora planeada para el inicio de la party.
  - **Level Cap:** Nivel máximo permitido.
  - **Item Level Cap:** Nivel máximo de ítem permitido.
  - **Número de miembros:** Número actual de miembros en la party.

### 2. Añadir Miembros a una Party:

- Cada party debe tener un botón **"Add Member"**.
- Al hacer clic en **"Add Member"**, se debe abrir un modal que permita agregar un miembro a la party.
- El modal debe contener los siguientes campos:
  - **User ID:** ID del usuario que se quiere añadir.
  - **Party Role** (enum: TANK, HEALER, DAMAGE, SUPPORT): Rol que se le asignará al miembro.
- Debe validarse que:
  - El miembro no está ya en la party.
  - El rol que se le asigna esté disponible (según el tamaño de la party).
- La funcionalidad debe interactuar con el endpoint **indicado en el YML** para añadir el miembro.

### 3. Remover Miembros de una Party:

- Cada party debe tener un botón **"Remove Member"** para cada miembro listado.
- Al hacer clic en **"Remove Member"**, se debe enviar una solicitud para remover al miembro de la party utilizando el endpoint **indicado en el YML**.
- El sistema debe verificar que el miembro existe en la party antes de removerlo.

### 4. Validaciones:

- No se deben añadir más miembros de los permitidos por el **Party Size**.
- Si un rol (TANK, HEALER, DAMAGE, SUPPORT) ya está cubierto, no se debe permitir añadir otro miembro con el mismo rol.
- No se debe permitir remover al creador de la party.



## Pruebas a Realizar:

### Prueba 1: Visualización de Parties Creadas:

- Abre la pantalla de gestión de parties y verifica que todas las parties creadas se muestran correctamente con los datos necesarios.
- **Resultado esperado:** Todas las parties deben listarse con su información básica (ID, creador, fecha de inicio, caps, número de miembros).

### Prueba 2: Añadir Miembro a una Party:

- Selecciona una party y añade un nuevo miembro utilizando el modal.
- Llena los campos obligatorios y asegúrate de que el rol asignado esté disponible.
- **Resultado esperado:** El miembro se añade exitosamente a la party y el número de miembros se actualiza.
- El miembro que se añade debe existir en la base de datos teniendo en cuenta que debería salir un error de no existir.

### Prueba 3: Validación de Roles al Añadir Miembro:

- Intenta añadir un miembro con un rol que ya está cubierto en la party.
- **Resultado esperado:** El sistema debe evitar la adición del miembro y mostrar un mensaje de error indicando que el rol ya está ocupado.

### Prueba 4: Remover Miembro de una Party:

- Selecciona una party y remueve a un miembro existente utilizando el botón "**Remove Member**".
- **Resultado esperado:** El miembro es removido de la party y el número de miembros se actualiza.

### Prueba 5: Intentar Añadir Más Miembros del Límite de la Party:

- Intenta añadir más miembros de los permitidos por el **Party Size**.
- **Resultado esperado:** El sistema debe evitar la adición de nuevos miembros y mostrar un mensaje de error indicando que la party está llena.


# Formato de entrega

Todos estos ejercicios se deberán entregar en el formato establecido en clase o tablón de classroom, respetando las horas de entrega de cada uno de ellos indicados en la tarea de classroom.

¿Qué y cómo se entrega?

- Hay que realizar cada apartado de ejercicios en HTML diferentes y subirlo al repositorio a la carpeta SPRINT 3
- Un README.md para el ejercicio 1, un README.md para el ejercicio 2 y un README.md para los ejercicios 3, 4 y 5 juntos. Son tres README.md en total los que hay que documentar.
- Hay que realizar un vídeo en formato .gif para cada ejercicio y cada prueba en el que se interactúe de manera dinámica con la web y adjuntarlo en el README.md del repositorio GIT en la carpeta del sprint correspondiente.
- Hay que realizar una captura de pantalla de aquellos ejercicios que sean estáticos y adjuntarlos en el README.md del repositorio GIT en la carpeta del sprint correspondiente.
- Ejemplo de un README a continuación:

## ANGULAR

BOLETÍN A1 AVANZADO 

### Análisis del problema.

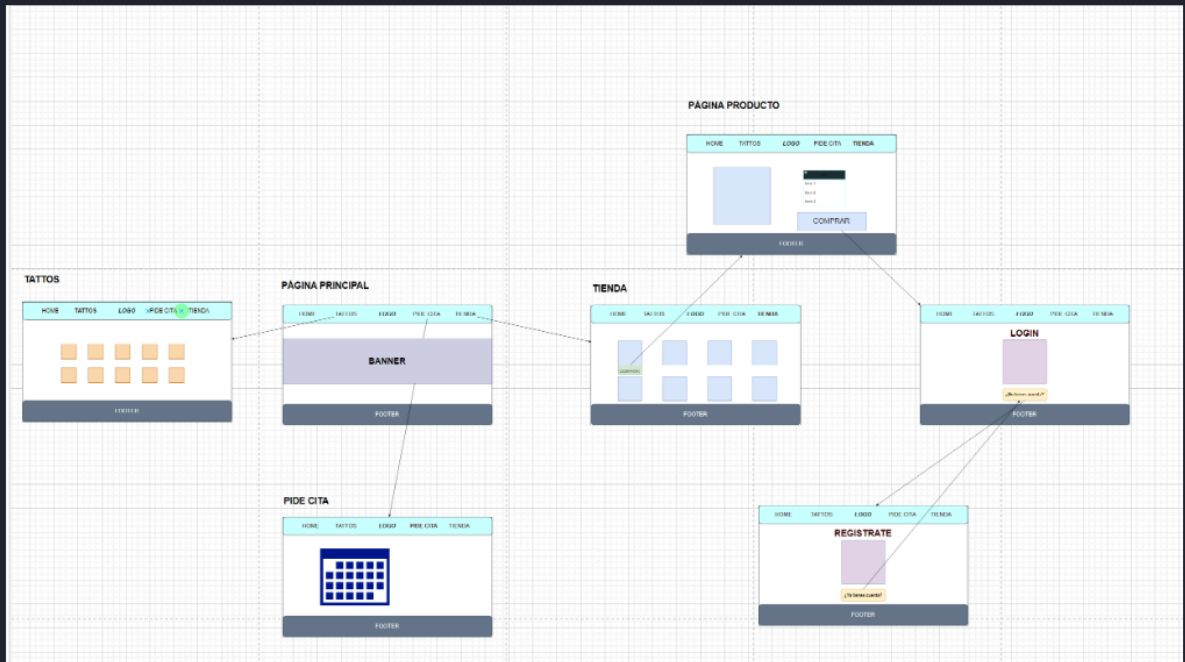
-> Se requiere realizar los siguientes ejercicios:

1. Seguir los pasos de instalación del vídeo suministrado "Instalación Angular.mp4".
2. Crear los componentes que se indican en la siguiente imagen con la ayuda del vídeo "Componentes Angular.mp4".
3. Añadir las verificaciones de cada input-botón mostrando un error en caso de que no se cumplan:
  - Email: Debe ser un email con su @ y otras validaciones. Buscar en internet. No debe estar vacío.
  - Password: No deben mostrarse los caracteres, sólo los puntos. No debe estar vacío.
  - Last Name: Puede estar vacío. Es un input normal.
  - First Name: No puede estar vacío. Es un input normal.
  - Botón con texto: Puede estar habilitado o deshabilitado.
  - Hiperenlace: Mostrará diferentes textos.
4. Añadir un componente que se pueda utilizar para el proyecto individual.
5. Realizar un boceto de cada una de las pantallas que se vayan a utilizar en el proyecto individual con la finalidad de reutilizar este trabajo y así reducir el esfuerzo total.

Primero instalaremos Angular con las indicaciones del profesor y con ayuda del vídeo. Luego creamos los componentes (Login, registro y footer).

## Implementación de la solución. [🔗](#)

En este apartado vamos a ponernos a implementar todos los apartados anteriores, vamos a hacer el ejercicio completo y los gifs de cada prueba.

