

DIAGRAMAS DE CLASES Y DE COMUNICACIÓN



Ingeniería Informática de Gestión y Sistemas de Información
Análisis y Diseño de Sistemas de Información

Grupo: REGICODERS

Integrantes:

Ibai Munné

Aitana Niño

Ziyan Jiang

Adrián Vinagre

Sofia Granja

Mateo Ortiz

Diego Pomares

ÍNDICE

ÍNDICE	1
1. DIAGRAMA DE BASE DE DATOS	2
2. DIAGRAMA DE CLASES	3
3. DIAGRAMAS DE COMUNICACIÓN	4
3.1 GESTIÓN DE USUARIOS	4
3.1.1 Registrarse	4
3.1.2 Iniciar sesión	5
3.1.3 Gestionar amigos	6
3.1.4 Actualizar datos	8
3.1.5 Gestionar cuentas	9
3.2 CREAR EQUIPOS POKÉMON	11
3.2.1 Ver un equipo Pokémon.	11
3.2.1.1 Crear un Pokémon.	13
3.2.1.2 Crear una especie.	15
3.2.1.3 Crear un ataque.	16
3.2.1.4 Crear un tipo.	18
3.2.2 Eliminar un equipo Pokémon.	19
3.2.3 Crear un nuevo equipo Pokémon.	20
3.2.4 Añadir un Pokémon a un equipo.	21
3.2.5 Eliminar un Pokémon de un equipo.	24
3.3 CHANGELOG	26
3.4 LISTA COMPLETA DE POKÉMON Y BÚSQUEDAS CON FILTRO	27
3.5 CHATBOT	30
3.6 COMPARTIR A TRAVÉS DE TELEGRAM	33
3.7 POKEDLE	34

1. DIAGRAMA DE BASE DE DATOS

USUARIO	
PK	<u>username</u>
	email
	contraseña
	foto
	esAdmin
	aprobado
	cuentaTelegram

SEGUIDORES	
PK, FK1 PK, FK2	<u>seguidor</u> <u>seguido</u>

EQUIPO	
PK	<u>IDEquipo</u>
	nombre
	fechaCreación
FK	username

EQUIPO_POKÉMON	
PK, FK1 PK, FK2	<u>IDPokemon</u> <u>IDEquipo</u>

POKÉMON	
PK	<u>IDPokemon</u>
	Nombre
	Ataque
	AtaqueEsp
	Def
	DefEsp
	Vel
	Vida
FK	NombreEspecie

ESPECIE	
PK	<u>NombreEspecie</u>
	Ataque
	AtaqueEsp
	Def
	DefEsp
	Vida
	Velocidad
	foto
	esLegendario
	shiny

ATAQUE	
PK	<u>NombreAtaque</u>
	Damage
	Descripción
FK	nombreTipo

TIPO	
PK	<u>NombreTipo</u>

ESPECIE_ATAQUE	
PK, FK1 PK, FK2	<u>NombreEspecie</u> <u>NombreAtaque</u>

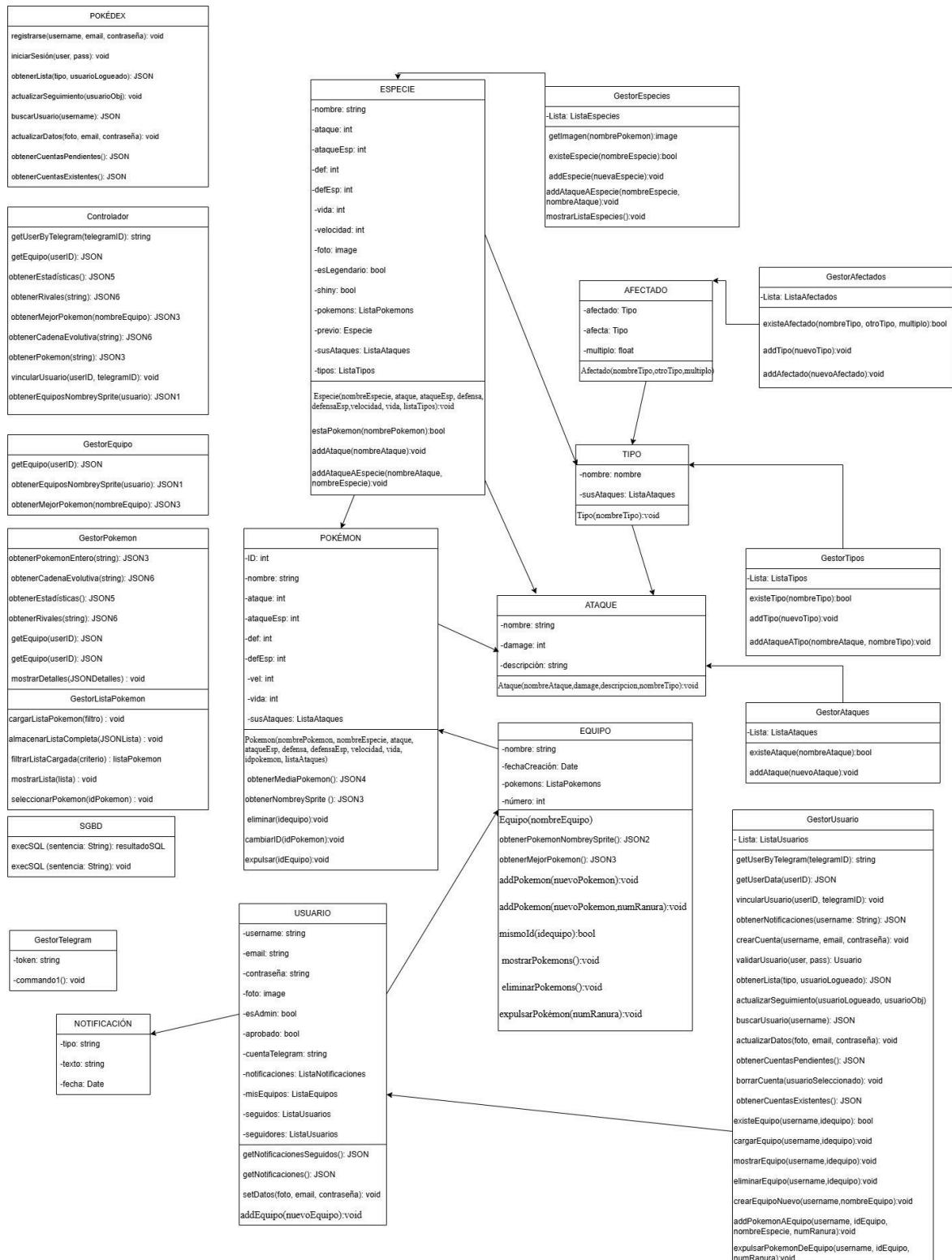
ESPECIE_TIPO	
PK, FK1 PK, FK2	<u>NombreEspecie</u> <u>NombreTipo</u>

NOTIFICACIONES	
PK, FK PK	<u>username</u> <u>Fecha</u>
	tipo
	texto

AFECTADO	
PK, FK1 PK, FK2	<u>afectaTipo</u> <u>afectadoTipo</u>
	múltiplo

POKÉMON_ATAQUE	
PK, FK1 PK, FK2	<u>IDPokemon</u> <u>NombreAtaque</u>

2. DIAGRAMA DE CLASES

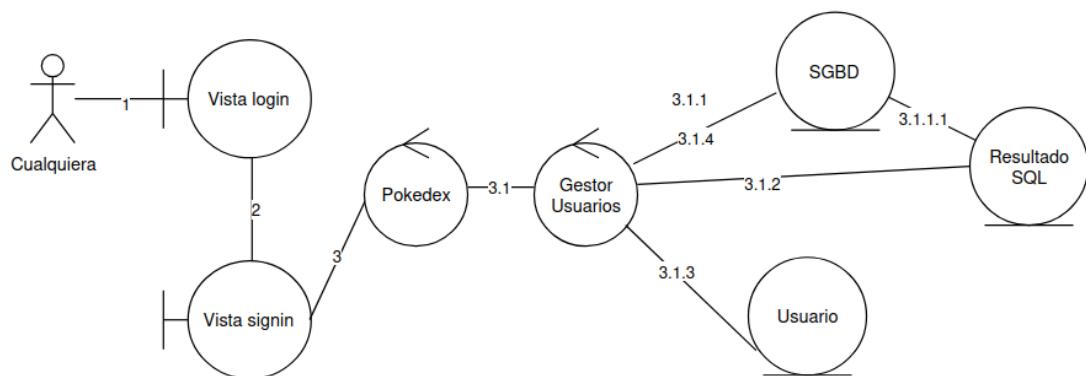


3. DIAGRAMAS DE COMUNICACIÓN

3.1 GESTIÓN DE USUARIOS

3.1.1 Registrarse

Este caso de uso permite a un actor **Cualquiera** crear una cuenta de usuario, la cual se guardará en la base de datos con el estado inicial de “pendiente de aprobación”.



```
1. En la página de inicio de sesión, pulsa "Registrarse"
2. En la vista de registro introduce sus datos y pulsa "Registrarse"
3. registrarSe(username, email, contraseña): void
    3.1. crearCuenta(username, email, contraseña)
        3.1.1. execSQL(sql1): ResultadoSQL
            3.1.1.1. new ResultadoSQL()
        3.1.2. next(): bool
            [SI FALSE]
            3.1.3. new Usuario(username, email, pass, aprobado=false)
            3.1.4. execSQL(sql2)
```

```
sql1: "SELECT username FROM USUARIO WHERE
username=%user%"

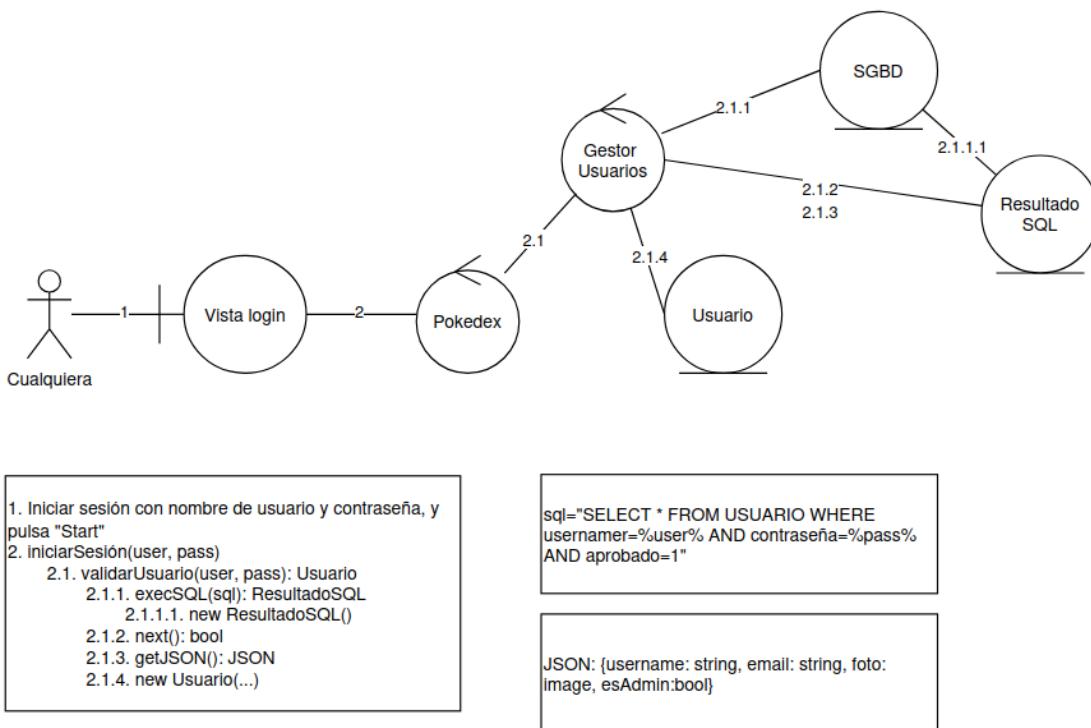
sql2: "INSERT INTO USUARIO(username, email, contraseña,
esAdmin, aprobado) VALUES (%username%, %email%
%contraseña%, 0, 0)"
```

EXPLICACIÓN:

- El actor pulsa “Registrarse” de la vista login (Ilustración 1) y en la vista signin (Ilustración 2) ingresa sus datos (nombre de usuario, email y contraseña).
- El controlador Pokédex ejecuta la función registrarSe(...) que delega a GestorUsuarios.crearCuenta(...).
- Se verifica en la base de datos si el username ya existe (sql1), en caso afirmativo, el proceso termina, de lo contrario, continúa con el registro.
- Se crea un objeto Usuario en memoria con aprobado=false y se inserta en la base de datos con aprobado=0 y esAdmin=0.

3.1.2 Iniciar sesión

El proceso de inicio de sesión se inicia con la interacción del actor **Cualquiera** y su objetivo es verificar la identidad del usuario y activar su sesión en el sistema.

