

Proyecto MARCODEX

Equipo: Liviu y sus Marcos

Liviu Deleanu

Gorka Bidaguren Ibañez

Tabata Morente Arnaiz

Laura Calvo Posada

Marco Lartategui Ugarte

Iker Fuente Otero

05-10-2025

Índice

1	Introducción	3
2	Diagrama de Casos de Uso y Jerarquía de actores	3
2.1	Actores	3
3	Casos de Uso Extendidos	4
3.1	Gestión de usuarios (Liviu)	4
3.2	Crear equipos Pokemon (Tabata)	15
3.3	ChangeLog (Gorka)	20
3.4	Lista Pokemon y filtros (Laura)	22
3.5	Chatbot (Iker)	24
3.6	Ranking (Marco)	27
4	Modelo de Dominio	30
4.1	Introducción	30
4.2	Usuario	31
4.3	Equipo	31
4.4	Changelog	31
4.5	Pokédex	31
4.6	Especie Pokémon	31
4.7	Pokémon	31
4.8	Ataques	32
4.9	Tipo	32
4.10	Chatbot	32
5	Plan de Pruebas	34
5.1	Gestión Usuarios (Gorka)	34
5.2	Crear Equipos Pokémon (Iker)	39
5.3	Changelog (Marco)	42
5.4	Lista Pokemon y filtros (Liviu)	45
5.5	Chatbot (Tabata)	49
5.6	Ranking (Laura)	51
6	Bases de Datos	53
7	Diagrama de clases	54
8	Diagramas de comunicación	55
8.1	Gestión de usuarios (Liviu)	55
8.2	Crear equipos Pokemon (Tabata)	63
8.3	ChangeLog (Gorka)	70
8.4	Lista Pokémon y filtros (Laura)	73
8.5	Chatbot (Iker)	75
8.6	Ranking (Marco)	77
9	Diagramas de secuencia	78
9.1	Gestión de usuarios (Liviu)	78
9.2	Crear equipos Pokemon (Tabata)	97
9.3	ChangeLog (Gorka)	103
9.4	Lista Pokémon y filtros (Laura)	106
9.5	Chatbot (Iker)	108
9.6	Ranking (Marco)	110
10	Problemas Encontrados	119
11	Conclusiones	120

1 Introducción

En este proyecto se nos ha encomendado desarrollar un sistema Pokédex. Nuestro equipo está conformado por seis integrantes, por lo que vamos a describir e implementar seis funcionalidades; las cinco funcionalidades obligatorias y una adicional. Para la funcionalidad extra, hemos creado un ranking que ordena todos los equipos Pokémon de los usuarios por rareza.

Para este primer hito hemos elaborado un esquema general del sistema, centrandonos principalmente en los casos de uso y dejando un poco de lado las funcionalidades secundarias. Cada caso de uso ha sido descrito de manera individual y se ha añadido un esquema mostrando las relaciones. Además, se ha ilustrado un concepto inicial de las interfaces gráficas de cada caso de uso.

2 Diagrama de Casos de Uso y Jerarquía de actores

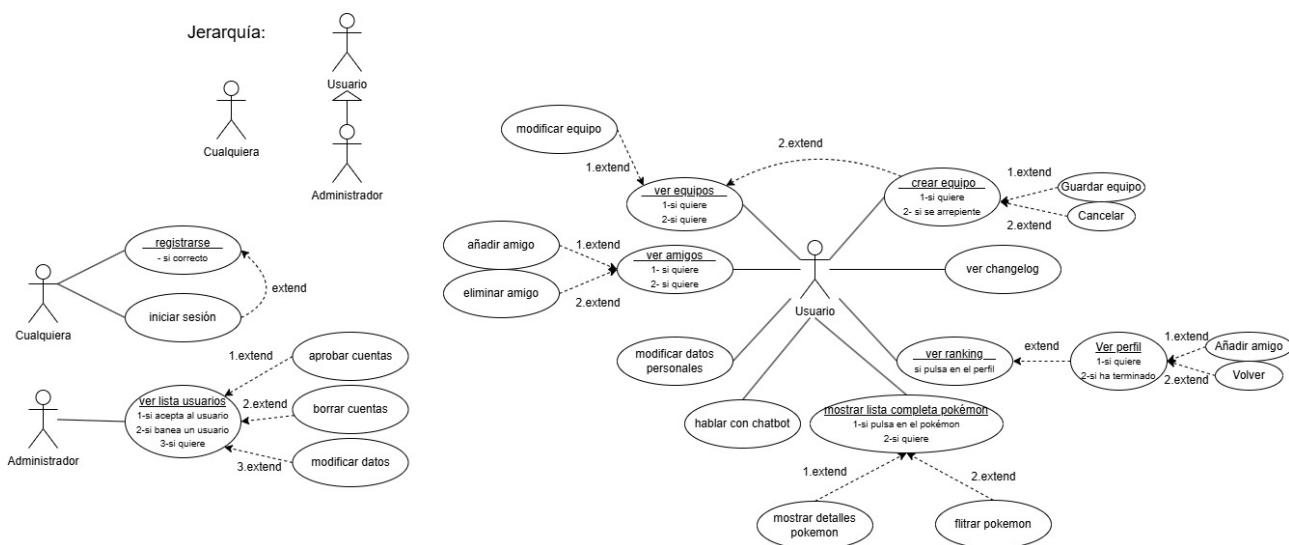


Figure 1: Diagrama completo Casos de Uso

2.1 Actores

Cualquiera:	Persona que ingresa por primera vez en la aplicación y no tiene sesión iniciada.
Usuario:	Persona que ingresa en la aplicación y ha iniciado sesión correctamente
Administrador:	Persona que ingresa en la aplicación y ha iniciado sesión correctamente con la cuenta de administrador.

3 Casos de Uso Extendidos

3.1 Gestión de usuarios (Liviu)



Figure 2: Diagrama registrarse

Nombre:	Registrarse
Descripción:	Permite al actor Cualquiera (persona que ingresa por primera vez en la aplicación y no tiene sesión iniciada) crear una cuenta en el sistema, proporcionando sus datos de registro. Una vez registrado con éxito, se extiende el caso de uso Iniciar Sesión y, al iniciar sesión correctamente, el actor pasa a ser Usuario , accediendo a la página principal de la aplicación.
Actores:	Cualquiera
Precondiciones:	El actor debe ingresar por primera vez o no contar con un usuario creado, y no estar con una sesión iniciada.
Requisitos no funcionales:	Las contraseñas deben almacenarse como hash, usando un algoritmo resumen.
Flujo de Eventos:	<p>1.- El actor Cualquiera selecciona el botón “Registrarse” en la pantalla inicial. (<i>Figure 3</i>)</p> <p>2.- El sistema muestra un modal con los campos requeridos: Nombre, Apellido, Nombre de usuario, Contraseña y Correo. (<i>Figure 4</i>)</p> <p>3.- El actor completa los campos y pulsa “Registrarse”. (<i>Figure 4</i>)</p> <p>4.- El sistema valida que:</p> <ul style="list-style-type: none"> 4a.- El nombre de usuario y el correo no existan previamente. 4b.- La contraseña cumpla con las políticas mínimas de seguridad. 4c.- Ningún campo obligatorio esté vacío. <p>[Si algún campo no es válido:]</p> <ul style="list-style-type: none"> 4a.- El sistema muestra un mensaje de error indicando qué campo debe corregirse. El flujo regresa al paso 2. <p>5.- [Si el registro es correcto:] El sistema crea la nueva cuenta en la base de datos (EXTEND Iniciar Sesión).</p>
Postcondiciones:	<ul style="list-style-type: none"> • Si el registro es exitoso, se crea un nuevo usuario en el sistema y este queda identificado en la aplicación. • Si el registro falla, el usuario permanece como Cualquiera.

Interfaz Gráfica:



Figure 3: Pantalla Inicial

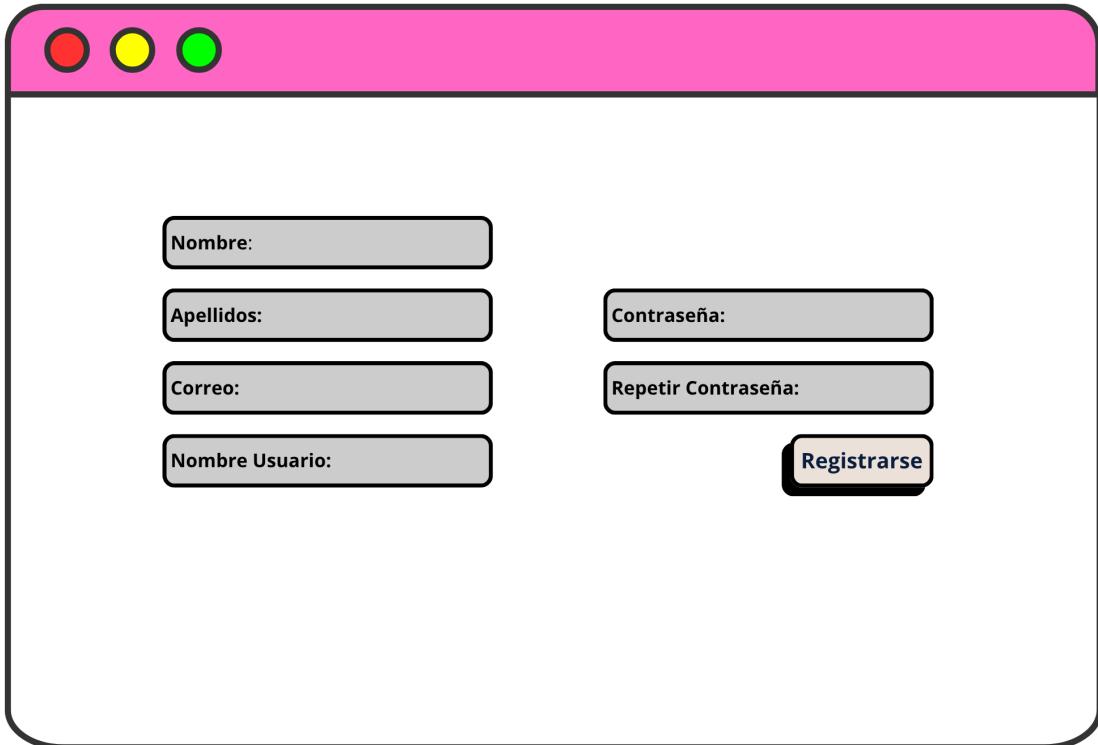


Figure 4: Pantalla Registrarse

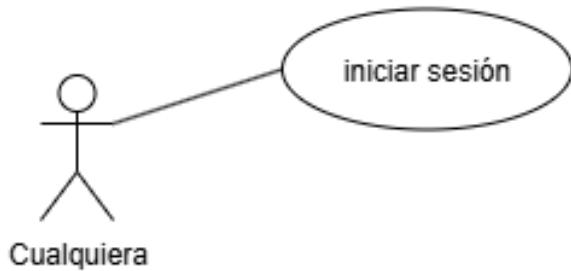


Figure 5: Diagrama Iniciar sesión

Nombre:	Iniciar Sesión
Descripción:	Permite al actor Cualquiera acceder al sistema mediante la introducción de su nombre de usuario y contraseña. Si las credenciales son correctas, el actor pasa a ser Usuario y se le redirige a la página principal de la aplicación.
Actores:	Cualquiera
Precondiciones:	No tener una sesión activa en el sistema.
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<p>1.- El sistema muestra un modal con los campos requeridos: Nombre de Usuario y Contraseña. (<i>Figure 6</i>)</p> <p>2.- El actor introduce los datos y pulsa “Iniciar Sesión”.</p> <p>3.- El sistema valida que:</p> <ul style="list-style-type: none"> • Ningún campo esté vacío. • El nombre de usuario exista en la base de datos. • La contraseña coincida con la registrada. <p>[Si algún campo está vacío:]</p> <p>3a.- El sistema muestra un mensaje de error indicando el campo que falta. El flujo regresa al paso 2.</p> <p>[Si el nombre de usuario no existe o la contraseña es incorrecta]</p> <p>3b.- El sistema muestra un mensaje de error: “Usuario o contraseña incorrectos”. El flujo regresa al paso 2.</p> <p>4.- [Si las credenciales son correctas] El sistema inicia la sesión y convierte al actor Cualquiera en Usuario.</p> <p>5.- El sistema redirige al Usuario a la página principal de la aplicación.</p>
Postcondiciones:	<ul style="list-style-type: none"> • Si la validación es exitosa, el actor queda identificado como Usuario dentro del sistema.