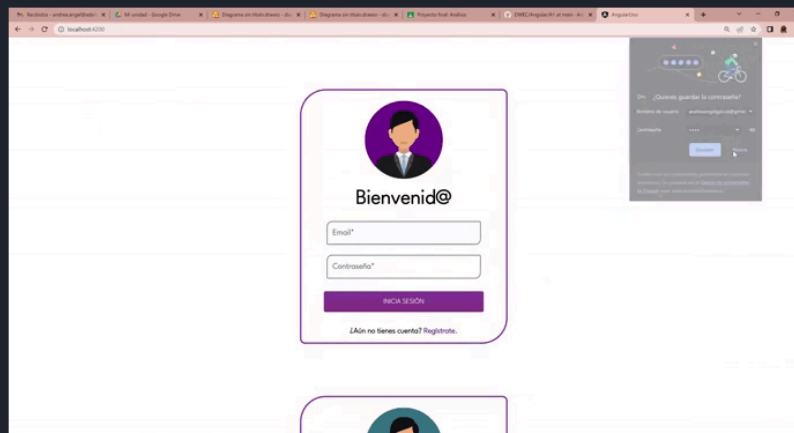


## Pruebas. [🔗](#)

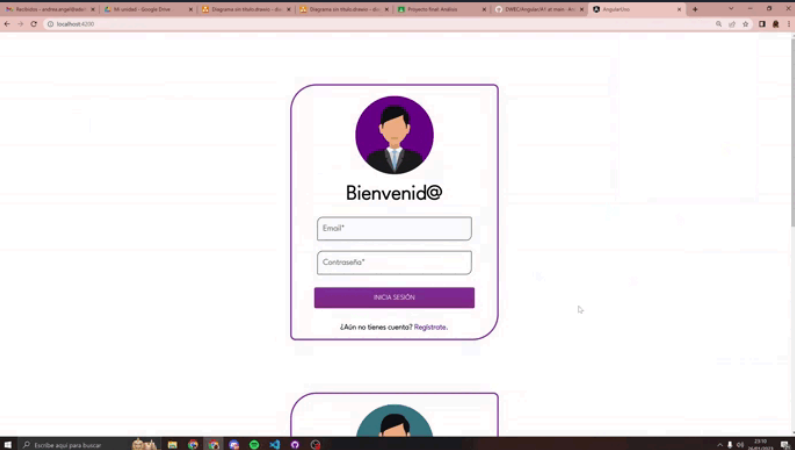
-> Plan de pruebas:

TestID	TestName	Description	StepNumber	StepAction	StepCondition
1	CASO OK - LOGIN	Entramos en la página, escribimos el correo y la contraseña correctamente y nos deja darle al botón.	1	Abrimos la URL.	Aparece una página web con login, registro y footer
			2	Escribimos el correo correctamente	Se muestra el correo
			3	Escribimos la contraseña	Aparece la contraseña oculta y podemos darle al botón
2	CASO KO1 - LOGIN	Entramos en la página, escribimos el correo y no ponemos ninguna contraseña.	1	Abrimos la URL.	Aparece una página web con su logo, buscador y una tabla
			2	Escribimos el correo correctamente	Se muestra la tabla vacía y el mensaje de que no hay ningún pokémon
			3	No escribimos ninguna contraseña	No nos deja darle al botón
3	CASO KO2 - LOGIN	Entramos en la página, escribimos mal el correo y no nos deja darle al botón.	1	Abrimos la URL.	Aparece una página web con su logo, buscador y una tabla
			2	Escribimos mal el correo	Se muestra la tabla vacía y el mensaje de que no hay ningún pokémon
			3	No escribimos ninguna contraseña	No nos deja darle al botón
4	CASO OK - REGISTRO	Entramos en la página, escribimos los datos correctamente y nos deja darle al botón.	1	Abrimos la URL.	Aparece una página web con login, registro y footer
			2	Escribimos el correo correctamente	Se muestra el correo
			3	Escribimos la contraseña	Aparece la contraseña oculta y podemos darle al botón
5	CASO KO - REGISTRO	Entramos en la página, escribimos los datos incorrectamente y no nos deja darle al botón.	1	Abrimos la URL.	Aparece una página web con login, registro y footer
			2	Escribimos los datos incorrectamente	No nos deja darle click
			3	No nos deja darle al botón	Aparece la contraseña oculta y no podemos darle al botón