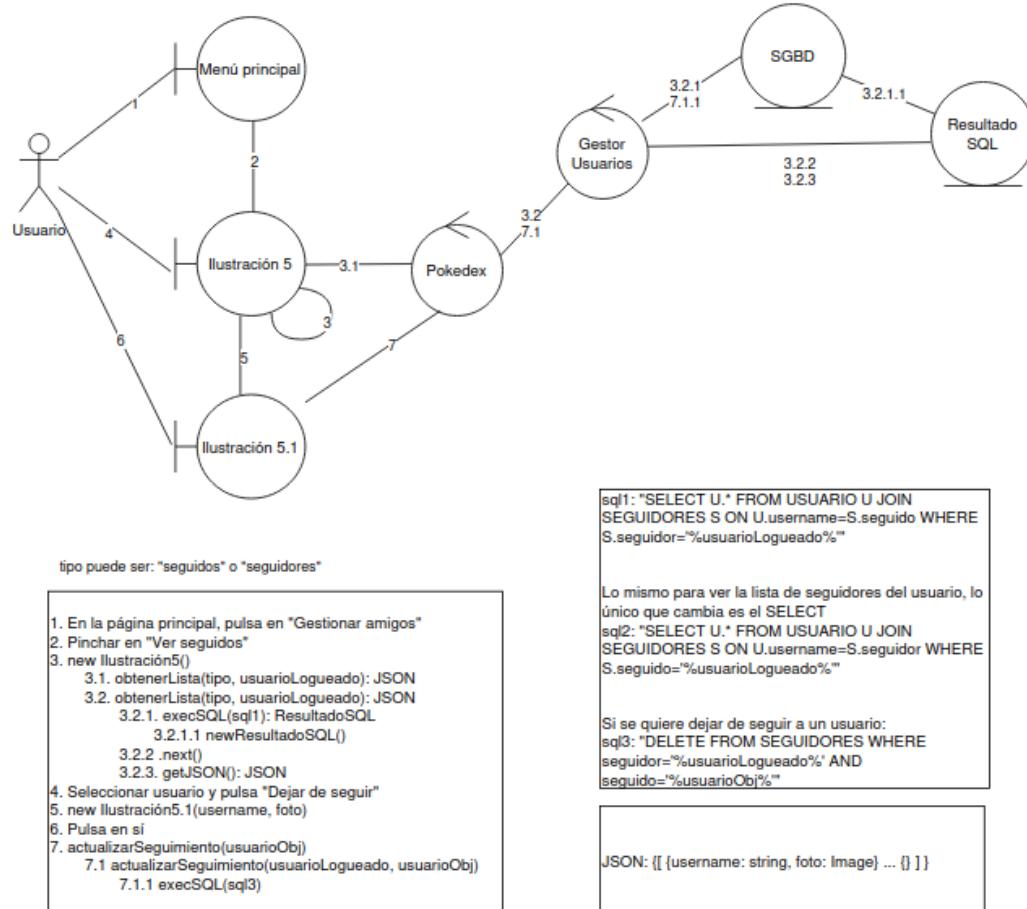


EXPLICACIÓN:

- En la vista login (Ilustración 1), el actor ingresa username y contraseña y pulsa “Start”.
- El sistema ejecuta una consulta SQL que verifica si las credenciales son correctas y si la cuenta está aprobada.
- Si la validación es exitosa, se crea un objeto Usuario en memoria con los datos del usuario, mediante el método getJSON().
- La interfaz se actualiza mostrando el panel de usuario normal o administrador según el valor de esAdmin.

3.1.3 Gestionar amigos

En el siguiente diagrama de comunicación se refleja el flujo de listar usuarios seguidos y dejar de seguir a un usuario.



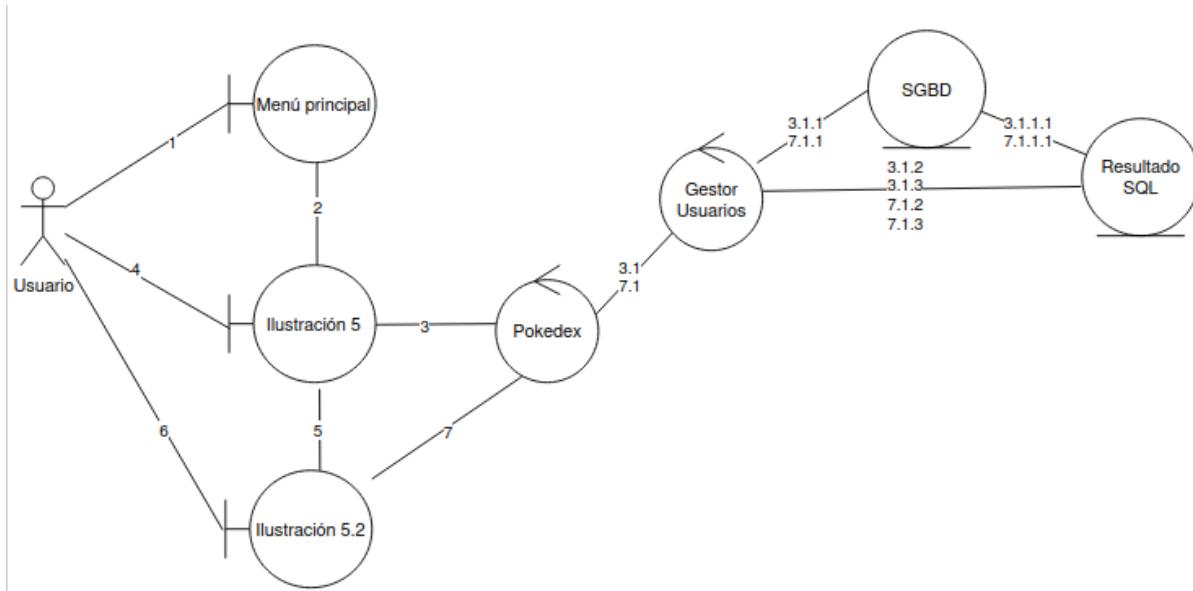
EXPLICACIÓN:

El proceso de gestión de usuarios seguidos tiene dos fases:

- Listar usuarios seguidos (pasos 1 al 3.2.3)
 - El usuario en el menú principal (Ilustración 4) pulsa “Gestionar amigos” y en la Ilustración 5 pulsa en “Ver seguidos”.
 - El sistema consulta la base de datos para obtener la lista de usuarios que sigue el usuario actual.
 - Se construye un JSON con los datos (username y foto) de los usuarios seguidos y se muestra la lista completa en la interfaz.
- Dejar de seguir (pasos 4 al 7.1.1)
 - El usuario selecciona un contacto de la lista de usuarios seguidos y pulsa “Dejar de seguir”.
 - Se ejecuta una sentencia DELETE en la tabla de SEGIDORES para eliminar la relación.

Para ver la lista de seguidores sigue los mismos pasos desde el 1 hasta 3.2.3. Lo único que cambia es la consulta, en vez de ejecutar el sql1, se ejecuta sql2.

Buscar usuario y seguirle



1. En la página principal, pulsa en "Gestionar amigos"
2. Introduce un nombre y pulsa en el ícono de la lupa
3. buscarUsuario(username): JSON
 - 3.1. buscarUsuario(username)
 - 3.1.1. execSQL(sql1): ResultadoSQL
 - 3.1.1.1. new ResultadoSQL()
 - 3.1.1.2. next(): bool
 - 3.1.1.3. getJSON(): JSON
 4. Seleccionar usuario y pulsa "Seguir"
 5. new Ilustración5.2(username, foto)
 6. Pulsa en si
 7. actualizarSeguimiento(usuarioObj)
 - 7.1 actualizarSeguimiento(usuarioObj)
 - 7.1.1. execSQL(sql2): ResultadoSQL
 - 7.1.1.1. new ResultadoSQL()
 - 7.1.1.2. next(): bool
 - [SI FALSE]
 - 7.1.3. execSQL(sql3)

```
JSON: [[ {nombre: string, foto: Image} ... {} ]]
```

```

sql1: "SELECT username, foto FROM USUARIO WHERE username
LIKE %username%"

sql2: "SELECT * FROM SEGUIDORES WHERE
seguido=%usuarioLogueado% AND seguido=%usuarioObj%"

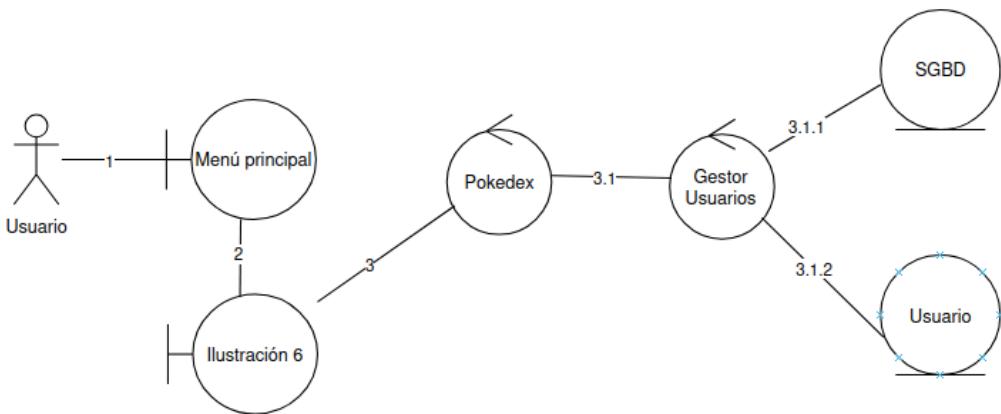
sql3: "INSERT INTO SEGUIDORES VALUES(%usuarioLogueado%,
%usuarioObj%)"
  
```

EXPLICACIÓN:

El proceso de búsqueda y gestión de seguimiento tiene dos fases:

- Búsqueda de usuario (pasos 1 al 3.1.3)
 - El usuario en el menú principal (Ilustración 4) pulsa “Gestionar amigos” y en la barra de búsqueda (Ilustración 5) introduce el nombre del usuario a buscar.
 - El sistema consulta la base de datos usando LIKE para encontrar coincidencias parciales en usernames y se devuelve un JSON con los usuarios encontrados (username y foto) para mostrar en pantalla.
- Gestión de seguimiento (pasos 4 al 7.1.1)
 - Selecciona un usuario y pulsa “Seguir”. Verifica con sql2, sólo se inserta el seguimiento en la tabla SEGUIDORES si next() es FALSE, es decir, no hay una tupla con dicho seguidor y seguido.

3.1.4 Actualizar datos



1. Pulsa el ícono de perfil
2. Pulsa en "Guardar cambios"
3. actualizarDatos(foto, email, contraseña)
3.1 actualizarDatos(foto, email, contraseña)
3.1.1. execSQL(sql)
3.1.2. setDatos(foto, email, contraseña)

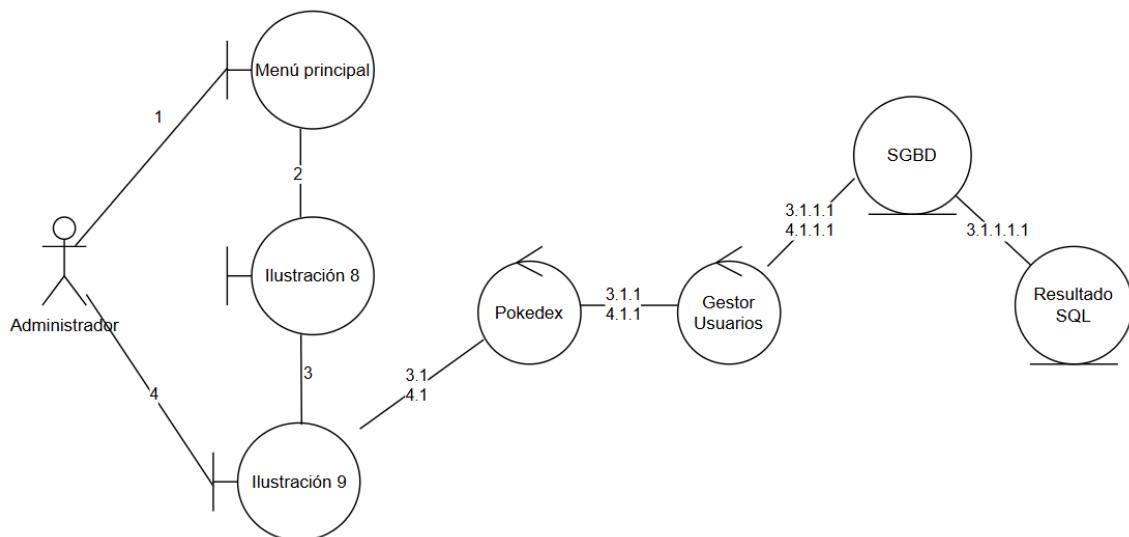
```
sql: "UPDATE USUARIO SET email=%email%,  
contraseña=%contraseña%, foto=%foto% WHERE  
username=%username%"
```

EXPLICACIÓN:

- En el menú principal (Ilustración 4) pulsa en el ícono del perfil. El usuario introduce nuevos cambios y pulsa en “Guardar cambios”.
- Se ejecuta una sentencia UPDATE en la base de datos para modificar los datos del usuario en la tabla USUARIO y se actualiza el objeto Usuario en memoria, que tiene la sesión activa, con los nuevos datos.