Interfaz Gráfica:



Figure 6: Pantalla Inicial

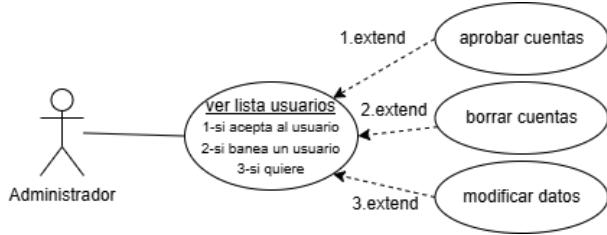


Figure 7: Diagrama Ver lista usuarios

Nombre:	Ver Lista de Usuarios
Descripción:	Permite al Administrador acceder a la lista completa de usuarios registrados no verificados en el sistema. Desde esta vista, el administrador podrá aprobar estas cuentas, borrar cuentas o modificar datos de una cuenta. Importante aclarar que sólo serán de las cuentas que no estén verificadas. Estas funcionalidades se consideran casos de uso secundarios que se extienden desde este caso principal.
Actores:	Administrador
Precondiciones:	Ninguna.
Requisitos no funcionales:	Los datos de los usuarios deben mostrarse de manera segura y sin exponer contraseñas.
Flujo de Eventos:	<p>1.- El Administrador selecciona el botón “Ver Lista de Usuarios” en el menú principal. (<i>Figure 8</i>)</p> <p>2.- El sistema consulta la base de datos y muestra la lista completa de usuarios. (<i>Figure 9</i>)</p> <p>3.- El administrador visualiza los botones de aprobar cuenta, borrar o modificar datos por cada usuario:</p> <ul style="list-style-type: none"> [Si el administrador acepta al usuario] <ul style="list-style-type: none"> • Aprobar cuentas (<i>EXTEND Aprobar Cuentas</i>). [Si el administrador banea a un usuario] <ul style="list-style-type: none"> • Borrar cuentas (<i>EXTEND Borrar Cuentas</i>). [Si el administrador quiere] <ul style="list-style-type: none"> • Modificar datos(<i>EXTEND Modificar Datos</i>).
Postcondiciones:	<ul style="list-style-type: none"> • El administrador accede a la lista completa de usuarios y puede realizar acciones de gestión. • Si se ejecutan los casos de uso extendidos, los cambios se verán reflejados en la base de datos, y en la lista actualizada.

Interfaz Gráfica:

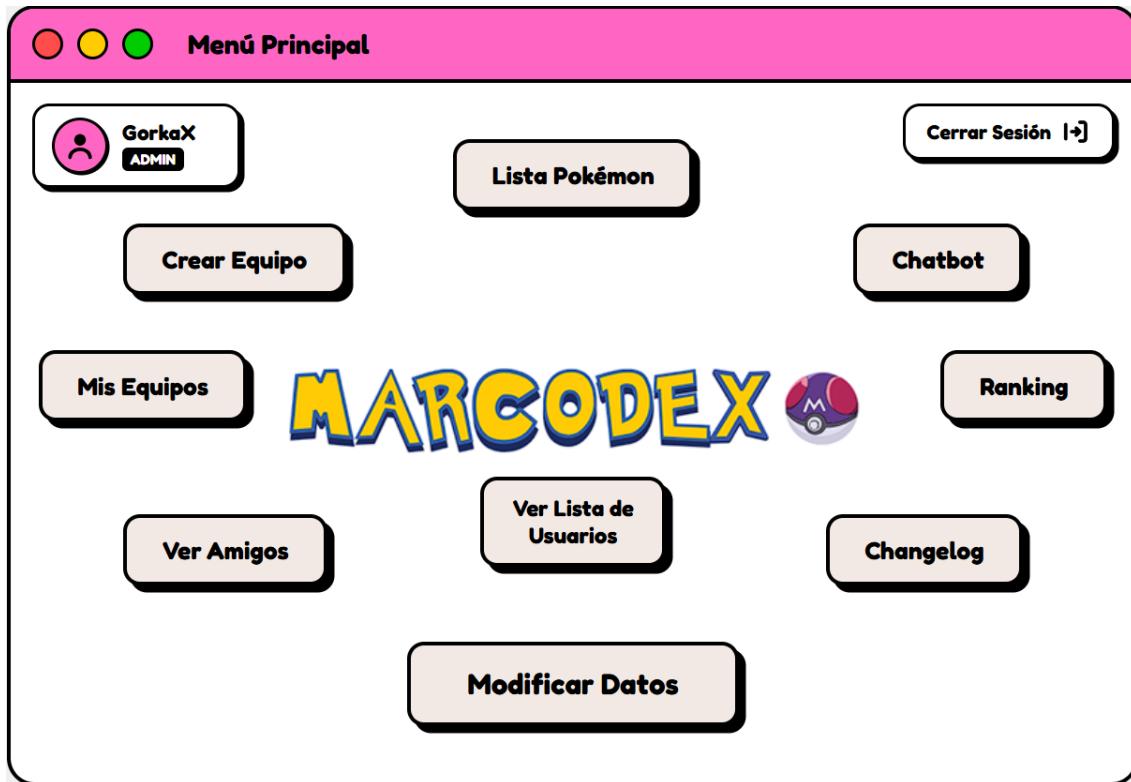


Figure 8: Pantalla Administrador

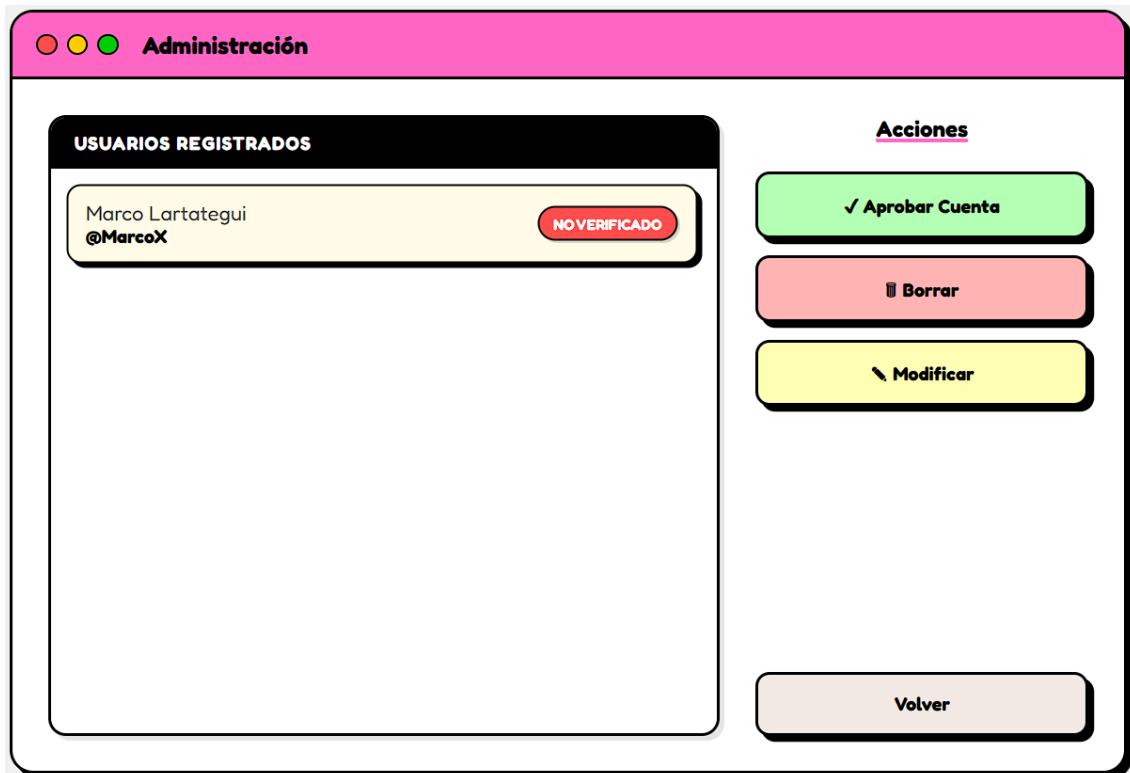


Figure 9: Pantalla Admin aprobar/borrar/modificar

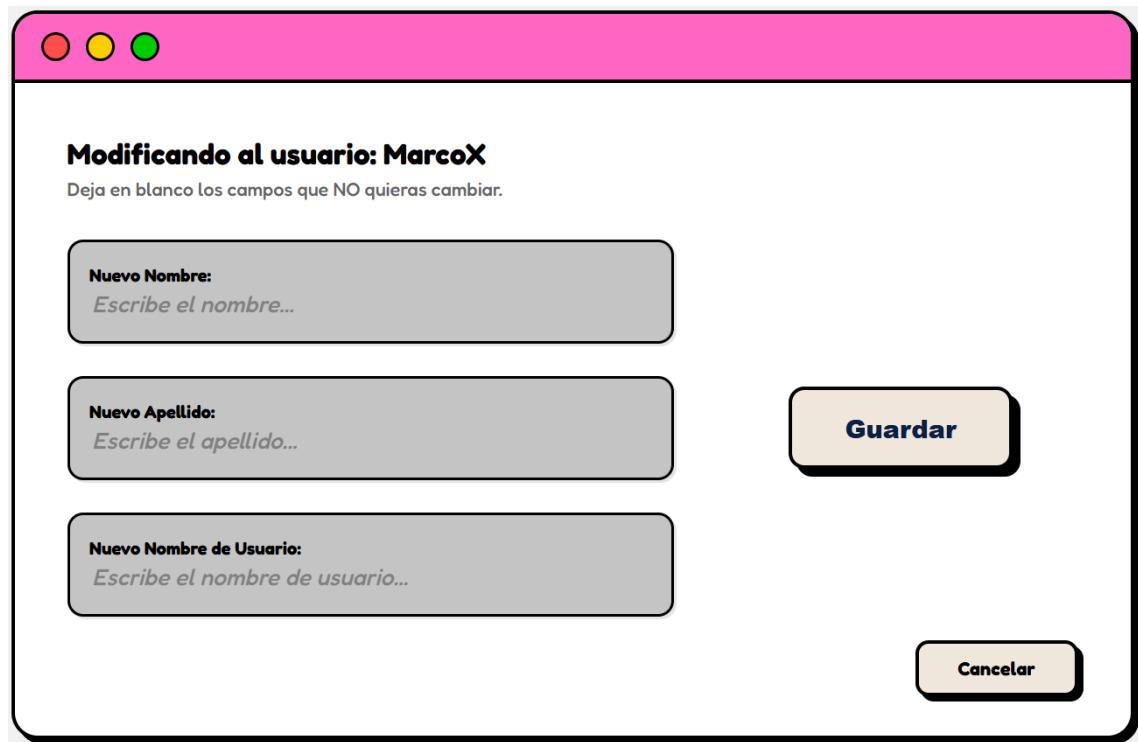


Figure 10: Pantalla Administrador Modificar Datos

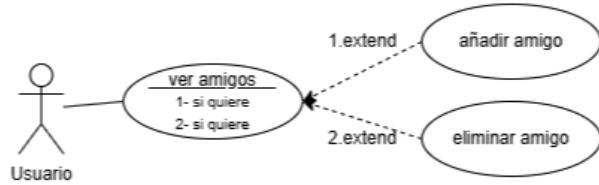


Figure 11: Diagrama Ver amigos

Nombre:	Ver Amigos
Descripción:	Permite al Usuario acceder a su lista de amigos para consultarla y poder gestionarla. El Usuario podrá eliminar y/o añadir amigos de su lista.
Actores:	Usuario
Precondiciones:	Ninguna.
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<p>1.- El Usuario selecciona el botón “Mis Amigos” en la aplicación. (<i>Figure 11</i>)</p> <p>2.- El sistema consulta la base de datos y muestra la lista completa de amigos del usuario (si existen). (<i>Figure 12</i>)</p> <p>3.- En la lista, al lado de cada amigo, aparece la opción “Eliminar” (<i>EXTEND Eliminar Amigo</i>).</p> <p>4.- En la misma vista, aparece un botón “Añadir Amigo”.</p> <ul style="list-style-type: none"> • [Si el usuario pulsa “Añadir Amigo”] <p>Se abre un buscador para introducir el nombre de usuario (<i>EXTEND Añadir Amigo</i>). (<i>Figure 13</i>)</p> <p>5.- El usuario puede realizar estas acciones secundarias de gestión o simplemente visualizar la lista.</p> <p>6.- En cualquier momento, el usuario puede volver a la pantalla principal de la aplicación.</p>
Postcondiciones:	<ul style="list-style-type: none"> • El usuario accede a la lista de amigos y puede gestionarlos. • Si se ejecutan los casos de uso extendidos (Añadir o Eliminar), la lista se actualiza en tiempo real con los cambios aplicados, también guardándose en la base de datos.