-> LOGIN: [🔗](#)

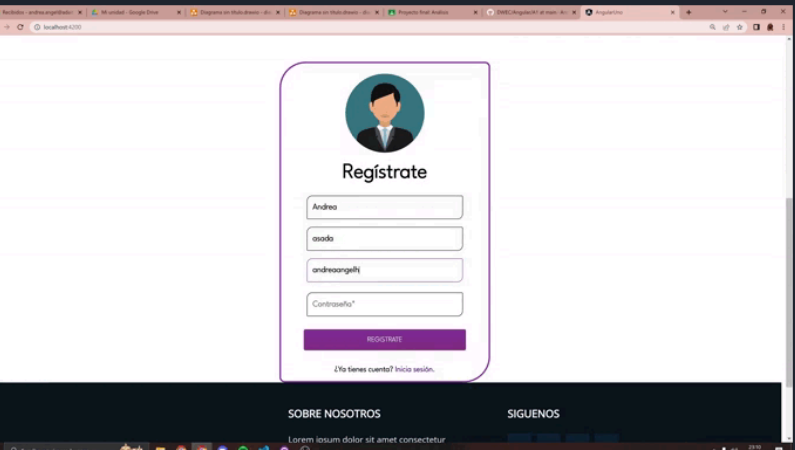


-> LOGIN: [🔗](#)



CASO OK:  
Entramos en la página,  
ponemos el correo correctamente  
e introducimos la contraseña.

-> REGISTRO: [🔗](#)



CASO OK:  
Entramos en la página,  
ponemos los datos correctamente  
e introducimos la contraseña.

Para los que quieran poner vídeos de la ejecución:

### Convertidor Vídeo a GIF

- Hacer click a *Elegir Archivo* y elige el vídeo.
- Una vez elegido, click en *Upload video!*
- Seleccionar *Convert*.
- Puedes recortar el GIF para hacerlo más corto.

Video to animated GIF converter (ezgif.com)

# Criterios de evaluación

<b>Ejercicio 1</b>	No se cumple con alguno de los criterios del formato de entrega o no se realizan las pruebas o no se realiza el gif que verifique el correcto funcionamiento del ejercicio. <b>0 puntos</b>	Se cumple con lo establecido en los criterios del formato de entrega, se realizan las pruebas con éxito y se evidencia mediante un gif. Se encuentra todo documentado en el README.md <b>1 punto</b>
<b>Ejercicio 2</b>	No se cumple con alguno de los criterios del formato de entrega o no se realizan las pruebas o no se realiza el gif que verifique el correcto funcionamiento del ejercicio. <b>0 puntos</b>	Se cumple con lo establecido en los criterios del formato de entrega, se realizan las pruebas con éxito y se evidencia mediante un gif. Se encuentra todo documentado en el README.md <b>1 punto</b>
<b>Ejercicio 3</b>	No se cumple con alguno de los criterios del formato de entrega o no se realizan las pruebas o no se realiza el gif que verifique el correcto funcionamiento del ejercicio. <b>0 puntos</b>	Se cumple con lo establecido en los criterios del formato de entrega, se realizan las pruebas con éxito y se evidencia mediante un gif. Se encuentra todo documentado en el README.md <b>2 puntos</b>
<b>Ejercicio 4</b>	No se cumple con alguno de los criterios del formato de entrega o no se realizan las pruebas o no se realiza el gif que verifique el correcto funcionamiento del ejercicio. <b>0 puntos</b>	Se cumple con lo establecido en los criterios del formato de entrega, se realizan las pruebas con éxito y se evidencia mediante un gif. Se encuentra todo documentado en el README.md <b>2 puntos</b>
<b>Ejercicio 5</b>	No se cumple con alguno de los criterios del formato de entrega o no se realizan las pruebas o no se realiza el gif que verifique el correcto funcionamiento del ejercicio. <b>0 puntos</b>	Se cumple con lo establecido en los criterios del formato de entrega, se realizan las pruebas con éxito y se evidencia mediante un gif. Se encuentra todo documentado en el README.md <b>4 puntos</b>