3.1.5 Gestionar cuentas

Aprobar cuenta



1. En el menú principal, pulsa "Gestionar cuentas"
2. En Ilustración 8, pulsa "Aprobar cuenta"
3. new Ilustración9()
3.1 obtenerCuentasPendientes(): JSON
3.1.1 obtenerCuentasPdtes(): JSON
3.1.1.1. execSQL(sql1): ResultadoSQL
3.1.1.1. new ResultadoSQL()
3.1.1.2. next()
3.1.1.3. getJSON(): JSON
4. Seleccionar cuenta y pulsa "Aprobar"
4.1. aprobarCuenta(usuarioSeleccionado)
4.1.1. aprobarCuenta(usuarioSeleccionado)
4.1.1.1. execSQL(sql2)

sql1: "SELECT username, email, foto FROM USUARIO WHERE aprobado=0"

sql2: "UPDATE USUARIO SET aprobado=1 WHERE username=%usuarioSeleccionado%"

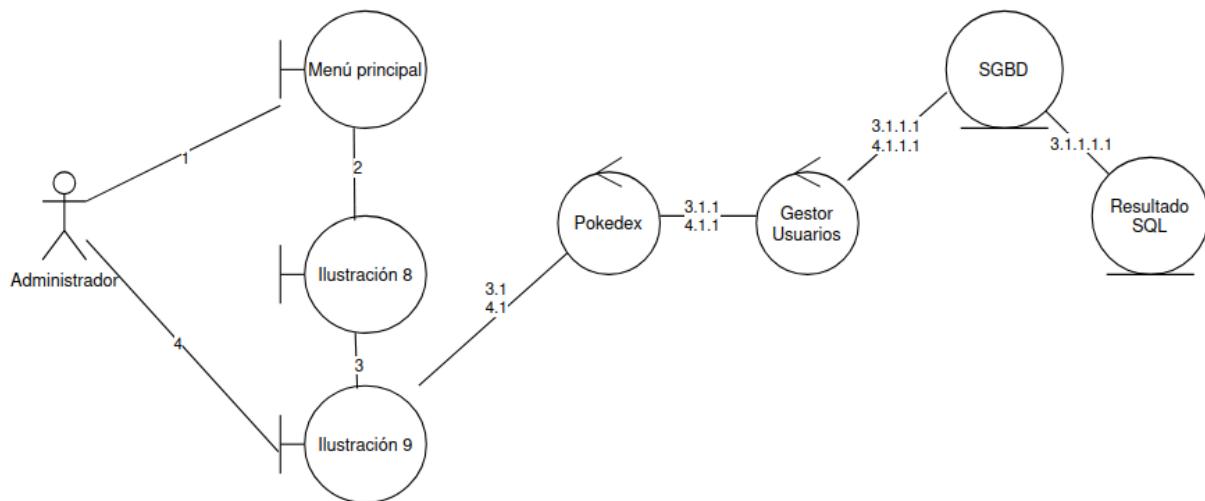
JSON: [{username: string, email:string, foto: Image} ... {}]

EXPLICACIÓN:

El proceso de aprobación de cuentas por parte del administrador se divide en dos partes:

- Listar cuentas pendientes (pasos 1 al 3.1.1.3)
 - El administrador accede a Gestionar cuentas desde el menú principal (Ilustración 4) y pulsa en “Aprobar cuenta”.
 - El sistema consulta la base de datos para obtener usuarios con cuentas pendientes de aprobar (aprobado=0). Y se construye un JSON (username, email, foto) de las cuentas pendientes.
 - En la interfaz se muestra la lista de cuentas por aprobar.
- Aprobar cuenta
 - El administrador selecciona una cuenta y pulsa “Aprobar”. Se ejecuta la sentencia UPDATE que cambia el atributo aprobado de 0 a 1 para ese usuario.

Borrar cuenta



1. En el menú principal, pulsa "Gestionar cuentas"
2. En Ilustración 8, pulsa "Borrar cuenta"
3. new Ilustración10()
 - 3.1 obtenerCuentasExistentes(): JSON
 - 3.1.1 obtenerCuentasExistentes(): JSON
 - 3.1.1.1. execSQL(sql1): ResultadoSQL
 - 3.1.1.1. new ResultadoSQL()
 - 3.1.1.2. next()
 - 3.1.1.3. getJSON(): JSON
4. Seleccionar cuenta y pulsa "Borrar"
 - 4.1. borrarCuenta(usuarioSeleccionado)
 - 4.1.1. borrarCuenta(usuarioSeleccionado)
 - 4.1.1.1. execSQL(sql2)

```
sql1: "SELECT username, email, foto FROM USUARIO WHERE
aprobado=1"
```

```
sql2: "DELETE FROM USUARIO WHERE
username=%usuarioSeleccionado%"
```

```
JSON: [{ username: string, email:string, foto: Image} ... {} ] }
```

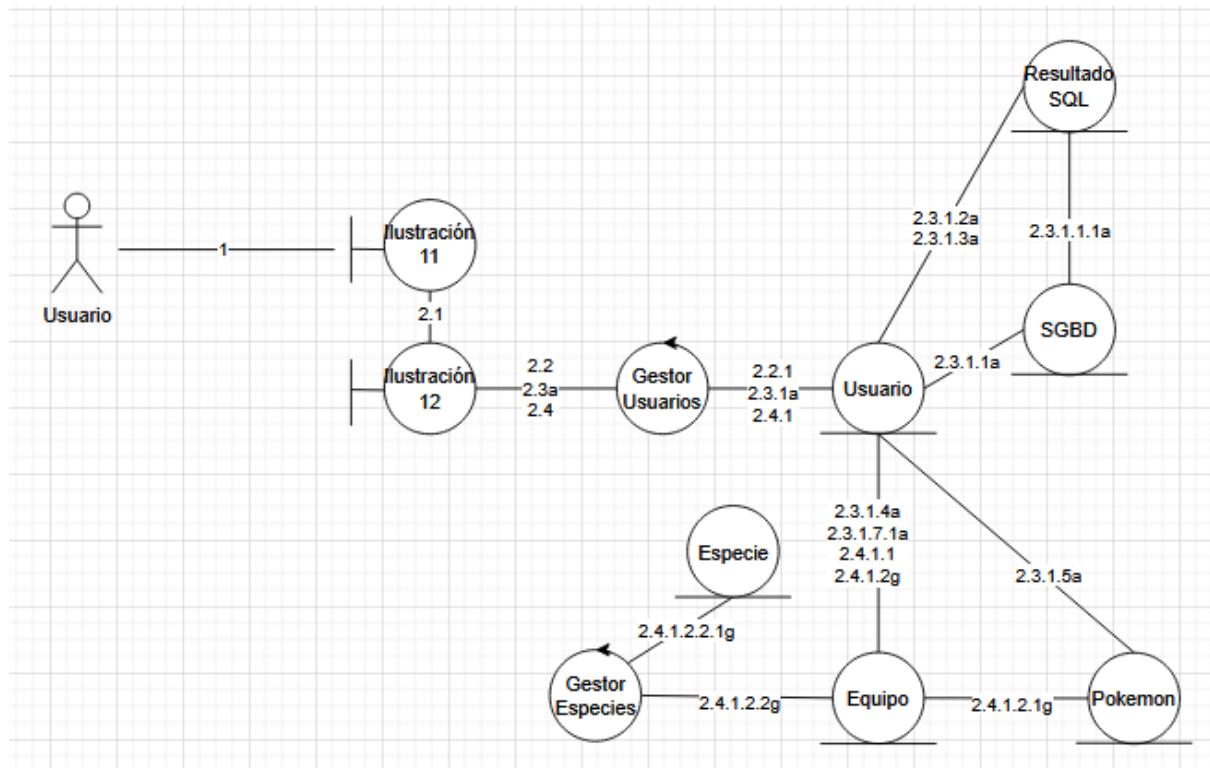
EXPLICACIÓN:

El proceso de borrado de cuentas es similar al de aprobar cuentas. También se divide en dos partes:

- Listar cuentas existentes (pasos 1 al 3.1.1.3)
 - El administrador accede a Gestionar cuentas desde el menú principal (Ilustración 4) y pulsa en "Borrar cuenta".
 - El sistema consulta la base de datos para obtener todos los usuarios existentes y aprobados. Y se construye un JSON (username, email, foto) de las cuentas existentes.
 - En la interfaz se muestra la lista de usuarios aprobados.
- Borrar cuenta
 - El administrador selecciona una cuenta y pulsa "Borrar". Se ejecuta la consulta DELETE.

3.2 CREAR EQUIPOS POKÉMON

3.2.1 Ver un equipo Pokémon.



1. Abrir aplicación para acceder al menú principal
2. Hacer click en el botón “Ver equipos”
 - 2.1 new Ilustración 12()
 - 2.2 existeEquipo(username,idequipo): bool
 - 2.2.1 existeEquipo(idequipo):bool
[SI FALSE]
 - 2.3a cargarEquipo(username,idequipo):void
 - 2.3.1a cargarEquipo(idequipo):void
 - 2.3.1.1a execSQL(sql1)
 - 2.3.1.1.1a new ResultadoSQL():void
 - 2.3.1.2a next():bool
 - 2.3.1.3a getJSON():JSON1
 - 2.3.1.4a nuevoEquipo= new Equipo(nombreEquipo)

```

2.3.1.5a nuevoPokemon = new Pokemon(nombrePokemon, nombreEspecie,
    ataque, ataqueEsp, defensa, defensaEsp, velocidad, vida, idpokemon,
    listaAtaques) <-- ejecutar por cada pokemon

2.3.1.6a addEquipo(nuevoEquipo):void

2.3.1.7a addPokemonAEquipo(nuevoPokemon,idequipo):void ← ejecutar por
    cada pokemon

2.3.1.7.1a addPokemon(nuevoPokemon):void

2.4 mostrarEquipo(username,idequipo):void
    2.4.1 mostrarEquipo(idequipo):void
        2.4.1.1 mismoid(idequipo):bool
            [SI TRUE]
        2.4.1.2g mostrarPokemons():void
            2.4.1.2.1g getNombrePokemon():String ← por cada Pokemon
            2.4.1.2.2g getImagen(nombrePokemon):image← por cada
                Pokemon
            2.4.1.2.2.1g estaPokemon(nombrePokemon):bool

```

JSON1: {nombreEquipo:String,

[{nombreEspecie:String, nombrePokemon:String, ataque:Int, ataqueespecial:Int, defensa:Int,
defensaespecial:Int, velocidad:Int, vida:Int, idpokemon:Int,
[{nombreAtaque:String},{}] },{}] }

SQL1:

```

SELECT eq.nombreequipo, pok.nombreEspecie, pok.nombre, pok.ataque, pok.ataqueespecial,
pok.defensa, pok.defensaespecial, pok.velocidad, pok.vida, pok.IDPokemon, atq.nombre
FROM Equipo AS eq, Equipo_Pokemon AS eqpok, Pokemon AS pok, Ataque AS atq,
Pokemon_Ataque AS atqpok, Usuario AS usr
WHERE eqpok.idequipo = %idequipo% AND usr.username = %username% AND eq.idequipo =
eqpok.idequipo AND pok.idpokemon = eqpok.idpokemon AND atq.nombre = atqpok.nombreataque
AND usr.username = eq.username

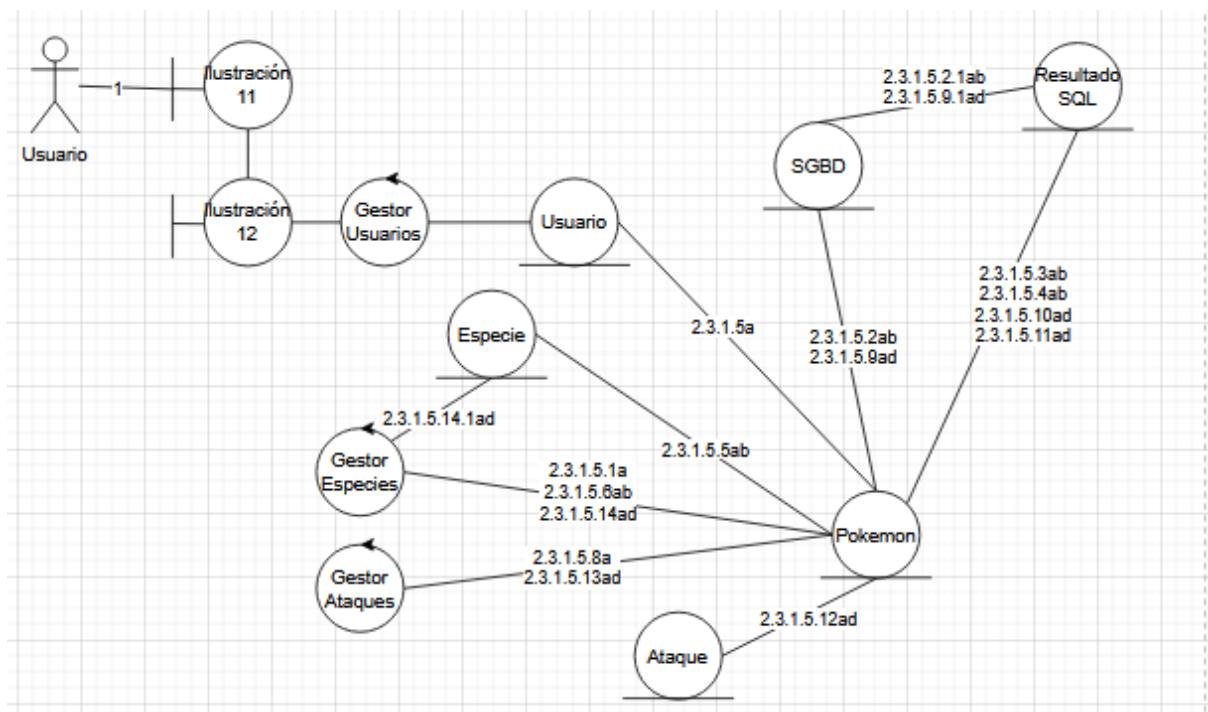
```

EXPLICACIÓN:

- La creación de pokémons, equipos, tipos, y especies son algo complejas y ocupan mucho texto y espacio en los diagramas, así que aparecen por separado.
- Dado un id de equipo y un nombre de usuario, acceder al gestor de usuarios para ver si está el equipo a mostrar cargado o no.