Interfaz Gráfica:

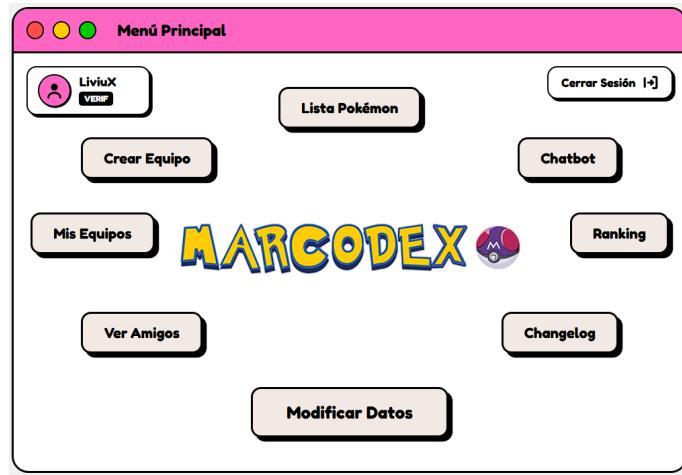


Figure 12: Menú Principal



Figure 13: Ver Amigos

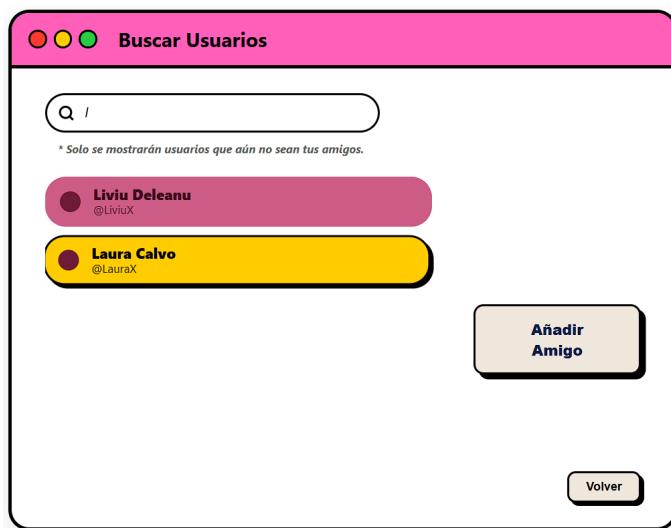


Figure 14: Añadir Amigo

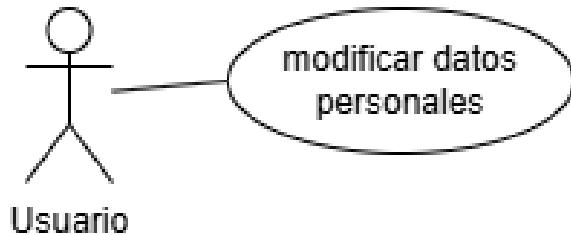


Figure 15: Diagrama Modificar datos personales

Nombre:	Modificar datos personales
Descripción:	<p>Permite al Usuario cambiar los datos anteriormente introducidos al registrarse. Desde esta vista el Usuario podrá modificar el Nombre, Apellido, Nombre de usuario, Contraseña y Correo; para cambiar el nombre de usuario o la contraseña se pedirá volver a introducir la contraseña.</p> <p>Además, se comprobará que el nombre de usuario introducido no exista en la base de datos y que el correo electrónico tenga un formato válido. En caso de errores o conflictos, se mostrará un mensaje de error.</p>
Actores:	Usuario
Precondiciones:	Ninguna.
Requisitos no funcionales:	<ul style="list-style-type: none"> La contraseña no debe mostrarse en texto plano en la interfaz de modificar datos. La comunicación con la base de datos debe realizarse correctamente.
Flujo de Eventos:	<ol style="list-style-type: none"> El Usuario pulsa el botón de “Modificar Datos Personales” en el menú principal. (<i>Figure 15</i>) El sistema analiza la base de datos y obtiene los datos del perfil actual. Muestra los datos por pantalla y da la opción de modificarlos. (<i>Figure 16</i>) Después de que el Usuario modifique los datos que quiera, deberá pulsar el botón “Guardar” de la parte inferior de la pantalla. (<i>Figure 16</i>) Si el Nombre de usuario o la Contraseña han sido modificados se le pedirá al Usuario que introduzca la contraseña que se ha utilizado para iniciar sesión. (<i>Figure 17</i>) [Si el Usuario pulsa sobre ”Volver”:] <p>5a.- Devuelve al Usuario al Menú Principal.</p>
Postcondiciones:	<ul style="list-style-type: none"> Al pulsar el botón de “Guardar” los datos se quedarán modificados en la pantalla actual y se modificarán en la base de datos.

Interfaz Gráfica:

The screenshot shows a window titled 'Modificar datos personales'. It contains four input fields: 'Nombre' (Nuevo Nombre), 'Apellido' (Nuevos Apellidos), 'Correo' (Nuevo Correo), and 'Nombre Usuario' (Nuevo Usuario). There are two buttons at the bottom right: 'Guardar' and 'Volver'.

Figure 16: Modificar datos personales

The screenshot shows a window titled 'Seguridad Marcodex'. It displays a message: 'Estás intentando modificar datos sensibles (Usuario o Contraseña). Por favor, confirma tu contraseña actual para guardar los cambios.' Below this is a password input field labeled 'Tu contraseña actual'. At the bottom are two buttons: 'Cancelar' and 'Confirmar y Guardar'.

Figure 17: Introducir contraseña actual

3.2 Crear equipos Pokemon (Tabata)

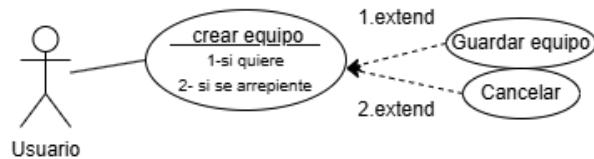


Figure 18: Diagrama Crear equipo

Nombre:	Crear equipo
Descripción:	Permite a los usuarios crear sus propios equipos Pokémon.
Actores:	Usuario
Precondiciones:	Ninguna.
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<p>1.- El Usuario selecciona en su interfaz gráfica la opción de Crear Equipo. (<i>Figure 12</i>)</p> <p>2.- Al usuario se le mostrará la lista completa de Pokémon para que pueda ir eligiéndolos y añadiéndolos o borrándolos si lo desea. (<i>Figure 19</i>)</p> <p>3.- [Si el usuario ha elegido 6 pokémon:]</p> <p>3a.- [Si el usuario pulsa en “Guardar”:] (<i>EXTEND Guardar equipo</i>)</p> <p> a.- Se guarda el equipo en la base de datos y devuelve al Usuario al Menú Principal.</p> <p>3b.- [Si el usuario pulsa en “Cancelar”:] (<i>EXTEND Cancelar</i>)</p> <p> a.- Se devolverá al Usuario al Menú Principal.</p>
Postcondiciones:	<ul style="list-style-type: none"> • El equipo se ha guardado correctamente y el usuario podrá acceder a él en un futuro. • Si el equipo no se ha creado, no hay postcondiciones.

Interfaz Gráfica:

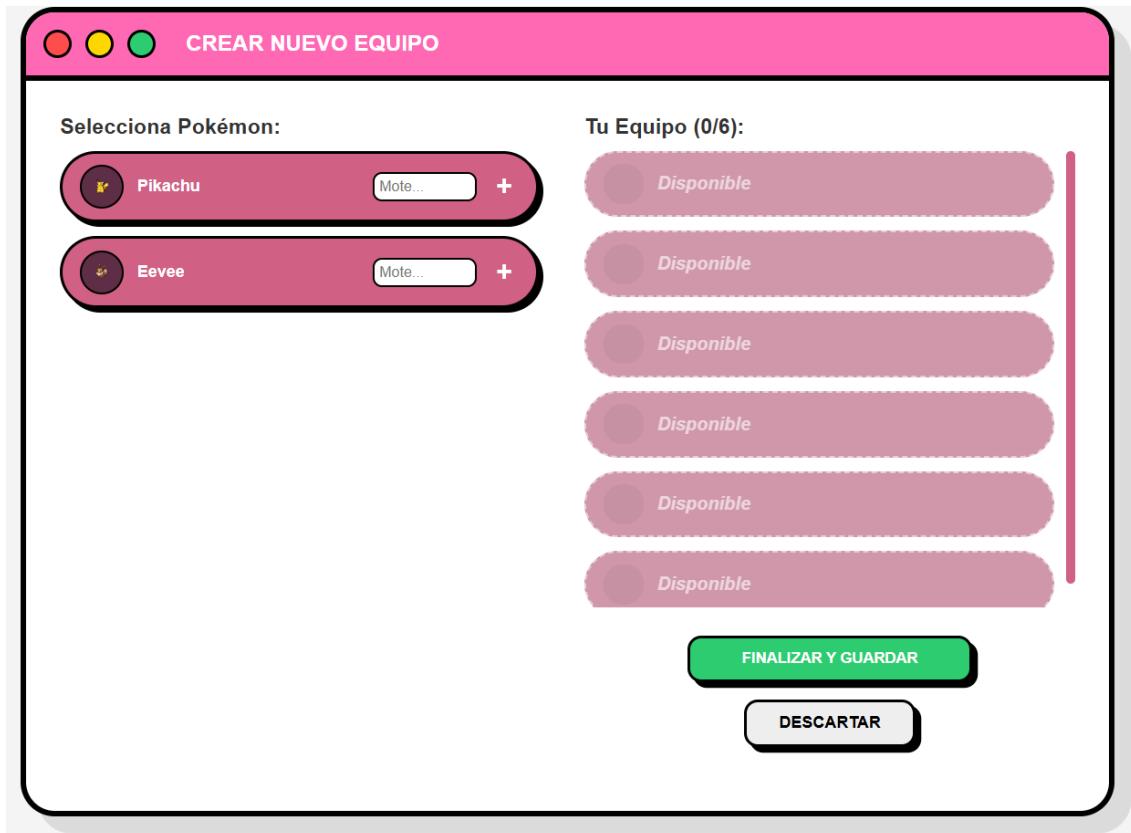


Figure 19: Crear Equipo

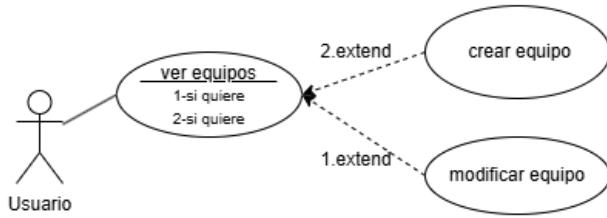


Figure 20: Diagrama Ver equipo

Nombre:	Ver equipos
Descripción:	Permite visualizar al Usuario los equipos que ha creado. En caso de que no tenga ninguno, le saldrá en pantalla un mensaje de error indicando esto mismo. En el caso de que sí tenga alguno creado, aparte de aparecer en una lista el/los equipo/s, habrá un botón de crear equipo , por si el Usuario quiere crear otro, y un botón de Volver al Menú . Además si pulsa sobre uno de los equipos del listado se mostrará los Pokémon del equipo elegido junto con algunos datos más; como el nombre personalizado, la especie a la que pertenece, si es shiny o no y una foto del pokémon. En esta pantalla también habrá un botón de Modificar Equipo por si el usuario quisiera eliminar algún Pokémon y añadir otro.
Actores:	Usuario
Precondiciones:	Ninguna
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<ol style="list-style-type: none"> 1.- El Usuario selecciona en el Menú Principal el botón de “Mis equipos” para ver la lista de los equipos que ha creado. (<i>Figure 12</i>) 2.- Si en este momento no tiene ninguno creado saldrá un mensaje de error. (<i>Figure 21</i>) 3.- Por el contrario, si existe alguno, se mostrará la lista de equipos con el botón de Crear Equipo y el de Volver. (<i>Figure 22</i>) 4.- [Si el Usuario pulsa sobre uno de los equipos:] <ol style="list-style-type: none"> 4a.- Le aparecerán los datos del equipo junto con un botón de Modificar Equipo. (<i>EXTEND Modificar equipo</i>) (<i>Figure 23</i>) 5.- [Si el Usuario pulsa sobre ”Crear Equipo”:] 5a.- Lleva al usuario a la pantalla de Crear Equipo (<i>EXTEND Crear equipo</i>) (<i>Figure 19</i>). 6.- [Si el Usuario pulsa sobre ”Volver”:] 6a.- Devuelve al Usuario al Menú Principal (<i>Figure 12</i>).
Postcondiciones:	<ul style="list-style-type: none"> • Si el usuario ha modificado alguno de los equipos se harán los cambios en la base de datos. • Si ha creado un equipo se añadirá a la base de datos. • Si no ha hecho ninguna de las anteriores no cambia nada.