- Encontrar al usuario en el gestor, y acceder al usuario dándole el id del equipo como parámetro para encontrar el equipo.
- Si no está (1), cargar equipo. Para ello, pedir a la base de datos el nombre del equipo, los atributos de los Pokémon que forman parte de él, y los nombres de esas especies. Además, por cada Pokémon se pide también una lista de nombres de ataques que tienen aprendidos.
- (1) Se crea un nuevo equipo con el nombre obtenido.
- (1) Por cada Pokémon, se crea un nuevo Pokémon con la información obtenida.
- (1) Se añade el nuevo equipo creado a la lista de equipos del usuario.
- (1) Por cada Pokémon, se añade al equipo recién creado.

3.2.1.1 Crear un Pokémon.



2.3.1.5a nuevoPokemon = new Pokemon(nombrePokemon, nombreEspecie, ataque, ataqueEsp, defensa, defensaEsp, velocidad, vida, idpokemon, listaAtaques)

2.3.1.5.1a existeEspecie(nombreEspecie):bool

[SI FALSE]

2.3.1.5.2ab execSQL(sql2)

2.3.1.5.2.1ab new ResultadoSQL(): void

2.3.1.5.3ab next():bool

2.3.1.5.4ab getJSON():JSON2

2.3.1.5.5ab nuevaEspecie = new Especie(nombreEspecie, ataque, ataqueEsp,

```

defensa, defensaEsp, velocidad, vida, listaTipos):void
2.3.1.5.6ab addEspecie(nuevaEspecie):void
2.3.1.5.7a addPokemonAEspecie(nombrePokemon):void
2.3.1.5.8a existeAtaque(nombreAtaque):bool <-- ejecutar por cada ataque (*)
[SI FALSE]
2.3.1.5.9ad execSQL(sql3)<-- (*)
    2.3.1.5.9.1ad new ResultadoSQL():void
    2.3.1.5.10ad next():bool<-- (*)
    2.3.1.5.11ad getJSON():JSON3<-- (*)
    2.3.1.5.12ad nuevoAtaque = new Ataque(nombreAtaque, damage,
                                              descripcion, nombreTipo):void<-- (*)
    2.3.1.5.13ad addAtaque(nuevoAtaque):void<-- (*)
    2.3.1.5.14ad addAtaqueAEspecie(nombreEspecie, nombreAtaque):void<--(*)
        2.3.1.5.14.1ad addAtaque(nombreAtaque):void

```

JSON2: {nombreEspecie:String, ataque:Int, ataqueEsp:Int, defensa:Int, defensaEsp:Int, Vida:Int, Velocidad:Int, esLegendario:bool, esShiny:bool, [{nombreTipo:String}, {}]}

JSON3: {nombreAtaque:String, damage:Int, descripcion:String, nombreTipo:String}

SQL2: SELECT e.NombreEspecie, e.ataque, e.ataqueespecial, e.defensa, e.defensaespecial, e.velocidad, e.vida, e.foto, e.esLegendario, e.esShiny, t.nombretipo
FROM Especie AS e, Tipo AS t, Especie_Tipo AS et
WHERE e.NombreEspecie = %nombreEspecie% AND e.nombreEspecie = et.nombreEspecie AND t.nombreTipo = et.nombreTipo

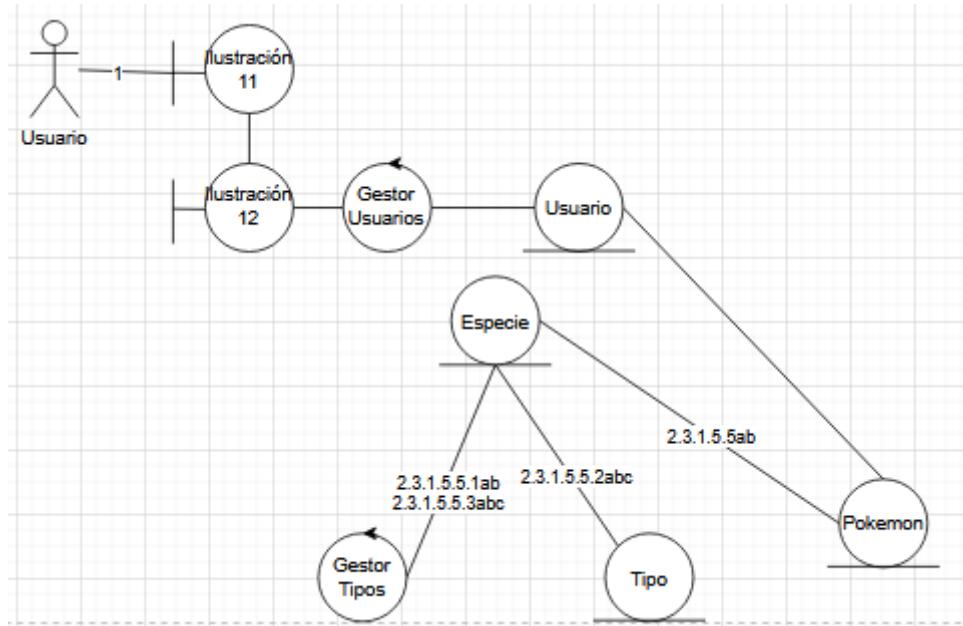
SQL3: SELECT a.nombreAtaque, a.damage, a.descripcion, a.nombreTipo
FROM Ataque AS
WHERE a.nombreAtaque = %nombreAtaque%

EXPLICACIÓN

- Crear un Pokémon pasándole sus estadísticas, y una lista de nombres de ataques.
- Comprobar si ya está cargada la especie a la que pertenece. Si no (1), hay que cargarla.
- (1) Pedir a la base de datos las estadísticas de la especie, y una lista de los nombres de los tipos que tiene (fuego, agua, planta...)
- (1) Crear la nueva especie. Añadir la nueva especie al gestor de especies.

- (1) Añadir el nombre del nuevo Pokémon a la especie.
- Por cada ataque, comprobar si existe en el gestor de ataques, dándole un nombre de ataque al gestor. Si no existe (2), pedir a la base de datos los atributos del ataque.
- (2) Crear nuevo ataque con la información obtenida. Guardar el ataque en el gestor de ataques.
- (2) Añadir el nombre del ataque a la lista de ataques de la especie.

3.2.1.2 Crear una especie.

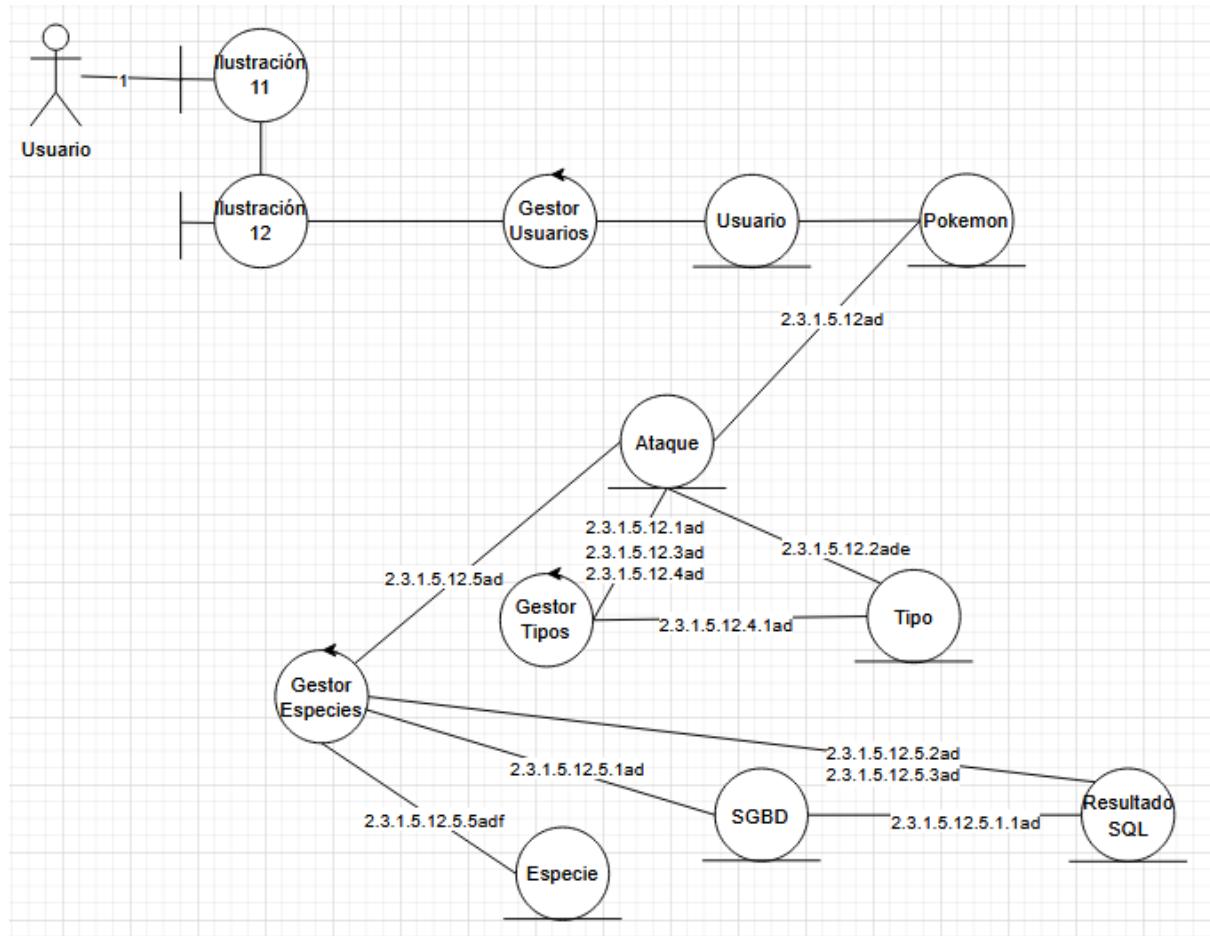


2.3.1.5.5ab new Especie(nombreEspecie, ataque, ataqueEsp, defensa, defensaEsp, velocidad, vida, listaTipos):void
 2.3.1.5.5.1ab existeTipo(nombreTipo):bool <- ejecutar por cada tipo (*)
 [SI FALSE]
 2.3.1.5.5.2abc nuevoTipo = new Tipo(nombreTipo):void<--(*)
 2.3.1.5.5.3abc addTipo(nuevoTipo):void<--(*)

EXPLICACIÓN

- Comprobar si existen los tipos de la especie. Para ello, Por cada tipo, acceder al gestor de tipos con el nombre del tipo y ver si está. Si no está, crear un nuevo tipo y añadirlo al gestor de tipos. También añadir su nombre a la lista de tipos de la especie (lista.add(nombreTipo);).