Interfaz Gráfica:



Figure 21: Error: No hay equipos creados



Figure 22: Lista equipos

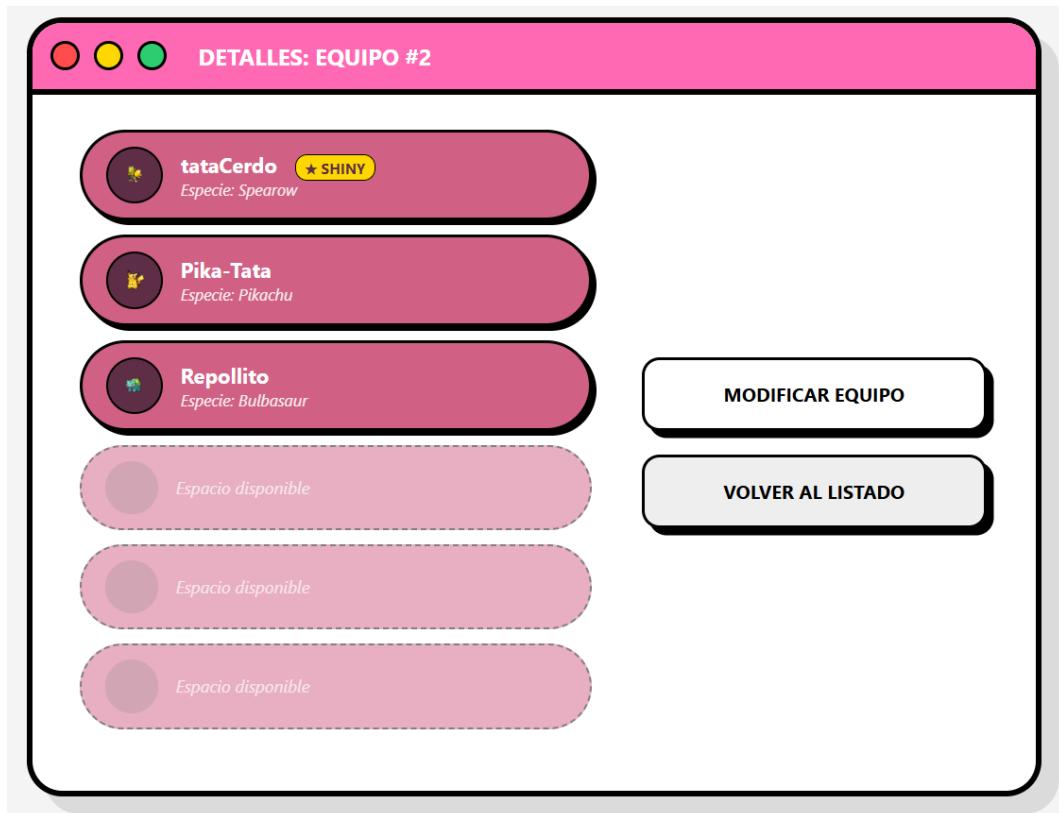


Figure 23: Detalles equipo

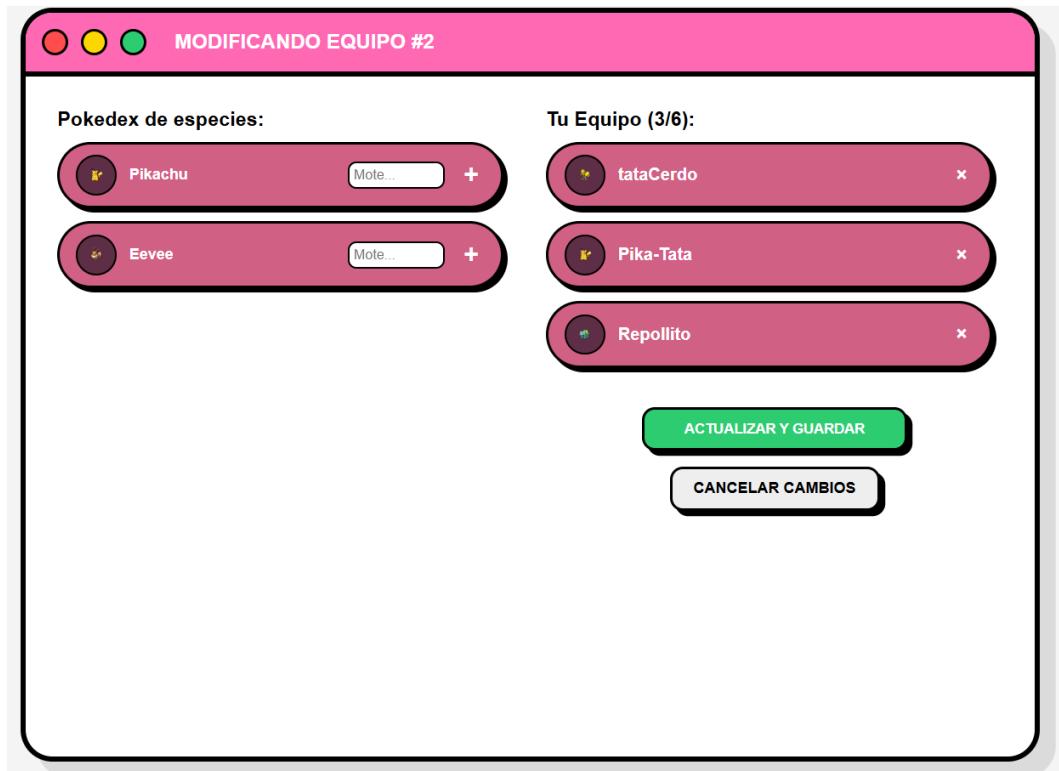


Figure 24: Modificar equipo

3.3 ChangeLog (Gorka)

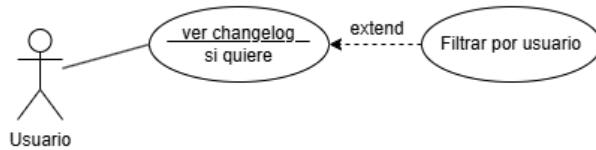


Figure 25: Diagrama Changelog

Nombre:	Changelog
Descripción:	Permite al Usuario ver los eventos más recientes como las modificaciones a los equipos, capturas de pantalla y otros comentarios que suban los usuarios a los que sigue. También, muestra eventos generados por el equipo de desarrollo, notificaciones de actualizaciones por ejemplo. Además se podrá filtrar un usuario específico al que siga para ver sus publicaciones.
Actores:	Usuario
Precondiciones:	Ninguna
Requisitos no funcionales:	El sistema mostrará la información en orden cronológico, de lo más reciente a lo mas antiguo.
Flujo de Eventos:	<p>1.- El Usuario selecciona en la interfaz la opción de ver el Changelog. (<i>Figure 27</i>)</p> <p>2.- Si en este momento no sigue a ningún usuario solo saldrán las notificaciones del equipo. (<i>Figure 29</i>)</p> <p>3.- Si se sigue a algun usuario, también muestra el changelog con los eventos (si hay) recientes de todos los usuarios que sigue en el sistema. (<i>Figure 29</i>)</p> <p>4.- [Si el usuario desea filtrar por un usuario en particular y tiene amigos] (<i>EXTEND Filtrar por usuario</i>)</p> <p>4a.- El sistema solicita seleccionar el usuario del que se quieren ver los eventos.</p> <p>4b.- El sistema muestra únicamente los eventos generados por ese usuario.</p> <p>5.- [Si el usuario desea filtrar por un usuario pero no tiene amigos] (<i>EXTEND Filtrar por usuario</i>)</p> <p>5a.- El sistema mostrará un mensaje de error. (<i>Figure 28</i>)</p> <p>6.- [Usuario pulsa en “Volver”]</p> <p>6a.- El sistema devuelve al usuario a la interfaz principal.</p>
Postcondiciones:	Ninguna.

Interfaz Gráfica:

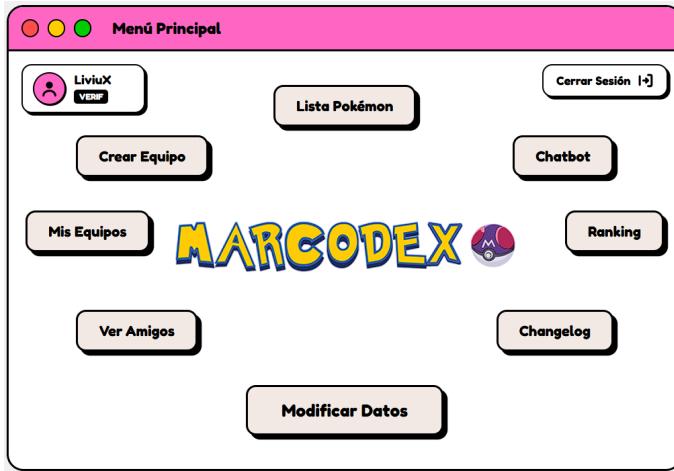


Figure 26: Menú principal



Figure 27: Changelog si no sigues a ningnº usuario



Figure 28: Changelog

3.4 Lista Pokemon y filtros (Laura)

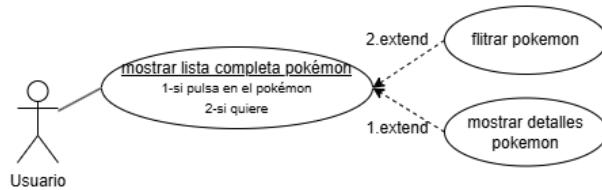


Figure 29: Diagrama Mostrar lista completa Pokémon

Nombre:	Mostrar lista completa Pokémon
Descripción:	Permite al actor Usuario (persona que ya ha ingresado a la aplicación, con la sesión iniciada) revisar la lista de todos los Pokémon (obtenida de la API). Una vez se muestra la lista, hay 2 opciones. Por un lado, se puede filtrar Pokémon (se extiende al caso de uso secundario Filtrar Pokémon). Por otro lado, se puede elegir un Pokémon de la lista y mostrar toda la información de ese Pokémon (se extiende al caso de uso secundario Mostrar detalles Pokémon).
Actores:	Usuario
Precondiciones:	Ninguna.
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<p>1.- El actor Usuario selecciona en su menú principal “Lista Pokémon”. (<i>Figure 31</i>)</p> <p>2.- El sistema muestra una lista de todos los Pokémon. (<i>Figure 32</i>)</p> <p>3.- El Usuario, si quiere, puede filtrar por nombre o tipo y le aparecerán los Pokémon existentes pulsando “Filtrar”(EXTEND Filtar pokemon). En caso de que no exista el nombre del Pokémon, no aparecerá ninguno.</p> <p>4.- El Usuario puede elegir un Pokémon de la lista y ver toda su información (EXTEND Mostrar detalles pokemon): (<i>Figure 33</i>)</p> <ul style="list-style-type: none"> • Nombre del Pokémon • Descripción • Altura • Peso • Legendario • Movimientos • Imagen del Pokémon <p>5.- [Si el usuario pulsa en “Volver”] Se devolverá al Usuario al Menú Principal.</p>
Postcondiciones:	Ninguna.

Interfaz Gráfica:

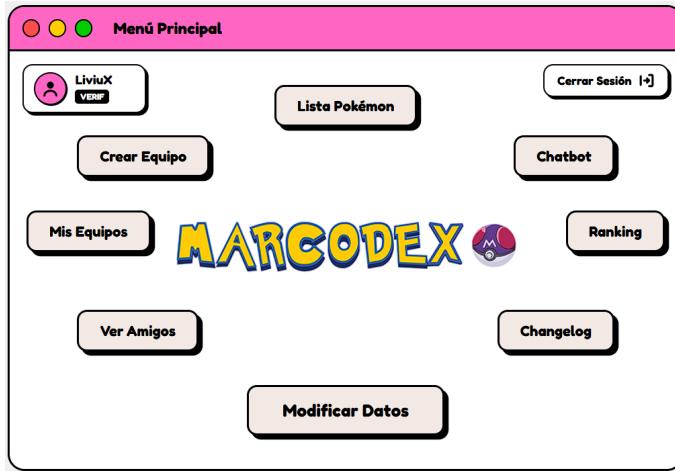


Figure 30: Menú principal

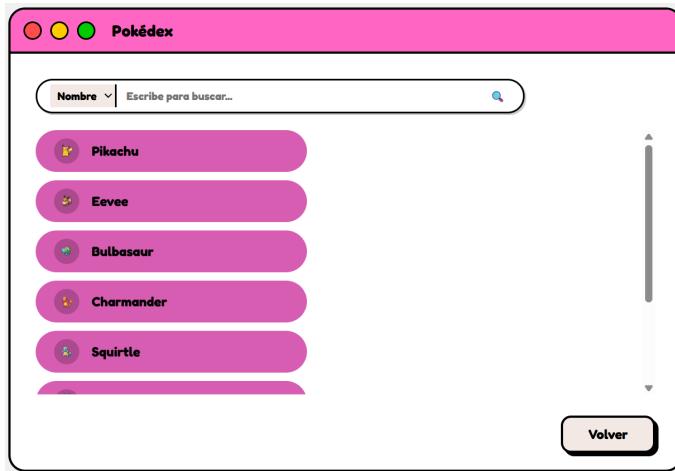


Figure 31: Lista completa Pokémon



Figure 32: Detalles Pokémon

3.5 Chatbot (Iker)



Figure 33: Diagrama Chatbot

Nombre:	Hablar con Chatbot
Descripción:	Permite a los usuarios interactuar con el chatbot para que este le devuelva información sobre diferentes características, como los tipos contra los que un Pokémon es fuerte, sus habilidades y características , su cadena evolutiva o determinar cuál es el mejor Pokémon de un equipo.
Actores:	Usuario
Precondiciones:	Ninguna
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<ol style="list-style-type: none"> 1.- El Usuario selecciona en su interfaz gráfica la opción de hablar con el chatbot. (<i>Figure 35</i>) 2.- Al Usuario se le mostrarán 5 opciones para elegir. (<i>Figure 36</i>) 3.- [Si el usuario ha elegido ver el mejor Pokémon de un equipo] (<i>Figure 37</i>) <ul style="list-style-type: none"> 3a.- Se le pedirá elegir un equipo para que el chatbot le diga cuál es su mejor Pokémon. 4.- [Si el usuario ha elegido ver contra qué tipos un Pokémon es fuerte] <ul style="list-style-type: none"> 4a.- Se le pedirá introducir un Pokémon para que le diga contra qué tipos es fuerte. 5.- [Si el usuario ha elegido ver la cadena evolutiva de un Pokémon] <ul style="list-style-type: none"> 5a.- Se le pedirá introducir un Pokémon para que le diga su cadena evolutiva. 6.- [Si el usuario ha elegido ver las habilidades y características de un Pokémon] <ul style="list-style-type: none"> 6a.- Se le pedirá introducir un Pokémon para que le diga sus cualidades. 7.- [Si el usuario ha elegido “Volver”] <ul style="list-style-type: none"> 7a.- Se le devolverá al usuario a la interfaz principal.
Postcondiciones:	Ninguna.

Interfaz Gráfica:

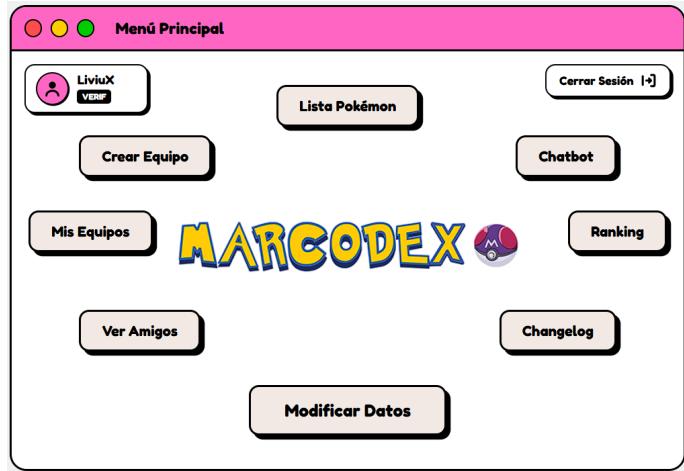


Figure 34: Menú principal

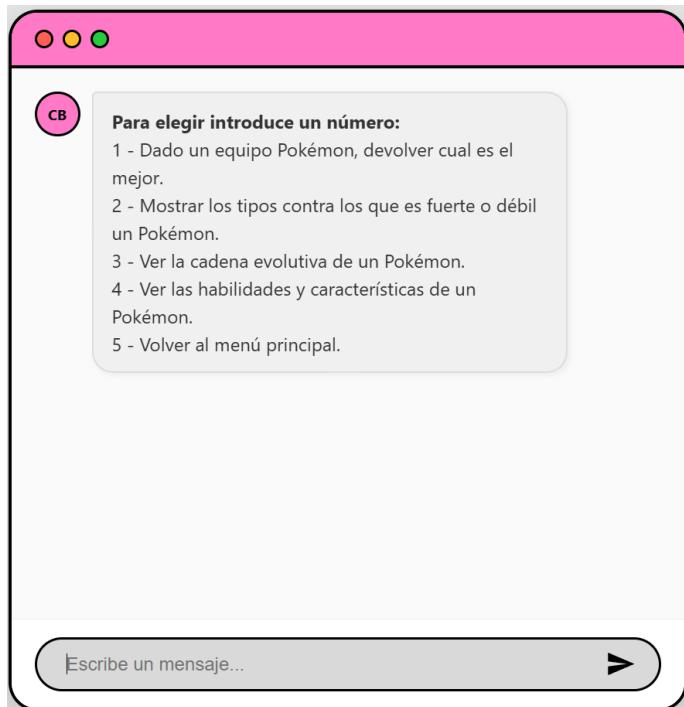


Figure 35: Chatbot

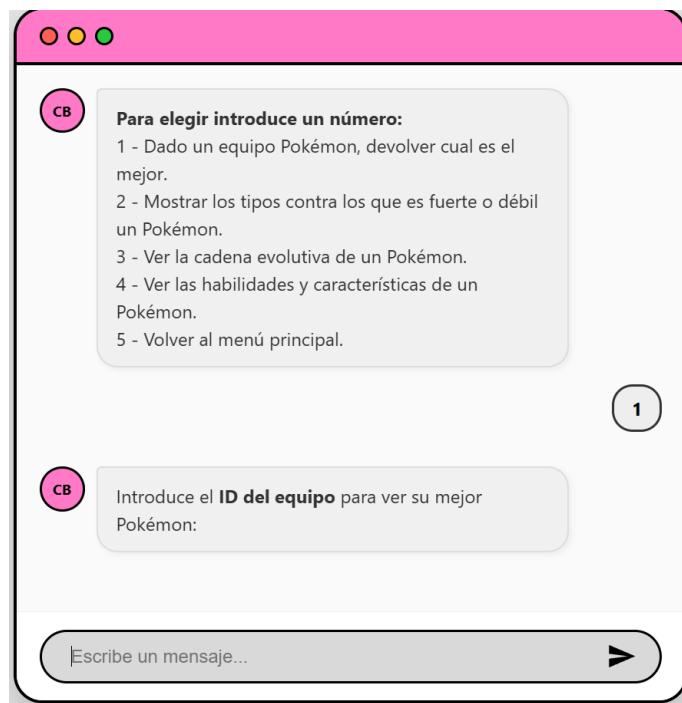


Figure 36: Chatbot: Caso de elegir la primera opción

3.6 Ranking (Marco)

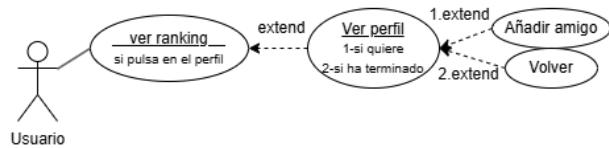


Figure 37: Diagrama Ver Ranking

Nombre:	Ver ranking
Descripción:	Permite al Usuario consultar el ranking de otras cuentas, ordenado según qué cuenta tiene una colección de pokémon con más rareza, dentro del equipo principal de cada uno.
Actores:	Usuario
Precondiciones:	Ninguna.
Requisitos no funcionales:	Ninguno.
Flujo de Eventos:	<p>1.- El Usuario selecciona en su Menú Principal “Ranking” para ver el ranking de usuarios con pokémon más raros. (<i>Figure 39</i>)</p> <p>2.- Se muestra la lista en orden, con la foto de perfil, el nombre de usuario y la posición que ocupa en el ranking mundial. (<i>Figure 40</i>)</p> <p>3.- [Si el usuario presiona uno de los perfiles:] (<i>EXTEND Ver perfil</i>) (<i>Figure 41</i>)</p> <p>3a.- Se accede al perfil completo del usuario seleccionado, incluyendo los pokémon que se ven dentro del equipo principal, foto de perfil, nombre de usuario y botón de añadir amigo.</p> <p>i.- [Si el usuario presiona al botón de solicitud de amistad:] (<i>EXTEND Añadir amigo</i>) Se añadirá a la lista de amigos del usuario al destinatario.</p> <p>ii.- [Si el usuario presiona “Volver”:] (<i>EXTEND Volver</i>) Se vuelve a la lista del ranking.</p>
Postcondiciones:	<ul style="list-style-type: none"> • Si el Usuario ha presionado el botón de añadir amigo se añadirá a su lista de amigos en la base de datos el usuario al que ha hecho la solicitud de amistad. • Si no, no se realiza ningún cambio.

Interfaz Gráfica:

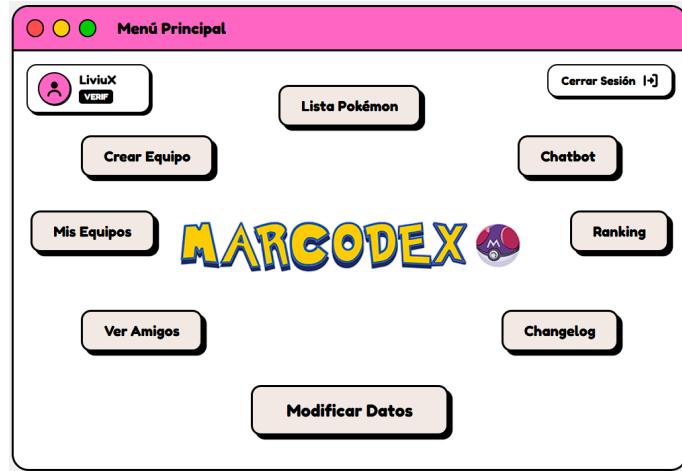


Figure 38: Menú principal

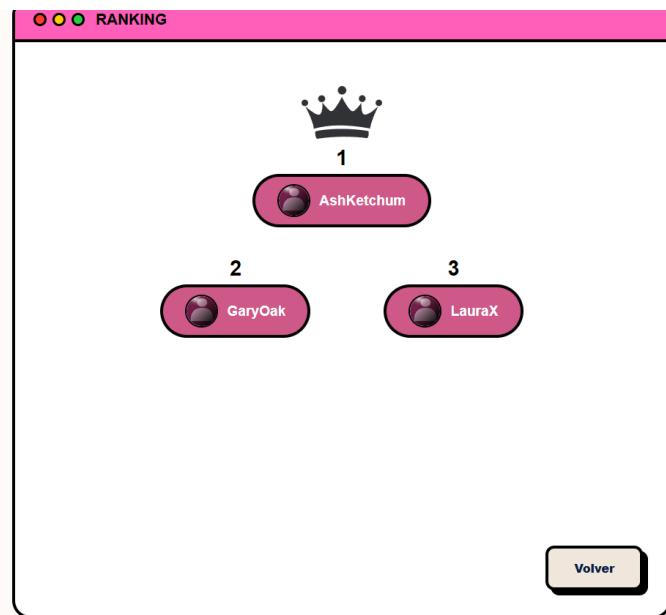


Figure 39: Ranking



Figure 40: Perfil Usuario

4 Modelo de Dominio

4.1 Introducción

En nuestro modelo de dominio se especifican cuáles serán las relaciones entre los distintos tipos de entidades que formarán parte del sistema web que hemos desarrollado. Estas entidades contienen atributos y relaciones de acuerdo con el enunciado propuesto para desarrollar la aplicación de Pokémon. Tras consultar las diferentes funcionalidades, se ha concluido que la estructura del modelo de dominio de la aplicación será la siguiente:

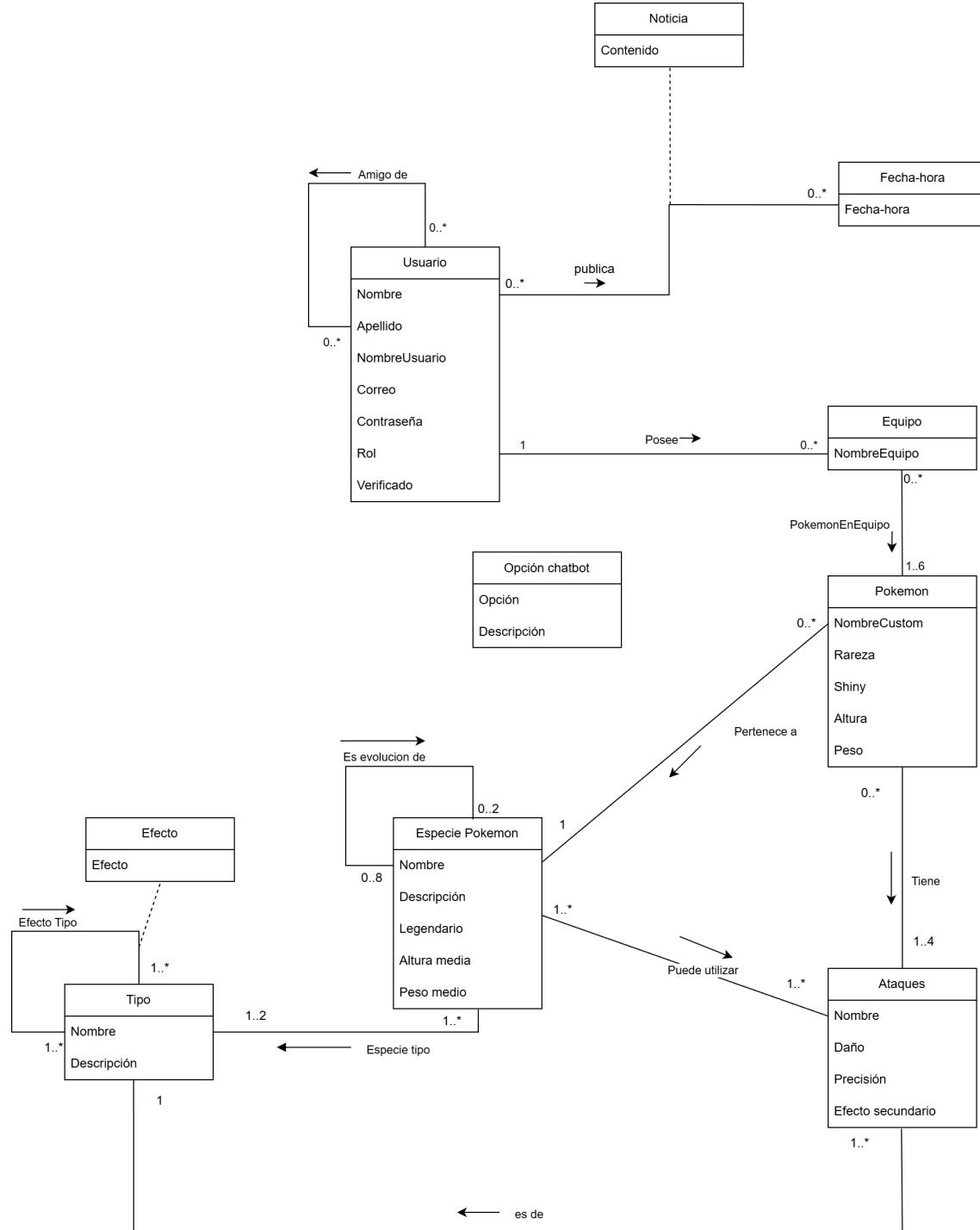


Figure 41: Diagrama del modelo de dominio

4.2 Usuario

La entidad Usuario representa a la persona que usa la aplicación. Esta entidad tiene un nombre, apellido, nombre de usuario, un correo electrónico, la contraseña y un atributo **Rol** que indica el rol que cumple el usuario en el sistema. Este rol puede variar entre si el usuario es administrador o si el usuario está aceptado en el sistema por el administrador o no.

Una de las funcionalidades que será capaz de realizar el usuario, es la de **hacerse amigo** de otros usuarios. Esta relación de amistad no necesita de confirmación mutua, es decir, basta con que un usuario añada a otro para que este aparezca en su lista de amigos.

4.3 Equipo

Otra de las funcionalidades que será capaz de hacer el usuario es la de **crear equipos** de Pokémon, el cual será único por cada usuario. Cada equipo tendrá un nombre. A su vez, cada equipo creado por el usuario debe contener como mínimo 1 Pokémon, con un total de hasta 6 Pokémon. En caso de que al crear el equipo no se añada ningún Pokémon, el equipo no podrá ser creado.

4.4 Changelog

Para mantener al usuario informado sobre cualquier cambio realizado por él mismo o por sus amigos, podrá **consultar el changelog**: una lista de mensajes o notificaciones con un contenido predeterminado que se mostrarán según los cambios que ocurran, donde se mostrará la hora en la que ocurrió ese cambio. El usuario no puede recibir más de una notificación en el mismo momento, por lo que será de importancia guardar la fecha y hora en la que se emitió la notificación. En el sistema también se almacena el contenido de la noticia en cuestión, como atributo de la relación entre el usuario y la fecha.

4.5 Pokédex

En la aplicación disponemos de diferentes Pokédex que, al igual que en los juegos, se encargan de contener toda la información relacionada a las diferentes especies de los Pokémon. En el modelo planteado, se almacena una Pokédex por cada región, conteniendo las especies del lugar. Por cada Pokédex se almacena la región a la que representa y la generación en la que fue introducida.

4.6 Especie Pokémon

Como ya se ha indicado, existen diferentes Pokedex que contienen información sobre las especies de los Pokémon. Por cada especie se almacena su nombre, descripción, un atributo que indica si es legendario o no, su altura y su peso medios. Las especies pertenecen a una única Pokedex, en concreto, a la región de la que son originarios.

Cabe mencionar que los Pokémon que se almacenan en las Pokedex no forman parte de la misma entidad de Pokémon que poseen los usuarios. Por cada especie, pueden existir muchos Pokémon de diferentes usuarios, mientras que cada Pokémon pertenece a una especie. Es decir, por ejemplo, puede haber varios Pikachu almacenados en equipos de diferentes usuarios, pero todos ellos pertenecen a la especie Pikachu. También pueden haber especies que no tienen Pokémon registrados en ningún equipo.

Las especies disponen de un determinado número de ataques que podrán usar, todos los ataques posibles que ese Pokémon puede poseer. Además, hay una relación con tipo en la que una especie de Pokémon puede ser de hasta dos tipos.

4.7 Pokémon

A parte de la especie, tenemos una entidad pokémon la cual pertenece a los equipos. Cada pokémon tiene un nombre propio el cual elige el usuario, la rareza que tiene cada pokémon, si es shiny o no y su altura y su peso. Cada pokémon tiene de uno a cuatro ataques y pertenece a una sola especie pokémon.

4.8 Ataques

Para cada ataque se almacena el daño que infinge, la probabilidad de acierto, el nombre y los efectos secundarios que tiene.

4.9 Tipo

Como es bien sabido, todos los Pokémon y ataques que estos usan están categorizados por diferentes tipos. Esto es, siempre van a ser mínimo de un tipo. Por un lado, una especie de Pokémon puede ser de hasta dos tipos. Por otro lado, los ataques solo pueden tener un tipo. Cabe recalcar que no hay ningún Pokémon ni ningún ataque que no tenga tipo.

De los tipos se guarda su nombre y su descripción, a la vez que los tipos contra los que tenga superioridad, los que tenga inferioridad como los que sean inmunes. Ya es sabido que un tipo puede ser más eficaz contra unos y más débil contra otros. Esa relación entre los tipos también se va a guardar teniendo en cuenta que no todos los tipos van a estar relacionados entre sí.

4.10 Chatbot

Por último, tenemos la entidad del chatbot, la cual se encarga de almacenar las descripciones de las opciones que tiene el chatbot (del 1 al 5). Al disponer de 5 opciones, consideramos que los mensajes que envía el chatbot son limitados e iguales entre sí, y consecuentemente, predeterminados.

5 Plan de Pruebas

5.1 Gestión Usuarios (Gorka)

Cód.	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.1	CU: Registrarse El registro de usuario se realiza sin dejar ningún campo vacío, introduciendo un correo y un nombre de usuario que no están repetidos.	Se crea un nuevo usuario sin problemas, se almacena en la BD y te manda al inicio de sesión.	Se crea un nuevo usuario sin problemas, se almacena en la BD y te manda al inicio de sesión.	
1.2	CU: Registrarse El registro de usuario se realiza sin dejar ningún campo vacío, introduciendo un correo ya registrado y un nombre de usuario que no está repetido.	Salta un aviso de que el correo ya está registrado, no se crea un nuevo usuario y vuelve al registro.	Salta un aviso de que el correo ya está registrado, no se crea un nuevo usuario y vuelve al registro.	
1.3	CU: Registrarse El registro de usuario se realiza sin dejar ningún campo vacío, introduciendo un correo que no está repetido y un nombre de usuario ya registrado.	Salta un aviso de que el nombre de usuario ya está registrado, no se crea un nuevo usuario y vuelve al registro.	Salta un aviso de que el nombre de usuario ya está registrado, no se crea un nuevo usuario y vuelve al registro.	
1.4	CU: Registrarse El registro de usuario se realiza dejando algún campo vacío, introduciendo un correo y un nombre de usuario que no están repetidos.	Salta un aviso de que alguno de los campos está vacío.	Salta un aviso de que alguno de los campos está vacío.	
1.5	CU: Registrarse El registro de usuario se realiza dejando algún campo vacío, introduciendo un correo que no está repetido y un nombre de usuario ya registrado.	Salta un aviso de que alguno de los campos está vacío.	Salta un aviso de que alguno de los campos está vacío.	
1.6	CU: Registrarse El registro de usuario se realiza dejando algún campo vacío, introduciendo un correo que ya está registrado y un nombre que no está repetido.	Salta un aviso de que alguno de los campos está vacío.	Salta un aviso de que alguno de los campos está vacío.	
1.7	CU: Registrarse El registro de usuario se realiza dejando algún campo vacío, introduciendo un correo que ya está registrado y un nombre de usuario ya registrado.	Salta un aviso de que alguno de los campos está vacío.	Salta un aviso de que alguno de los campos está vacío.	

Cód.	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.8	CU: Registrarse El registro de usuario se realiza introduciendo una contraseña distinta en el "campo contraseña" y en "repetir contraseña"	Salta un aviso de que las contraseñas son diferentes.	Salta un aviso de que las contraseñas son diferentes.	
1.9	CU: Registrarse El registro de usuario se realiza introduciendo en el nombre caracteres especiales no permitidos.	Salta un aviso de que el nombre de usuario no es válido.	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación de correo válido.
2.1	CU: Iniciar sesión El inicio de sesión se realiza introduciendo los datos correctamente.	Se le redirige a su página principal de usuario.	Se le redirige a su página principal de usuario.	
2.2	CU: Iniciar sesión El inicio de sesión se realiza con el campo del usuario vacío.	El sistema da un aviso de que el campo del usuario está vacío.	El sistema da un aviso de que el campo del usuario está vacío.	
2.3	CU: Iniciar sesión El inicio de sesión se realiza con el campo de la contraseña vacía.	El sistema da un aviso de que el campo de la contraseña está vacío.	El sistema da un aviso de que el campo de la contraseña está vacío.	
2.4	CU: Iniciar sesión El inicio de sesión se realiza con el usuario incorrecto, pero contraseña correcta.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	
2.5	CU: Iniciar sesión El inicio de sesión se realiza con el usuario correcto, pero contraseña incorrecta.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	
2.6	CU: Iniciar sesión El inicio de sesión se realiza con el usuario incorrecto y con la contraseña incorrecta.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	
2.7	CU: Iniciar sesión El inicio de sesión se realiza añadiendo espacios al principio o al final del nombre de usuario.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	
2.8	CU: Iniciar sesión El inicio de sesión se realiza añadiendo espacios al principio o al final de la contraseña.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	El sistema da un aviso de que el usuario o la contraseña son incorrectos.	

Table 1: Casos de prueba de la funcionalidad Gestión Usuarios

Desarrollo de casos de prueba utilizando IA generativa (ChatGPT)

Una vez escritos todos los casos de prueba que se ha sido capaz de pensar, se ha decidido consultar a la IA por posibles correcciones y por más casos de prueba con el siguiente mensaje:

"Buenas, he tenido que hacer los casos de prueba de mi aplicación de las funcionalidades de Iniciar Sesión y Registrarse. ¿Puedes corregirmelas si encuentras algún fallo y añadir alguna más que se te ocurra?"
(Con los casos desde el 1.1 al 1.8 y desde el 2.1 al 2.6)

y se ha obtenido la siguiente respuesta:

"Perfecto, Tu tabla de casos de prueba está muy bien planteada: es clara, estructurada y cubre la mayoría de los escenarios normales y de error. Aun así, puedo proponerte algunas correcciones menores y mejoras o casos adicionales para hacerla más completa y rigurosa desde el punto de vista de pruebas funcionales.