3.2.1.3 Crear un ataque.



```

2.3.1.5.12ad nuevoAtaque = new Ataque(nombreAtaque, damage,
                                         descripción, nombreTipo):void
2.3.1.5.12.1ad existeTipo(nombreTipo):bool
[SI FALSE]
2.3.1.5.12.2ade new Tipo(nombreTipo):void
2.3.1.5.12.3ade addTipo(nuevoTipo):void
2.3.1.5.12.4ad addAtaqueATipo(nombreAtaque, nombreTipo):void
2.3.1.5.12.4.1ad addAtaqueATipo(nombreAtaque):void
2.3.1.5.12.4.1.1ad addAtaque(nombreAtaque):void
2.3.1.5.12.5ad addAtaqueAEspecie(nombreAtaque):void
2.3.1.5.12.5.1ad execSQL(sql4)
2.3.1.5.12.5.1.1ad new ResultadoSQL():void

```

2.3.1.5.12.5.2ad next():bool
2.3.1.5.12.5.3ad getJSON():JSON4
2.3.1.5.12.5.4ad existeEspecie(nombreEspecie):bool <-- por cada especie

[SI TRUE]

2.3.1.5.12.5.5adf addAtaqueAEspecie(nombreAtaque, nombreEspecie):void<-- por cada especie

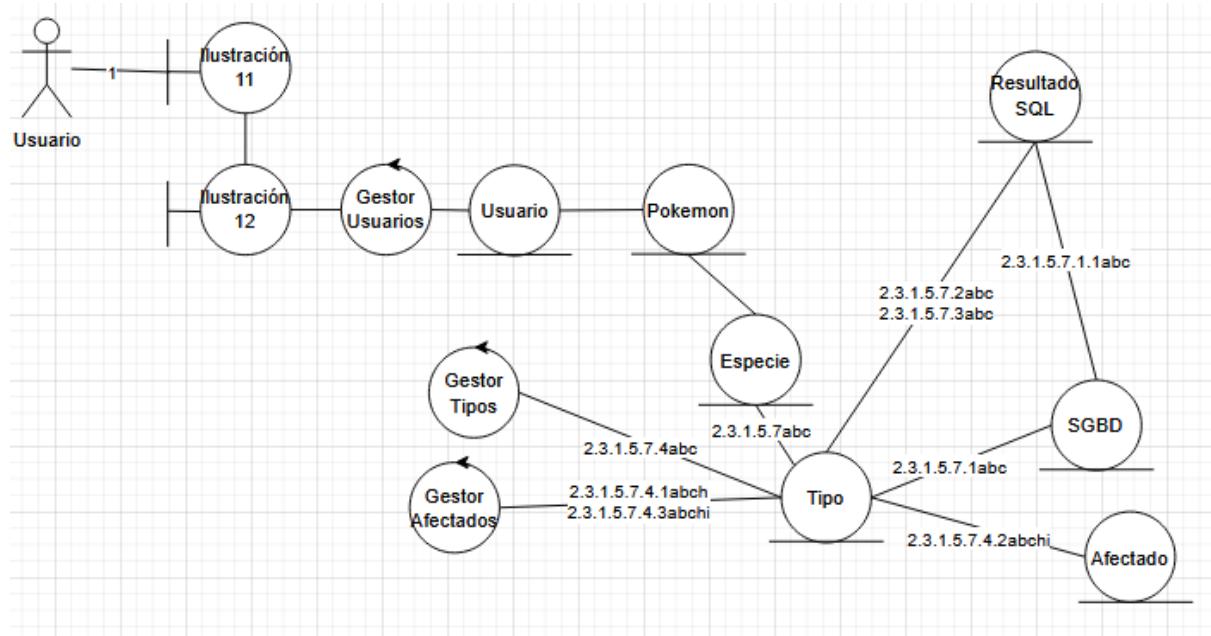
JSON4: {[{nombreEspecie:String},{}]}

SQL4: SELECT e.NombreEspecie
FROM Especie AS e, Ataque AS a , Especie_Ataque AS ea
WHERE a.NombreAtaque = %nombreAtaque% AND ea.NombreEspecie = e.NombreEspecie AND ea.NombreAtaque = a.NombreAtaque

EXPLICACIÓN

- Comprobar si existe el tipo del ataque, accediendo al gestor de tipos con el nombre del tipo. Si no existe, crear el nuevo tipo y añadirlo al gestor de tipos.
- Añadir el nombre del ataque a la lista de ataques del tipo al que pertenezca. Para ello, dado el nombre del tipo y del ataque, acceder al gestor de tipos.
- Añadir el nombre del ataque a la lista de ataques de las especies a las que pertenezca. Para ello, pedir a la base de datos una lista con los nombres de todas las especies que puedan aprender el ataque.
- Por cada especie que pertenezca a la lista obtenida, comprobar si existe. Si existe, añadir el ataque.

3.2.1.4 Crear un tipo.



2.3.1.5.7abc new Tipo(nombreTipo):void

2.3.1.5.7.1abc execSQL(sql5)

2.3.1.5.7.1.1abc new ResultadoSQL()

2.3.1.5.7.2abc next():bool

2.3.1.5.7.3abc getJSON():JSON5

2.3.1.5.7.4abc existeTipo(otroTipo):bool <--Por cada otro tipo

[SI TRUE]

2.3.1.5.7.4.1abch existeAfectado(nombreTipo, otroTipo, multiplo):bool

[SI FALSE]

2.3.1.5.7.4.2abchi nuevoAfectado = new Afectado(nombreTipo, otroTipo, multiplo):void

2.3.1.5.7.4.3abchi addAfectado(nuevoAfectado):void

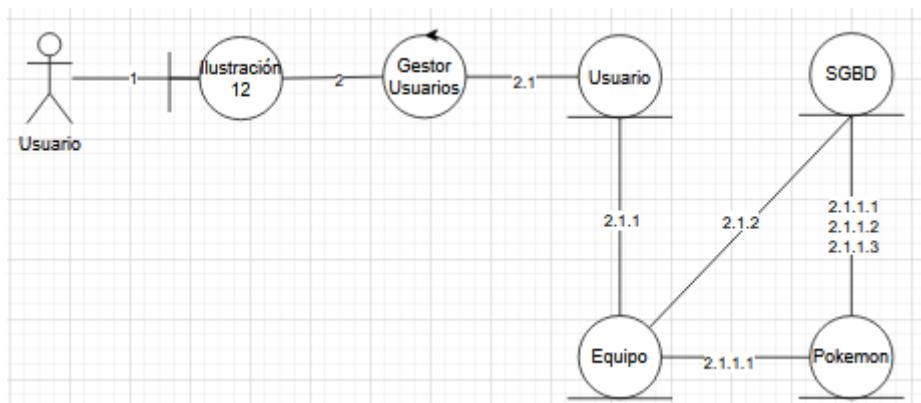
JSON5: {tipoAfectedo:String, tipoAfectedo:string, multiplo:Float}

SELECT * FROM Afectado AS a WHERE a.afectedoTipo = %nombreTipo% OR a.afectedoTipo = %nombreTipo%

EXPLICACIÓN:

- Para crear un tipo nuevo, dado un nombre de tipo, pedir a la base de datos todas las relaciones entre tipos en las que el tipo forma parte.
- Comprobar si existe el otro tipo que forma parte de la relación. Si existe, comprobar también si existe ya la relación, accediendo al gestor de afectados, proporcionando los nombres de los dos tipos y el múltiplo.
- Si no existe, crear nuevo afectado y añadirlo al gestor de afectados.

3.2.2 Eliminar un equipo Pokémon.



1. Desde la interfaz de visualización de un equipo, hacer click en el botón "Eliminar" y luego en el botón "Sí"
2. eliminarEquipo(username,idequipo):void
 - 2.1 eliminarEquipo(idequipo):void
 - 2.1.1 eliminarPokemons():void
 - 2.1.1.1 eliminar(idequipo):void ← por cada pokémon
 - 2.1.1.2 execSQL(sql1)<-- por cada ataque
 - 2.1.1.3 execSQL(sql2)
 - 2.1.1.4 execSQL(sql3)
 - 2.1.2 execSQL(sql4)
 - 2.2 mostrarEquipo(idEquipo):void

SQL1: DELETE FROM Pokemon_Ataque WHERE IDPokemon = %idPokemon% AND NombreAtaque = %nombreAtaque%

SQL2: DELETE FROM Equipo_Pokemon WHERE IDPokemon = %idPokemon% AND IDEquipo = %idEquipo%

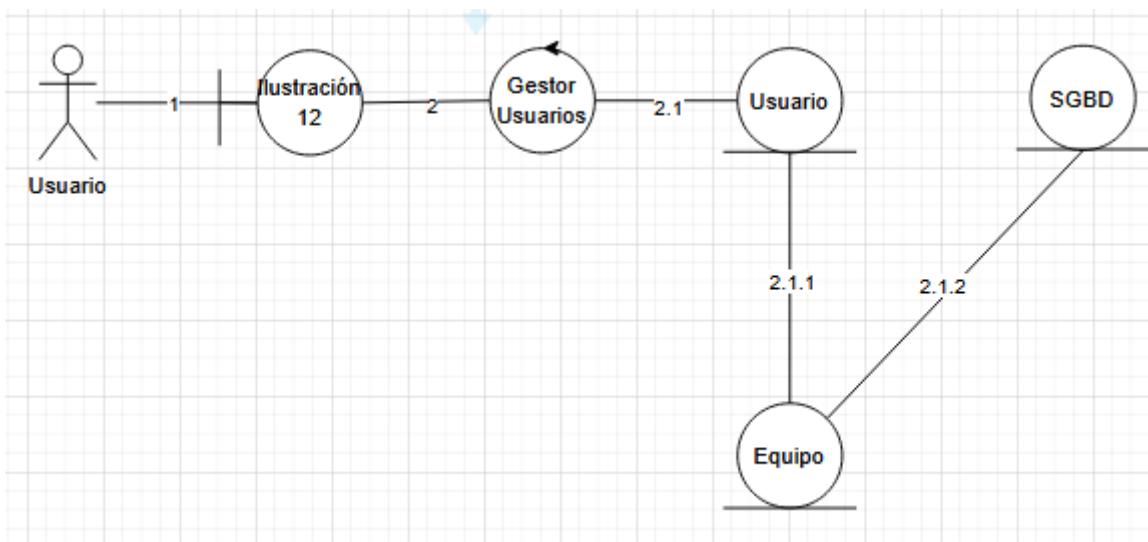
SQL3: DELETE FROM Pokemon WHERE IDPokemon = %idPokemon%

SQL4: DELETE FROM Equipo WHERE IDEquipo = %idEquipo%

EXPLICACIÓN:

- Eliminar el equipo que se muestra actualmente, accediendo al gestor de usuarios y dándole el nombre de usuario y el id del equipo a eliminar. Se recorren los usuarios hasta encontrar el indicado.
- Acceder al usuario encontrado y dado un id, que elimine ese equipo suyo. Para ello, primero hay que eliminar los Pokémon del equipo.
- Eliminar los Pokémon consiste en recorrer la lista de Pokémon del equipo, y por cada Pokémon, eliminarlo. Para ello, dado el id del equipo, se accede al Pokémon, y se recorren los nombres de sus ataques. Por cada nombre de ataque, se accede a la base de datos y se elimina la relación entre el Pokémon y el ataque (el ataque sigue estando en el gestor de ataques).
- Para eliminar un Pokémon también se usa el id del Pokémon y del equipo. Se accede a la base de datos para eliminar la relación entre ellos dos.
- Después de esto, se regresa al equipo en cuestión y se borran todos los Pokémon de su lista de Pokémon, y se accede a la base de datos para eliminar el equipo.
- Por último, se regresa al usuario, y se elimina el equipo de su lista de equipos, reorganizando el número de todos los equipos y mostrando el nuevo equipo de número 1.

3.2.3 Crear un nuevo equipo Pokémon.



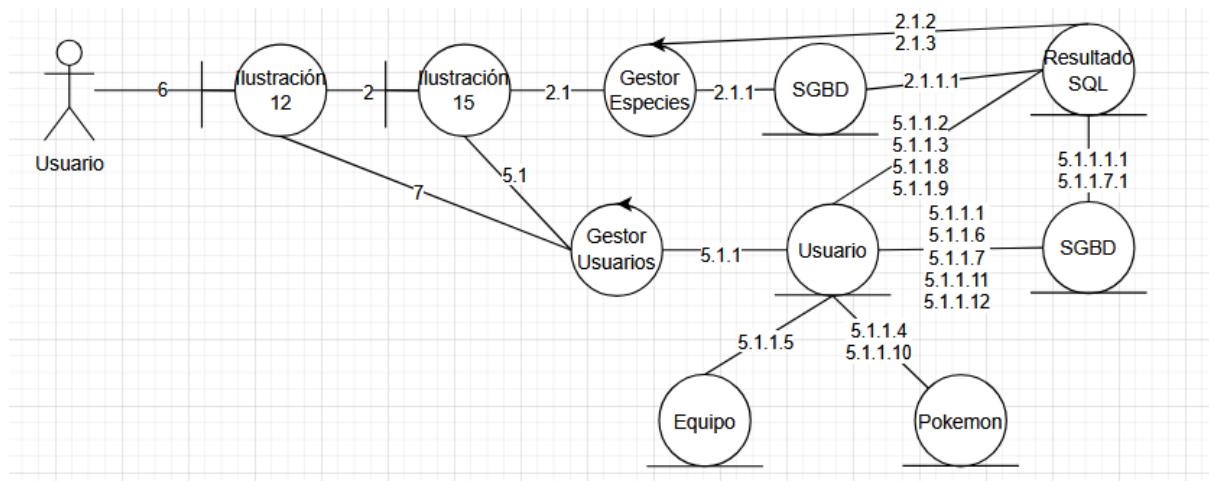
1. Desde la interfaz de visualización de un equipo, hacer click en el botón "Crear nuevo equipo" y luego en el botón "Sí"
2. `crearEquipoNuevo(username, nombreEquipo):void`
 - `2.1 crearEquipoNuevo(nombreEquipo):void`
 - `2.1.1 nuevoEquipo = new Equipo(nombreEquipo):void`
 - `2.1.2 addEquipo(nuevoEquipo):void`
 - `2.1.3 execSQL(sql1):void`

SQL1: `INSERT INTO Equipo(nombre, fechaCreacion, username) VALUES %nombreequipo%, %fecha%, %username%`

EXPLICACIÓN:

- Crear un equipo nuevo, accediendo al gestor de usuarios, y dándole un nombre de usuario y un nombre para el nuevo equipo.
- Buscar al usuario y acceder al mismo, pasándole el nombre del nuevo equipo.
- Crear el nuevo equipo, añadirlo a la lista de equipos del usuario, y acceder a la base de datos pasándole los datos del equipo. El ID se generará automáticamente en la BD, la fecha también se generará automáticamente.