Correcciones y mejoras en los casos actuales Tabla de "Registrarse":

- Los casos 1.5, 1.6 y 1.7 son redundantes: si hay campos vacíos, el sistema debería detectar primero los campos vacíos antes de validar duplicados. Por tanto, el resultado esperado debería centrarse solo en el aviso de "campos vacíos", sin importar si el correo o usuario están repetidos. Puedes unificarlos o mantenerlos para probar la prioridad de validaciones, pero aclara que no se llega a validar duplicados porque el sistema detiene el proceso antes.
- En el 1.4–1.7, podrías precisar qué campo está vacío (correo, nombre o contraseña) para probarlos individualmente. Eso da más cobertura.
- El mensaje de error podría normalizarse: "El sistema muestra un aviso indicando que hay campos obligatorios sin completar" (más formal que "salta un aviso").

Casos adicionales recomendados (Registro):

Cód.	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.9	CU: Registrarse El usuario introduce un correo con formato inválido (por ejemplo, sin "@" o dominio).	El sistema muestra un aviso indicando que el formato del correo electrónico no es válido.	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación de correo válido.
1.10	CU: Registrarse El usuario introduce una contraseña demasiado corta (según las políticas, p. ej. menos de 8 caracteres).	El sistema muestra un aviso indicando que la contraseña no cumple los requisitos mínimos.	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación de longitud de contraseña mínima.
1.11	CU: Registrarse El usuario intenta registrarse con caracteres especiales no permitidos en el nombre de usuario.	El sistema muestra un aviso indicando que el nombre de usuario contiene caracteres inválidos.	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación de nombre de usuario válido.	
1.12	CU: Registrarse El usuario se registra correctamente, pero luego intenta registrarse de nuevo con los mismos datos.	El sistema muestra un aviso de que el usuario ya existe.	El sistema muestra un aviso de que el usuario ya existe.	
1.13	CU: Registrarse El sistema pierde la conexión con la base de datos al intentar registrar.	Se muestra un mensaje de error indicando que no se ha podido completar el registro por un fallo del sistema.	Se muestra un mensaje de error indicando que no se ha podido completar el registro por un fallo del sistema.	

Casos adicionales recomendados (Iniciar sesión):

Cód.	Descripción	Resultado esperado	Resultado obtenido	Observaciones
2.7	CU: Iniciar sesión El usuario introduce caracteres especiales o un formato inválido en el campo usuario.	El sistema muestra un aviso indicando formato inválido.	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación de nombre de usuario válido.
2.8	CU: Iniciar sesión El usuario intenta iniciar sesión con un correo válido pero aún no verificado (si hay verificación por correo).	El sistema muestra un aviso indicando que la cuenta no ha sido verificada.	El sistema muestra un aviso indicando que la cuenta no ha sido verificada.	
2.9	CU: Iniciar sesión El usuario introduce espacios al inicio o final del usuario o contraseña.	El sistema ignora los espacios y valida correctamente (si está permitido) o muestra error (si no).	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación la verificación por correo.
2.10	CU: Iniciar sesión Se intenta iniciar sesión mientras el servidor o la base de datos están caídos.	El sistema muestra un aviso de error técnico o de conexión.	El sistema muestra un aviso de error técnico o de conexión.	
2.11	CU: Iniciar sesión El usuario ha sido bloqueado por intentos fallidos.	El sistema muestra un aviso indicando que la cuenta está bloqueada.	-	No se ha implementado esta prueba, puesto que no figura en nuestra aplicación el límite de intentos de inicio de sesión.

"

Esta información ha sido analizada, tratada y se ha utilizada. Las correcciones han sido prácticamente ignoradas porque no se cree que ningún caso añadido sea redundante y porque no tiene sentido añadir todas las combinaciones posibles de que campos pueden estar vacíos.

Por otro lado, sí que se ha utilizado para añadir tres casos de prueba adicionales: el 1.9 que se basa en el 1.11 que ha dado la IA y los 2.7 y 2.8 que se basan en el caso 2.9 que ha devuelto la IA.

5.2 Crear Equipos Pokémon (Iker)

Código de Prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.1	CU: Crear equipo. El usuario pulsa en un Pokémon de la lista.	El Pokémon se añade en uno de los 6 huecos que hay en el equipo.	El Pokémon se añade en uno de los 6 huecos que hay en el equipo.	
1.2	CU: Crear equipo. El usuario selecciona un Pokémon añadido ya previamente.	El Pokémon nuevo no se añade y sale un mensaje de error.	El Pokémon nuevo no se añade y sale un mensaje de error.	
1.3	CU: Crear equipo. Se pulsa sobre la <i>x</i> para borrar un Pokémon.	El Pokémon se elimina del equipo.	El Pokémon se elimina del equipo.	
1.4	CU: Crear equipo. Se selecciona otro Pokémon de la lista teniendo ya 6.	Salta un mensaje de error indicando que cada equipo puede tener un máximo de 6 Pokémon.	Salta un mensaje de error indicando que cada equipo puede tener un máximo de 6 Pokémon.	
2.1	CU Secundario: Cancelar. El usuario pulsa cancelar.	Se le devuelve a la página principal sin guardar nada en la base de datos.	El usuario pulsa cancelar.	Se le devuelve a la página principal sin guardar nada en la base de datos.
2.2	CU Secundario: Cancelar. El usuario pulsa cancelar tras haber añadido algún Pokémon al equipo.	Se le devuelve a la página principal sin haber guardado ningún equipo. El usuario tendrá que construir el equipo de cero en un futuro.	Se le devuelve a la página principal sin haber guardado ningún equipo. El usuario tendrá que construir el equipo de cero en un futuro.	
3.1	CU Secundario: Guardar. Se pulsa guardar teniendo menos de 6 Pokémon seleccionados.	El equipo se guarda en la base de datos con los Pokémon seleccionados.	El equipo se guarda en la base de datos con los Pokémon seleccionados.	Este bloque de pruebas se ha unificado en 1
3.2	CU Secundario: Guardar. El usuario pulsa guardar teniendo 6 Pokémon en el equipo.	El equipo se guarda en la base de datos con los 6 Pokémon seleccionados.	El equipo se guarda en la base de datos con los 6 Pokémon seleccionados.	Este bloque de pruebas se ha unificado en 1

Código de Prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
4.1	CU: Ver equipos. El usuario entra a la página sin tener equipos creados.	Se muestra un mensaje indicando que el usuario no tiene equipos.	Se muestra una pantalla indicando que el usuario no tiene equipos	
4.2	CU: Ver equipos. El usuario entra a la página habiendo creado algún equipo anteriormente.	El usuario entra a la interfaz, mostrando por pantalla una lista de sus equipos, mostrando un Pokémon por cada equipo, un botón para volver al menú principal y otro botón que permite crear un equipo nuevo.	El usuario entra a la interfaz, mostrando por pantalla una lista de sus equipos, mostrando un Pokémon por cada equipo, un botón para volver al menú principal y otro botón que permite crear un equipo nuevo	
4.3	CU: Ver equipos. El usuario pulsa el botón de volver a la página principal.	Se le devuelve a la página principal	Se le devuelve a la página principal	
4.4	CU: Ver equipos. El usuario pulsa el botón de crear un nuevo equipo.	Se le lleva a la interfaz del caso de uso de crear equipo.	Se le lleva a la interfaz del caso de uso de crear equipo.	
4.5	CU: Ver equipos. El usuario pulsa sobre uno de sus equipos.	Se le lleva a una pantalla donde se le muestran los Pokémon que componen ese equipo. Además, esta pantalla dispone de un botón para volver y otro para modificar el equipo.	Se le lleva a una pantalla donde se le muestran los Pokémon que componen ese equipo. Además, esta pantalla dispone de un botón para volver y otro para modificar el equipo.	
4.6	CU: Ver equipos. Si, tras pulsar sobre un equipo, el usuario pulsa volver.	Se le devuelve a la interfaz anterior, la de ver equipo.	Se le devuelve a la interfaz anterior, la de ver equipo.	
4.7	CU: Ver equipos. Si, tras pulsar sobre un equipo, el usuario pulsa modificar equipo.	Se le llevará a la interfaz de modificar equipo. La interfaz es parecida a la de crear equipo, solo que la lista de Pokémon aparecerá al pulsar sobre uno de los integrantes del equipo para sustituirlos.	Se le llevará a la interfaz de modificar equipo, a la izquierda está la lista de especies y en el lado derecho aparecerán los Pokémon del equipo a modificar	Durante el proceso se ha cambiado la idea principal de cómo implementar la modificación

Código de Prueba	Descripción	Resultado esperado	Resultado obtenido	Observaciones
5.1	CU Secundario: Modificar equipo. El usuario pulsa cancelar.	Se le devuelve a la página que muestra los datos del equipo.	Se le devuelve a la página que muestra los datos del equipo.	
5.2	CU Secundario: Modificar equipo. El usuario pulsa en un Pokémon de la lista.	Se dan las misma condiciones que el añadir del Crear Equipo .	Se dan las misma condiciones que el añadir del Crear Equipo .	
5.3	CU Secundario: Modificar equipo. El usuario pulsa en un Pokémon de su equipo.	El Pokémon se borra del equipo.	El Pokémon se borra del equipo.	
5.4	CU Secundario: Modificar equipo. El usuario pulsa cancelar tras haber modificado un Pokémon en el equipo.	Se le devuelve a la página anterior sin haber modificado la base de datos.	Se le devuelve a la página anterior sin haber modificado la base de datos.	
5.5	CU Secundario: Modificar equipo. El usuario sustituye un Pokémon por otro de la misma especie.	El Pokémon nuevo es un Pokémon distinto de la misma especie.	El Pokémon nuevo es un Pokémon distinto de la misma especie.	
5.6	CU Secundario: Modificar equipo. Se pulsa guardar sin haber hecho cambios.	Se guardan los cambios en la base de datos pese a no haber cambiado el equipo.	Se guardan los cambios en la base de datos pese a no haber cambiado el equipo.	Realmente no se hace ninguna acción sobre la BD porque no se detecta cambio
5.7	CU Secundario: Modificar equipo. El usuario pulsa guardar habiendo modificado un Pokémon.	El equipo modificado se guarda en la base de datos.	El equipo modificado se guarda en la base de datos.	En esta prueba y la anterior se hace un sleep de 1.1 para que no haya problemas de integridad de clave primaria en la tabla Publica del ChangeLog, ya que sino habría más de una noticia en la misma fecha y hora

5.3 Changelog (Marco)

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
1.1	CU: Changelog El usuario no tiene amigos.	Solo se muestran eventos del equipo de desarrollo.	El changelog aparece vacío.	Para ver mensajes de desarrollo, es necesario tener agregado al administrador.
1.2	CU: Changelog El usuario tiene un amigo.	Muestra evento del amigo y recientes mezclados.	Se visualiza la noticia del amigo y del sistema.	
1.3	CU: Changelog El usuario tiene varios amigos.	Muestra eventos de amistades y recientes mezclados.	Se visualizan las noticias de múltiples usuarios.	
1.4	CU: Changelog Se presiona volver (sin amigos).	Vuelve al menú principal.	El sistema redirige al menú principal.	
1.5	CU: Changelog Se presiona volver (un amigo).	Vuelve al menú principal.	El sistema redirige al menú principal.	
1.6	CU: Changelog Se presiona volver (varios amigos).	Vuelve al menú principal.	El sistema redirige al menú principal.	
1.7	CU: Changelog Error de conexión al cargar.	Muestra mensaje de error y permite volver.	Muestra el mensaje de error.	
1.8	CU: Changelog Eventos duplicados.	Elimina duplicados, muestra uno solo.	Se muestran eventos únicos sin repeticiones.	
1.9	CU: Changelog Orden cronológico incorrecto.	Reordena de más reciente a antiguo.	El contenido aparece reordenado correctamente.	
1.10	CU: Changelog Actualización manual.	Recarga información y muestra recientes.	La vista se actualiza correctamente.	
1.11	CU: Changelog Error conexión + Volver.	Mensaje de error, luego vuelve al menú.	Funcionalidad de retorno operativa tras el error.	