3.2.4 Añadir un Pokémon a un equipo.



1. Desde la interfaz de visualización de un equipo, hacer click en el botón "Editar" y luego en el botón "Sí"
2. new Ilustracion15():void
 - 2.1 mostrarListaEspecies():void
 - 2.1.1 execSQL(sql3)
 - 2.1.1.1 new ResultadoSQL()
 - 2.1.2 next():bool
 - 2.1.3 getJSON():JSON3
3. Hacer click en una ranura del equipo Pokémon
4. Hacer click en una de las especies de la lista
5. Hacer click sobre el botón "Añadir" y luego sobre "Guardar"
 - 5.1 addPokemonAEquipo(username, idEquipo, nombreEspecie, numRanura)
 - 5.1.1 addPokemonAEquipo(idEquipo, nombreEspecie, numRanura)
 - 5.1.1.1 execSQL(sql1)
 - 5.1.1.1.1 new ResultadoSQL()
 - 5.1.1.2 next():bool
 - 5.1.1.3 getJSON():JSON1
 - 5.1.1.4 nuevoPokemon = new Pokemon(nombrePokemon, nombreEspecie, ataque, ataqueEsp, defensa,

```

        defensaEsp, velocidad, vida, listaAtaques)
5.1.1.5 addPokemon(nuevoPokemon, numRanura):void
5.1.1.6 execSQL(sql2)
5.1.1.7 execSQL(sql4)
    5.1.1.7.1 new ResultadoSQL()
5.1.1.8 next():bool
5.1.1.9 getIDPokemon():Int
5.1.1.10 cambiarID(idPokemon):void
5.1.1.11 execSQL(sql5) ←por cada ataque
5.1.1.12 execSQL(sql6)

6. new Ilustracion12():void
7. mostrarEquipo(username,idequipo):void

```

JSON1: {ataque:Int, ataqueespecial:Int, defensa:Int, defensaespecial:Int, velocidad:Int, vida:Int,
[{NombreAtaque:String}, {}] }

JSON3: {[{NombreEspecie:String, foto:String}, {}] }

SQL1: SELECT e.ataque, e.ataqueespecial, e.defensa, e.defensaespecial, e.velocidad, e.vida,
a.NombreAtaque

FROM Especie AS e, Ataque AS a, Especie_Ataque AS ea

WHERE e.NombreEspecie = %nombreEspecie% AND e.NombreEspecie = ea.NombreEspecie AND
a.NombreAtaque = ea.NombreAtaque

SQL2: INSERT INTO Pokemon(Nombre, Ataque, AtqueEsp, Def, DefEsp, Vel, Vida,
NombreEspecie) VALUES

%nombrePokemon%, %ataque%, %ataqueEsp%, %defensa%, %defensaEsp%, %velocidad%,
%vida%, %nombreEspecie%

SQL3: SELECT NombreEspecie, foto FROM Especie

SQL4: SELECT IDPokemon FROM Pokemon AS p WHERE p.Ataque = %ataque% AND
p.AtaqueEsp = %ataqueEsp% AND p.Def = %defensa% AND p.DefEsp = %defensaEsp% AND p.vel
= %velocidad% AND p.vida = %vida% AND p.NombreEspecie = %nombreEspecie%

SQL5: INSERT INTO Pokemon_Ataque(IDPokemon, NombreAtaque) VALUES %idPokemon%,
%nombreAtaque%

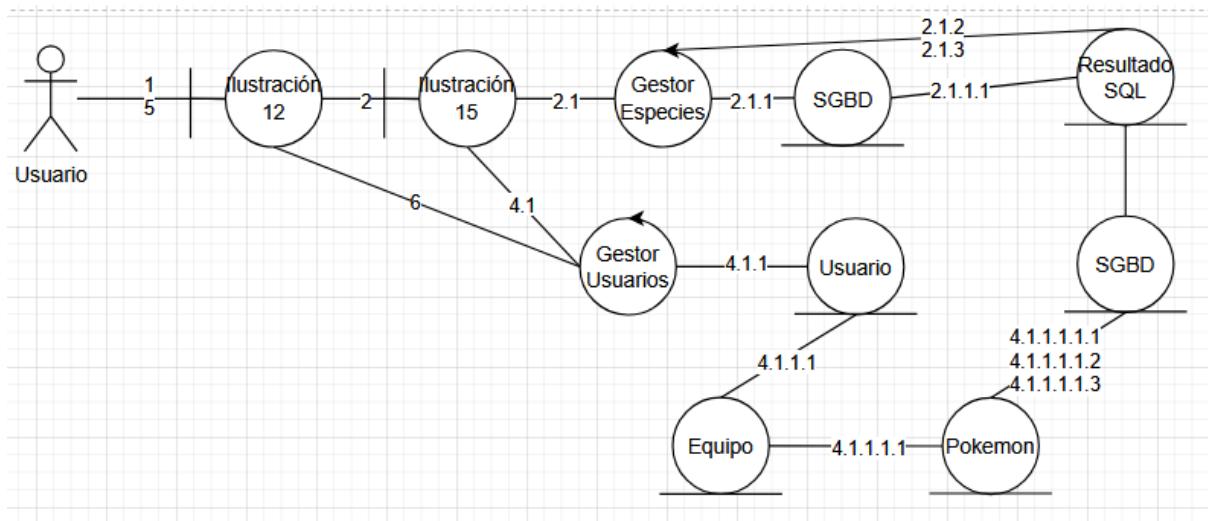
SQL6: INSERT INTO Equipo_Pokemon(IDPokemon, IDEquipo) VALUES %idPokemon%,
%idEquipo%

EXPLICACIÓN:

- PARTE DE LA FUNCIONALIDAD “EDITAR UN EQUIPO”. La idea es que se guardan las ejecuciones que se deben hacer, y al dar click sobre “Guardar” se ejecutan. Si se sale de la interfaz sin guardar, se descartan los cambios y se borran las ejecuciones.

- Abrir la interfaz de edición de equipos, en la que se muestra una lista de todas las especies Pokémon entre las que escoger, y sus respectivas fotos. Para ello, entrar a la base de datos y obtener un JSON con una lista de nombres y fotos de las especies.
- Para añadir un Pokémon, seleccionar una ranura del equipo, y una especie de Pokémon de la lista de especies, en la interfaz gráfica. Si se selecciona el botón añadir, cambia la foto y nombre del Pokémon que se encontraba en la ranura seleccionada, por los de la especie seleccionada, pero solo gráficamente. Cuando se hace click en el botón guardar, se ejecutan los cambios, si no, se cancelan.
- Para guardar los cambios, con el nombre del usuario, el ID del equipo, el nombre de la especie seleccionada, y la ranura del equipo seleccionada, acceder al gestor de usuarios y buscar al usuario.
- Una vez encontrado el usuario, acceder a él con el nombre de la especie seleccionada, el ID del equipo, y el número de la ranura seleccionada. Pedir a la base de datos las estadísticas de la especie, y una lista de los nombres de sus posibles ataques.
- A partir de las estadísticas de la especie, generar unas estadísticas aleatorias para un Pokémon individual, y escoger 4 ataques aleatorios que pueda aprender la especie. Crear un nuevo Pokémon con esta información.
- Acceder al equipo en cuestión y sustituir el nuevo Pokémon por el que haya en la ranura seleccionada.
- Acceder a la base de datos y guardar las estadísticas y ataques del nuevo Pokémon. En la base de datos se genera un ID para el Pokémon, para que sea distinto del resto.
- Acceder a la base de datos para obtener el ID generado. Para ello, se busca el id del Pokémon con las mismas estadísticas generadas aleatoriamente y la misma especie. Técnicamente podría no ser el mismo Pokémon, pero sería muy raro. (se podría evitar añadiendo un gestor de Pokémon).
- Una vez obtenido el ID, se le añade al Pokémon. Además, por cada ataque, añadir una relación de Pokémon_Ataque a la BD, y también añadir una relación de Equipo_Pokémon (hace falta el ID del pokémon para estas relaciones).
- Por último, se regresa a la interfaz de visualización del equipo, y mostrar el equipo dado el ID del equipo y el nombre del usuario (método explicado en anteriores funcionalidades).

3.2.5 Eliminar un Pokémon de un equipo.



1. Desde la interfaz de visualización de un equipo, hacer click en el botón "Editar" y luego en el botón "Sí"

2. new Ilustracion15():void

 2.1 mostrarListaEspecies():void

 2.1.1 execSQL(sql3)

 2.1.1.1 new ResultadoSQL()

 2.1.2 next():bool

 2.1.3 getJSON():JSON3

3. Hacer click en una ranura del equipo Pokémon

4. Hacer click sobre el botón "Expulsar" y luego sobre "Guardar"

 4.1 expulsarPokemonDeEquipo(username, idEquipo, numRanura):void

 4.1.1 expulsarPokemonDeEquipo(idEquipo, nombreEspecie,

 numRanura):void

 4.1.1.1 expulsarPokémon(numRanura):void

 4.1.1.1.1 expulsar(idEquipo):void

 4.1.1.1.1.1 execSQL(sql1):void <--por cada ataque

 4.1.1.1.1.2 execSQL(sql2):void

 4.1.1.1.1.3 execSQL(sql3):void

5. new Ilustracion12():void

6. mostrarEquipo(username,idequipo):void

SQL1: DELETE FROM Pokemon_Ataque WHERE IDPokemon = %idPokemon% AND

NombreAtaque = %nombreAtaque%

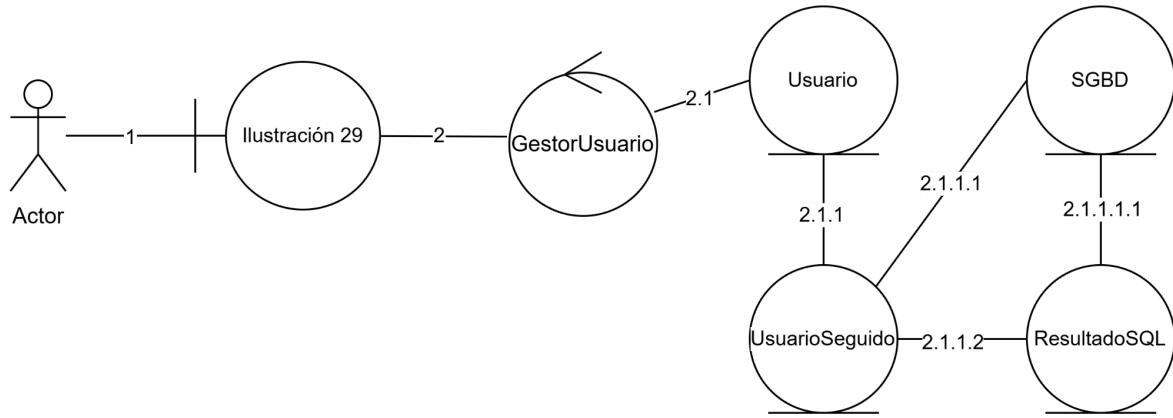
SQL2: DELETE FROM Equipo_Pokemon WHERE IDPokemon = %idPokemon% AND IDEquipo = %idEquipo%

SQL3: DELETE FROM Pokemon WHERE IDPokemon = %idPokemon%

EXPLICACIÓN

- PARTE DE LA FUNCIONALIDAD “EDITAR UN EQUIPO”. La idea es que se guardan las ejecuciones que se deben hacer, y al dar click sobre “Guardar” se ejecutan. Si se sale de la interfaz sin guardar, se descartan los cambios y se borran las ejecuciones.
- Abrir la interfaz de edición de equipos, en la que se muestra una lista de todas las especies Pokémon entre las que escoger, y sus respectivas fotos. Para ello, entrar a la base de datos y obtener un JSON con una lista de nombres y fotos de las especies.
- Seleccionar una ranura del equipo en la interfaz gráfica. Si se selecciona el botón añadir, borra la foto y nombre del Pokémon que se encontraba en la ranura seleccionada, pero solo gráficamente. Cuando se hace click en el botón guardar, se ejecutan los cambios, si no, se cancelan.
- Para expulsar el pokémon, se accede al gestor de usuarios, con un ID de equipo, un nombre de usuario, y un número de ranura (del 1 al 6) que ha sido seleccionado anteriormente. Se encuentra el usuario, y se accede al usuario con el ID del equipo y el número de ranura. Una vez encontrado el equipo, se accede al mismo con el número de ranura y se le indica que modifique el Pokémon que se encuentra ahí.
- Dado el ID del equipo se accede al Pokémon, y se recorren los nombres de sus ataques. Por cada nombre de ataque, se accede a la base de datos y se elimina la relación entre el Pokémon y el ataque (el ataque sigue estando en el gestor de ataques).
- Para eliminar un Pokémon también se usa el id del Pokémon y del equipo. Se accede a la base de datos para eliminar la relación entre ellos dos. Luego se vuelve a acceder a la BD para eliminar el Pokémon en si.
- Por último, se regresa a la interfaz de visualización del equipo, y mostrar el equipo dado el ID del equipo y el nombre del usuario (método explicado en anteriores funcionalidades).

3.3 CHANGELOG



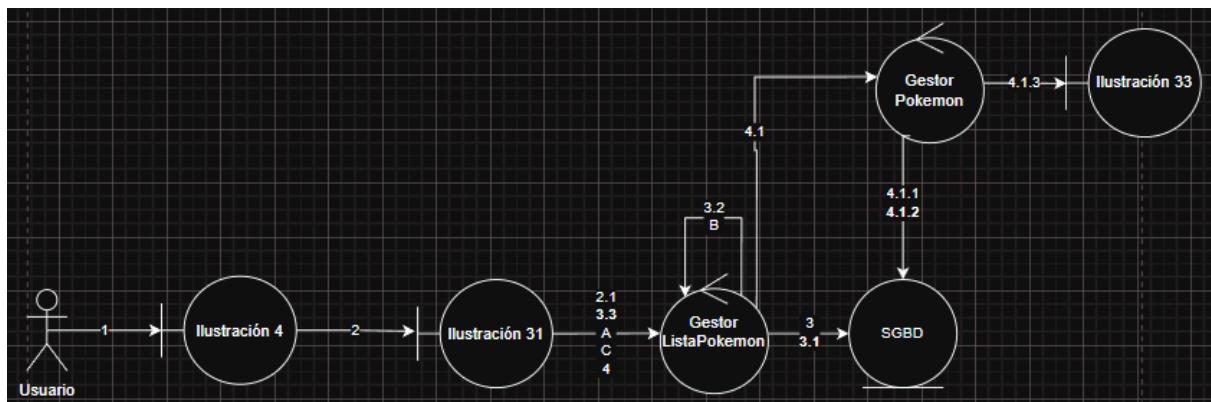
1. new VistaChangelog() (Ilustración 29)
2. obtenerNotificaciones(username): JSON1
 - 2.1. getNotificacionesSeguidos(): JSON1
 - 2.1.1. getNotificaciones(): JSON1
 - 2.1.1.1. execSQL ("SELECT * FROM notificaciones WHERE username = %usernameSeguido% ORDER BY fecha DESC"): ResultadoSQL
 - 2.1.1.1.1. new ResultadoSQL()
 - 2.1.1.2. next()

JSON1: [
 {username: string, fecha: date, tipo: string, texto: string
 },...
]

EXPLICACIÓN:

- Los datos relativos a las listas de usuarios seguidos y seguidores se cargarán al iniciar sesión, por lo que en este apartado estos datos no se consultan en la base de datos. Además, para aplicar los filtros, se devuelven las notificaciones con todos los campos, y será la propia vista (VistaChangelog, ilustración 29) la que decidirá mostrarlos o no. De esta manera, la consulta SQL será la misma independientemente de los filtros seleccionados.
- Al seleccionar la opción de ver el Changelog, se crea una nueva VistaChangelog(), que corresponde a la ilustración 29. GestorUsuario accede a la lista de personas seguidas del usuario que tiene la sesión iniciada (se conoce su *username*).
- Por cada usuario seguido, se busca en la base de datos qué notificaciones ha generado y se ordena por fecha (poniendo las más nuevas primero).
- GestorUsuario se encarga de recopilar todas las listas y devolverlas a la vista.

3.4 LISTA COMPLETA DE POKÉMON Y BÚSQUEDAS CON FILTRO



1. seleccionarVerListaPokemon()
2. new VistaListaPokemon()
 - 2.1. cargarListaPokemon(filtro= " ")
3. getListaPokemonDesdeBD(filtro)
 - 3.1. devolverResultado() : JSONLista
 - 3.2. almacenarListaCompleta(JSONLista)
 - 3.3. mostrarLista(JSONLista)

[SI SE AÑADE UN FILTRO O UNA BÚSQUEDA]

- A. aplicarFiltro(criterio)
- B. filtrarListaCargada(criterio)
- C. actualizarLista(listaFiltrada)

[SI SE PULSA EN UN POKEMON]

4. PokemonSeleccionado(idPokemon)
 - 4.1. solicitarDatosCompletos(idPokemon)
 - 4.1.1. getEstadisticas(idPokemon)
 - 4.1.2. devolverJSONDetalles() : JSONDetalles
 - 4.1.3. new Ilustración33(JSONDetalles)

JSON y sentencias SQL

```
JSONLista = { [ { id : int, nombre : string, imagenSprite : string} ] }
```

```
JSONDetalles = { [ { id : int, nombre : string, descripción : string, tipo : [string, string], ataque : int, defensa: int, velocidad: int, movimientos : [string, ...]
```

Para buscar Pokemon:

```
execSQL1("SELECT name FROM listaPokemon WHERE name ="+ nombrePokemon)
```

Para filtrar Pokemon:

```
execSQL2("SELECT name FROM listaPokemon WHERE type=" + tipoPokemon)
```

```
execSQL3("SELECT name FROM listaPokemon WHERE isLegendary =" + esLegendario)
```

EXPLICACIÓN:

1. Carga Inicial de la Lista (Pasos 1 a 3.3)

La funcionalidad comienza cuando el usuario selecciona la opción "Ver Lista Pokémon" en el Menú Principal (Ilustración 4) (Paso 1). Esta acción inicia el flujo creando la nueva Vista de Lista (VistaListaPokemon / Ilustración 31).

Una vez que la nueva Vista está lista, solicita al GestorListaPokemon (Controlador) que cargue la lista, pasando un filtro vacío inicialmente (Paso 2.1) (que al principio será vacía, es decir se pondrán todos los Pokemon sin filtrado).

El GestorListaPokemon se comunica con el SGBD para obtener la lista de datos básicos (nombre, ID, imagen) a través del método getListPokemonDesdeBD (Paso 3). El SGBD ejecuta la consulta implícita y devuelve el resultado (JSONLista) al Gestor (Paso 3.1).

Luego el GestorListaPokemon:

Almacena la lista completa en memoria (almacenarListaCompleta) (Paso 3.2). Esta lista se mantendrá viva para futuras búsquedas, para así no tener que estar todo el rato buscando en la base de datos. Muestra la lista (mostrarLista) en la Ilustración 31, renderizando los Pokémon básicos en la interfaz gráfica (Paso 3.3).

2. Busqueda y filtrado (Pasos A, B, C)

Esta secuencia se activa cada vez que el usuario utiliza la barra de búsqueda o aplica un filtro en la Ilustración 31.

La Vista captura el criterio introducido por el usuario y lo pasa al GestorListaPokemon mediante el mensaje aplicarFiltro (Paso A).

El GestorListaPokemon ejecuta el método filtrarListaCargada (Paso B). En este punto, el Gestor trabaja únicamente con la lista que ya tiene en la memoria (almacenada en 3.2), reduciendo la lista original a una listaFiltrada que coincide con el criterio (borrando Pokemons que no cumplen los filtros).

Finalmente, el Gestor notifica a la Vista (actualizarLista) para que se redibuje, mostrando únicamente los Pokémon de la listaFiltrada (Paso C).

3. Visualización de los Detalles (Pasos 4 a 4.1.3)

Esta secuencia se ejecuta cuando el usuario hace clic en un Pokémon de la lista para ver su información completa.

La Vista (Ilustración 31) notifica al GestorListaPokemon sobre la selección, pasándole el identificador único (idPokemon) del Pokemon pulsado (Paso 4).

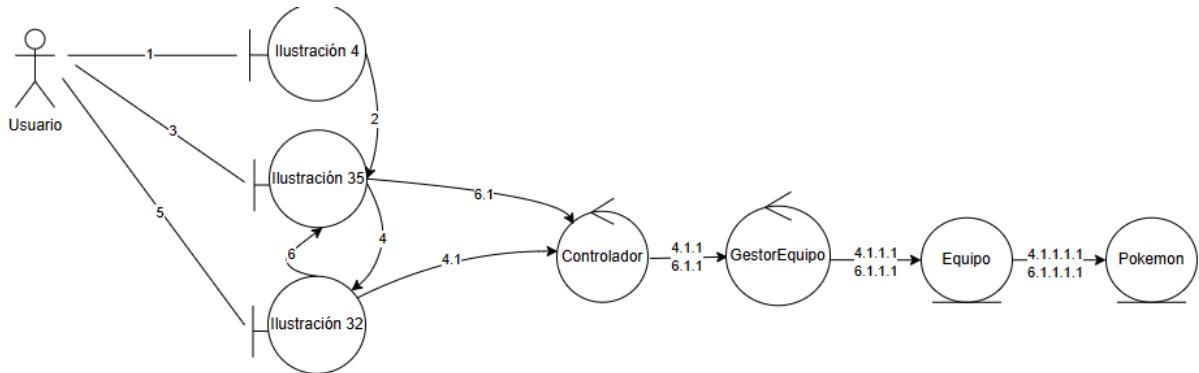
El GestorListaPokemon no tiene los detalles de cada Pokemon, por lo que lo hace el GestorPokemon (Paso 4.1).

El GestorPokemon accede directamente al SGBD solicitando todos los datos detallados (estadísticas, movimientos, habilidades, etc.) mediante getEstadisticas (Paso 4.1.1). El SGBD devuelve el objeto completo con todos los atributos (JSONDetalles) al GestorPokemon (Paso 4.1.2).

Finalmente, el GestorPokemon toma los datos recibidos y crea la nueva Vista de Especificaciones (Ilustración 33), pasándole el objeto JSONDetalles para que se renderice en pantalla (Paso 4.1.3).

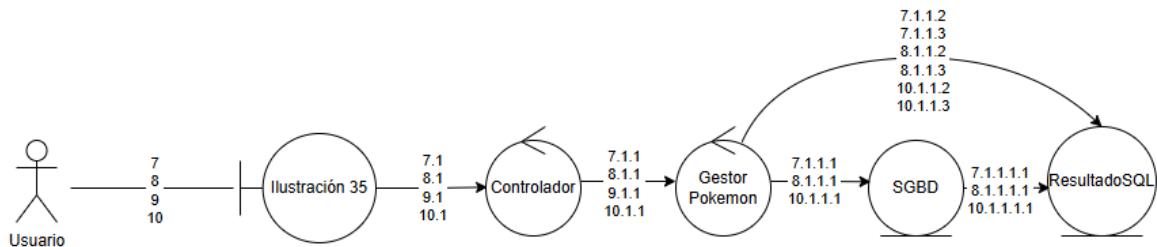
3.5 CHATBOT

3.5.1 Seleccionar equipo para obtener mejor



1. El usuario pulsa el botón "ChatBot"
2. newFigura35()
3. El usuario pulsa el botón "Seleccionar Equipo"
4. newFigura32()
 - 4.1 obtenerEquiposNombreySprite(usuario): JSON1
 - 4.1.1. obtenerEquiposNombreySprite(usuario): JSON1
 - 4.1.1.1 obtenerPokemonNombreySprite(): JSON2
 - 4.1.1.1.1 obtenerNombreySprite (): JSON3
5. El usuario pulsa el botón "Seleccionar Equipo"
6. newFigura35(nombreEquipo)
 - 6.1 obtenerMejorPokemon(nombreEquipo): JSON3
 - 6.1.1 obtenerMejorPokemon(nombreEquipo): JSON3
 - 6.1.1.1 obtenerMejorPokemon(): JSON3
 - 6.1.1.1.1 obtenerMediaPokemon(): JSON4

3.5.2 Seleccionar pokemon para hacer diferentes consultas



7. El usuario pulsa el botón "Seleccionar Pokémon" y en el desplegable escribe el nombre de un Pokémon correcto:

7.1 obtenerPokemon(string): JSON3

 7.1.1 obtenerPokemonEntero(string): JSON3

 7.1.1.1 execSQL(SQL1): ResultadoSQL

 7.1.1.1.1 newResultadoSQL()

 7.1.1.2 next()

 7.1.1.3 obtenerPokemonJSON(): JSON5

8. El usuario pulsa el botón "Ver cadena evolutiva"

 8.1 obtenerCadenaEvolutiva(string): JSON6

 8.1.1 obtenerCadenaEvolutiva(string): JSON6

 8.1.1.1 execSQL(SQL2): ResultadoSQL

 8.1.1.1.1 newResultadoSQL()

 8.1.1.2 next()

 8.1.1.3 obtenerFotoEvolucionesJSON(): JSON7

9. El usuario pulsa el botón "Ver estadísticas"

 9.1 obtenerEstadisticas(): JSON5

 9.1.1 obtenerEstadisticas(): JSON5

10. El usuario pulsa el botón "Ver rivales"

 10.1 obtenerRivales(string): JSON6

 10.1.1 obtenerRivales(string): JSON6

 10.1.1.1 execSQL(SQL3): ResultadoSQL

 10.1.1.1.1 newResultadoSQL()

 10.1.1.2 next()

 10.1.1.3 obtenerFotoJSON(): JSON7

3.5.3 JSON y consultas SQL

JSON1: { "equipo1" : JSON2,
.....
"equipoN" ; JSON2 }

JSON4: { "nombre": "string",
"foto": "image",
"media": "integer" }

SQL1: SELECT * FROM ESPECIE
WHERE NombreEspecie = %nombre%

JSON2: { "nombreEquipo" : "string"
"pokemon1": JSON3,
.....
"pokemonN": JSON3}

JSON5 { "nombre" : "string"
.....
"shiny" : "boolean"}

SQL2: SELECT previo FROM ESPECIE
WHERE NombreEspecie = %nombre%

JSON3: { "nombre": "string",
"foto": "image" }

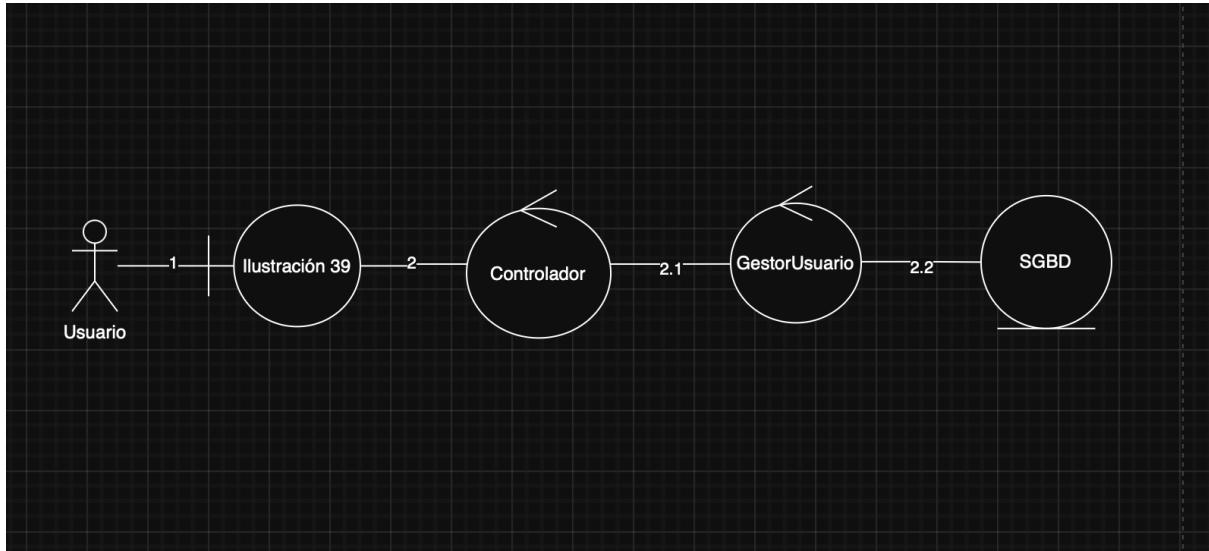
JSON6: { "pokemon1" : image,
.....
"pokemonN" ; image }

SQL3:
SELECT image
FROM ESPECIE
WHERE NombreEspecie = (
SELECT nombreEspecie
FROM Especie_Tipo
NATURAL JOIN TIPO
NATURAL JOIN AFECTADO
ORDER BY multiplo DESC
LIMIT 1
)
LIMIT 3;

JSON7: { "pokemon" : image }

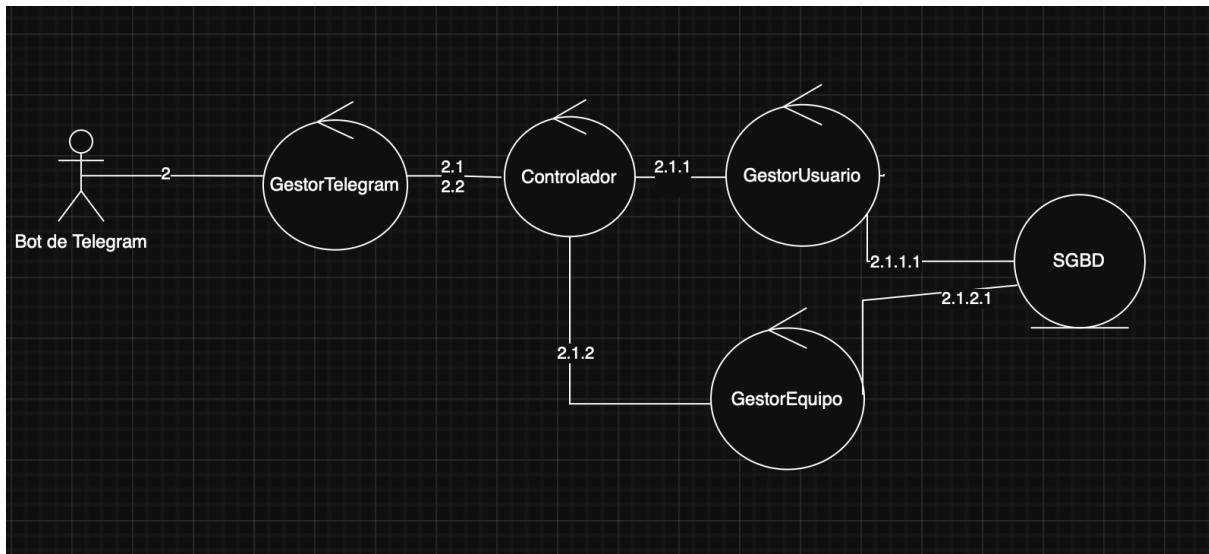
3.6 COMPARTIR A TRAVÉS DE TELEGRAM

Vincular telegram con usuario



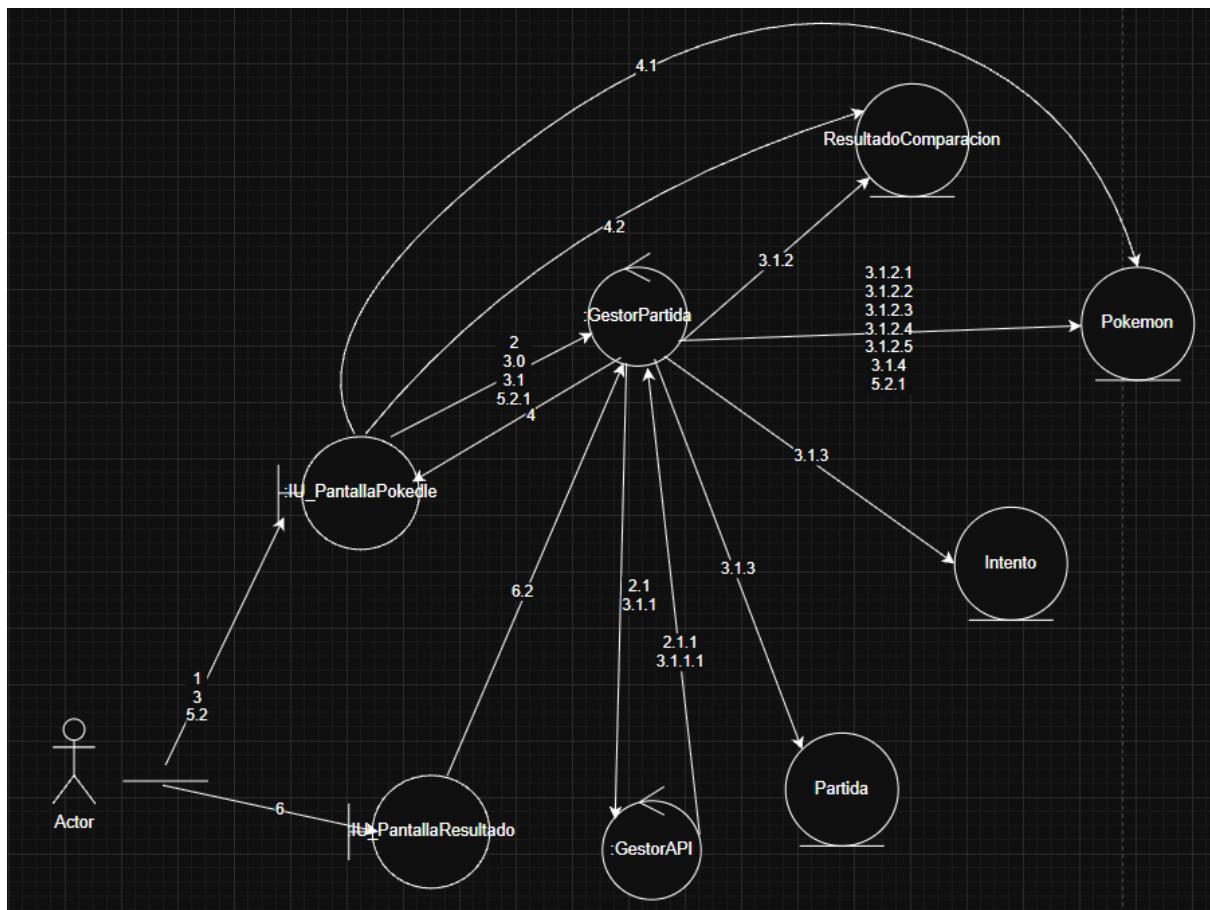
1. Un usuario ya identificado pincha en "Vincular cuenta", ingresa un nombre de telegram (que debería ser su username en Telegram) y pincha en el botón "continuar"
2. `vincularUsuario(username, telegramUsername)`
 - 2.1 `vincularUsuario(username, telegramUsername)`
 - 2.2 `vincularUsuario(username, telegramUsername)`

Compartir equipo telegram



1. El bot de telegram lee su comando característico
2. `commando1(telegramUsername)`
 - 2.1.1 `getUserByTelegram(tgUser): string`
 - 2.1.1.1 `getUserByTelegram(tgUser): string`
 - 2.1.2 `getEquipo(username): JSON`
 - 2.1.2.1 `getEquipo(username): JSON`

3.7 POKEDLE



1: Usuario pulsa “Jugar Pokedle”

2: IniciarPartida()

2.1: pokemonObjetivo = ElegirPokemonAleatorio()

2.1.1: getPokemonAleatorioAPI() : Pokemon

Se obtiene un Pokémon aleatorio de la API

3: Usuario introduce un nombre y pulsa ENTER

3.0: validarIntento(nombreUsuario)

→ Si ya existe en intentosPrevios → mostrar error y terminar el flujo

→ Si no existe → añadirlo a la lista de intentos

3.1: resultado = ComprobarAdivinanza(nombreUsuario)

3.1.1: pokemonJugador = BuscarPokemon(nombreUsuario) (llama a :GestorApi)

3.1.1.1: getPokemonAPI(nombreUsuario) : Pokemon

Petición a la API para obtener el pokemon introducido por el usuario

3.1.2: coincidencias = CompararPokemon(pokemonJugador, pokemonObjetivo)

3.1.2.1: compararTipos() : boolean[]

3.1.2.2: compararGeneracion() : boolean

3.1.2.3: compararAltura() : boolean

3.1.2.4: compararPeso() : boolean

3.1.2.5: compararColor() : boolean

Devuelve qué atributos coinciden para pintar verde/rojo las casillas

3.1.3: actualizarHistorialIntentos(pokemonJugador, coincidencias)

Se muestra la tabla de intentos en pantalla

3.1.4: esCorrecto = comprobarVictoria(pokemonJugador, pokemonObjetivo)

Determina si la partida termina

4: MostrarResultadoIntento()

4.1: mostrarPokemonAdivinado(pokemonJugador)

4.2: colorearCasillas(coincidencias)

5: Usuario decide continuar o rendirse

5.1: Si continúa, vuelve a paso 3

5.2: Si pulsa Rendirse()

5.2.1: revelarPokemonCorrecto(pokemonObjetivo)

6: Pantalla final

6.1: Usuario pulsa VolverMenuPrincipal()

vuelve al menú

o

6.2: Usuario pulsa JugarDeNuevo()

llama de nuevo a IniciarPartida()

EXPLICACIÓN:

El flujo comienza cuando el usuario pulsa “Jugar Pokedle”, lo que llama al método IniciarPartida(). Este método selecciona un Pokémon objetivo mediante getPokemonAleatorioAPI(), obteniendo sus datos directamente desde la API y almacenándolo como pokemonObjetivo.

Cada vez que el usuario introduce un nombre de Pokémon y pulsa ENTER, el sistema primero ejecuta validarIntento(nombreUsuario) para comprobar que no haya sido introducido antes. Si ya existe, se muestra un error; si no, se añade a la lista de intentos.

A continuación, se llama a ComprobarAdivinanza(nombreUsuario), que obtiene los datos del Pokémon introducido por el usuario a través de getPokemonAPI(nombreUsuario) y lo convierte en un objeto Pokemon. Este objeto se compara con el Pokémon objetivo mediante distintos métodos (compararTipos(), compararGeneracion(), compararAltura(), compararPeso() y compararColor()), obteniendo como resultado qué atributos coinciden.

El intento y sus resultados se almacenan con actualizarHistorialIntentos() y se comprueba si el usuario ha ganado mediante comprobarVictoria().

Finalmente, el resultado se muestra en la interfaz, coloreando las casillas según las coincidencias. El usuario puede seguir intentándolo, rendirse (mostrando el Pokémon correcto) o iniciar una nueva partida.