Table 2: Resultados de pruebas: Funcionalidad Changelog (Bloque 1)

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
2.1	CU: Filtrar El usuario no tiene amigos.	Muestra error y no permite filtrar.	Se muestra la pantalla del changelog vacía.	Conceptualmente imposible acceder al botón sin amigos.
2.2	CU: Filtrar Usuario con un amigo.	Muestra solo eventos de ese usuario.	Se filtra correctamente (aparece lo deseado, se oculta el resto).	
2.3	CU: Filtrar Usuario con varios amigos.	Muestra solo eventos del seleccionado.	Filtrado correcto por usuario seleccionado.	
2.4	CU: Filtrar Volver (sin amigos/error).	Vuelve al menú principal.	Redirección correcta al menú principal.	
2.5	CU: Filtrar Volver (viendo un amigo).	Vuelve al menú principal.	Redirección correcta al menú principal.	
2.6	CU: Filtrar Volver (viendo varios).	Vuelve al menú principal.	Redirección correcta al menú principal.	
2.7	CU: Filtrar Amigo sin cambios.	Mensaje de "sin eventos" o lista vacía.	Se muestra mensaje indicando falta de noticias.	
2.8	CU: Filtrar Volver sin seleccionar.	Vuelve al menú principal.	Redirección correcta al menú principal.	
2.9	CU: Filtrar Pérdida de conexión al filtrar.	Mensaje de error, permanece en vista.	Se muestra mensaje de error controlado.	
2.10	CU: Filtrar Error conexión + Volver.	Mensaje de error, permite volver al menú.	Funcionalidad de retorno operativa.	

Table 3: Resultados de pruebas: Funcionalidad Filtrado (Bloque 2)

5.4 Lista Pokemon y filtros (Liviu)

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
1.1	CU: Mostrar lista. Usuario pulsa "Lista Pokémon" en Menú Principal.	Se muestra en la pantalla "Pokédex" con la lista completa.	Se muestra en la pantalla "Pokédex" con la lista completa	
1.2	CU: Mostrar lista. Usuario pulsa "Lista Pokémon" en Menú Principal, pero falla la conexión o la petición a la API.	Se muestra en la pantalla "Ha ocurrido un error".	Se muestra que ha ocurrido un error".	
1.3	CU: Mostrar lista. Usuario pulsa "Volver" en la pantalla "Pokédex".	Devuelve al "Menú Principal".	Lleva a la pantalla del menú principal	
2.1	CU Secundario: Filtrar Pokémon. Clase Equivalente: nombre exacto. Ejemplo: introducir "Pikachu" en el filtro.	La lista muestra únicamente a "Pikachu".	La lista muestra únicamente a "Pikachu".	
2.2	CU Secundario: Filtrar Pokémon. Clase Equivalente: nombre parcial. Ejemplo: introducir "Pika" en el filtro.	La lista muestra "Pikachu" y cualquier otro Pokémon que empiece por "Pika". Pueden aparecer varios Pokémon si hay varios que empiecen por ese filtro.	La lista muestra a "Pikachu".	Solo muestra a Pikachu porque no hay ningún otro pokémon que empiece por "Pika"
2.3	CU Secundario: Filtrar Pokémon. Clase Equivalente: nombre inexistente. Ejemplo: introducir "Marco" en el filtro.	La lista se muestra vacía.	Aparece un mensaje de "No se encontraron Pokémon, prueba con otro filtro"	
2.4	CU Secundario: Filtrar Pokémon. Clase equivalente: nombre exacto en minúsculas. Ejemplo: introducir "pikachu" en el filtro.	La lista muestra a "Pikachu".	La lista muestra a "Pikachu"	
2.5	CU Secundario: Filtrar Pokémon. Clase equivalente: nombre exacto en mayúsculas. Ejemplo: introducir "PIKACHU" en el filtro.	La lista muestra a "Pikachu".	La lista muestra a "Pikachu"	

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
2.6	CU Secundario: Filtrar Pokémon. Clase equivalente: nombre exacto en minúsculas y mayúsculas. Ejemplo: introducir "PiKaChu" en el filtro.	La lista muestra a "Pikachu".	La lista muestra a "Pikachu".	
2.7	CU Secundario: Filtrar Pokémon. Clase equivalente: ningún carácter en la búsqueda. Dejar el filtro vacío y pulsar "Filtrar" (si existiera el botón) o borrar el texto.	La lista muestra todos los Pokémon.	La lista muestra todos los Pokémon.	
2.8	CU Secundario: Filtrar Pokémon. Clase equivalente: caracteres especiales. Ejemplo: introducir "Pika\$\$\$" o "1=1" en el filtro.	La lista se muestra vacía. El sistema no debe fallar.	Aparece un mensaje de "No se encontraron Pokémon, prueba con otro filtro"	
2.9	CU Secundario: Filtrar Pokémon. Clase Equivalente: cadena muy larga. Ejemplo: Introducir una cadena de 500 caracteres.	La lista se muestra vacía. El sistema no debe fallar ni colapsar.	Aparece un mensaje de "No se encontraron Pokémon, prueba con otro filtro"	
2.10	CU Secundario: Filtrar Pokémon. Clase Equivalente: nombre exacto con espacios. Ejemplo: introducir " Pikachu " (con espacios).	La lista muestra a "Pikachu". (El sistema debe limpiar la entrada).	La lista muestra a "Pikachu"	
3.1	CU Secundario: Mostrar detalles. Se muestran los detalles de un Pokémon en concreto. Ejemplo: pulsar en un Pokémon (ej. "Bulbasaur") de la lista.	Muestra la pantalla "Pokémon" con los detalles de "Bulbasaur".	Aparece los detalles de Bulbasaur (nombre, descripción, tipos, altura, peso, si es legendario y los movimientos)	Si no tiene movimientos no aparece ninguno (lista vacía)
3.2	CU Secundario: Mostrar detalles. Se muestran los detalles de un Pokémon en concreto, pero falla la conexión o la petición a la API. Ejemplo: pulsar en un Pokémon (ej. "Bulbasaur") de la lista.	Se muestra en la pantalla "Ha ocurrido un error".	Se muestra en la pantalla "Ha ocurrido un error"	

Desarrollo de chat con IA (Gemini Pro) para la creación de pruebas

En primer lugar, se facilitó al modelo el archivo con los **casos de uso del proyecto**, junto con una explicación detallada sobre la metodología que debía seguirse para el planteamiento de las pruebas (diapositivas 22–25 y 39–42 del tema 7). Se solicitó la elaboración de un **plan de pruebas de caja negra** y de la **documentación correspondiente**.

La **primera respuesta** obtenida fue demasiado general y contemplaba un número limitado de posibilidades, por lo que se le indicó al modelo que ampliara el conjunto de casos de prueba.

En su **segunda entrega**, se pidió la eliminación de información adicional que hacía referencia a las fuentes utilizadas para generar las pruebas, con el fin de mantener el documento más conciso y centrado en los resultados.

Finalmente, para **comprender mejor el concepto de clase equivalente**, se formuló una consulta adicional al modelo. Sin embargo, dicha pregunta no tuvo impacto en la solución final planteada para las pruebas del apartado 3.4.

5.5 Chatbot (Tabata)

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
1.1	CU:Chatbot Usuario pulsa la opción de “Chatbot” de Menu principal	Muestra la pantalla del ChatBot con las 5 opciones.	Las opciones del ChatBot se cargan y se muestran al usuario.	
1.2	CU:Chatbot Usuario envía “5” al chatbot	Devuelve al ”Menú Principal”.	Se le devuelve al usuario a la página del menú principal, borrando el chat que se había guardado en la interfaz del ChatBot.	
1.3	CU:Chatbot Usuario envía cualquier número que no esté entre 1 y 5.	Chatbot informa de que no existe dicha opción y pide que introduzcas un número del 1 al 5.	Se informa al usuario que solo se pueden introducir números del 1 al 5.	
2	CU:Chatbot Usuario envia “1” al chatbot, usuario tiene al menos un equipo y elige un equipo existente.	Chatbot pide que elija uno de sus equipos enumerados del 1 a n. Tras introducir el equipo Chatbot muestra el nombre del mejor Pokémon.	Se muestra el nombre y la información relacionada al Pokémon con más rareza del equipo en cuestión.	
2.1	CU:Chatbot Usuario envia “1” al chatbot, usuario tiene al menos un equipo y elige un equipo NO existente.	Chatbot pide que elija uno de sus equipos enumerados del 1 a n. Tras introducir el equipo Chatbot indica que el equipo seleccionado no existe.	Se indica que se debe introducir el ID de un equipo existente del usuario.	
2.2	CU:Chatbot Usuario envia “1” al chatbot, usuario no tiene equipos creados.	Chatbot indica que el usuario no tiene ningún equipo creado.	Se avisa de que el usuario no dispone de equipos.	
3	CU:Chatbot Usuario envia “2” al chatbot, usuario introduce el nombre de una especie Pokémon existente.	Chatbot muestra los tipos contra los cuales la especie seleccionada es fuerte.	ChatBot muestra los tipos contra los que la especie es fuerte y débil.	
3.1	CU:Chatbot Usuario envia “2” al chatbot, usuario introduce el nombre de una especie Pokémon NO existente.	Chatbot indica que la especie Pokémon seleccionada no existe.	Se avisa de que no se han encontrado datos con el parámetro introducido.	

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
4	CU:Chatbot Usuario envia “3” al chatbot, usuario introduce el nombre de una especie Pokémon existente.	Chatbot muestra la cadena evolutiva del Pokémon seleccionado.	Se muestran los nombres de los Pokémon que componen la cadena evolutiva de la especie.	
4.1	CU:Chatbot Usuario envia “3” al chatbot, usuario introduce el nombre de una especie Pokémon NO existente.	Chatbot indica que la especie Pokémon seleccionada no existe.	Se avisa de que no se han encontrado datos con el parámetro introducido.	
5	CU:Chatbot Usuario envia “4” al chatbot, usuario introduce el nombre de una especie Pokémon existente.	Chatbot muestra las características y todas las habilidades del Pokémon seleccionado.	Se muestra la información, habilidades y tipos de la especie en cuestión.	
5.1	CU:Chatbot Usuario envia “4” al chatbot, usuario introduce el nombre de una especie Pokémon NO existente.	Chatbot indica que la especie Pokémon seleccionada no existe.	Se avisa de que no se han encontrado datos con el parámetro introducido.	

Table 5: Casos de prueba de la funcionalidad Chatbot

5.6 Ranking (Laura)

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
1.1	CU: Ranking Usuario pulsa la opción "Ranking" en el menú principal y hay mínimo un usuario que tiene mínimo un equipo.	Muestra el ranking de usuarios con los pokémon más raros.	Se muestra el ranking de usuarios con los pokémon más raros.	
1.2	CU:Ranking Usuario pulsa la opción "Ranking" en el menú principal y no hay ningún usuario que tenga equipos.	El sistema muestra el ranking vacío.	Se muestra el ranking vacío.	
2	CU: Ranking Usuario pulsa la opción "Volver" en la pantalla del Ranking.	Devuelve al usuario al menú principal.	Se vuelve al menú principal.	
3.1	CU: Ranking Usuario presiona un perfil con el que ya es amigo.	Aparece el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial y los pokémon de su equipo principal.	Aparece el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial y los pokémon de su equipo principal.	Se muestra el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial y los pokémon de su equipo principal.
3.2	CU: Ranking Usuario presiona un perfil que no es su amigo.	Aparece el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, los pokémon de su equipo principal y botón de añadir amigo.	Aparece el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, los pokémon de su equipo principal y botón de añadir amigo.	Se muestra el perfil completo del usuario, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, los pokémon de su equipo principal y botón de añadir amigo.
3.3	CU: Ranking Usuario presiona su propio perfil.	Aparece su perfil completo, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, y los pokémon de su equipo principal.	Aparece su perfil completo, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, y los pokémon de su equipo principal.	Se muestra su perfil completo, incluyendo foto de perfil, nombre de usuario, posición en el ranking mundial, y los pokémon de su equipo principal.

Ref.	Descripción	Resultado Esperado	Resultado Obtenido	Observaciones
4	CU: Ranking Usuario pulsa "añadir amigo"	El usuario se añade a su lista de amigos y ya no le aparece el botón de añadir amigo.	El usuario se añade a la lista de amigos del usuario y ya no le aparece el botón de añadir amigo para ese usuario.	
5	CU: Ranking Usuario pulsa la opción "Volver" en la pantalla del Perfil que ha seleccionado	Devuelve al usuario a la pantalla del Ranking.	Se vuelve a la pantalla del Ranking.	

Table 6: Casos de prueba de la funcionalidad Ranking

6 Bases de Datos

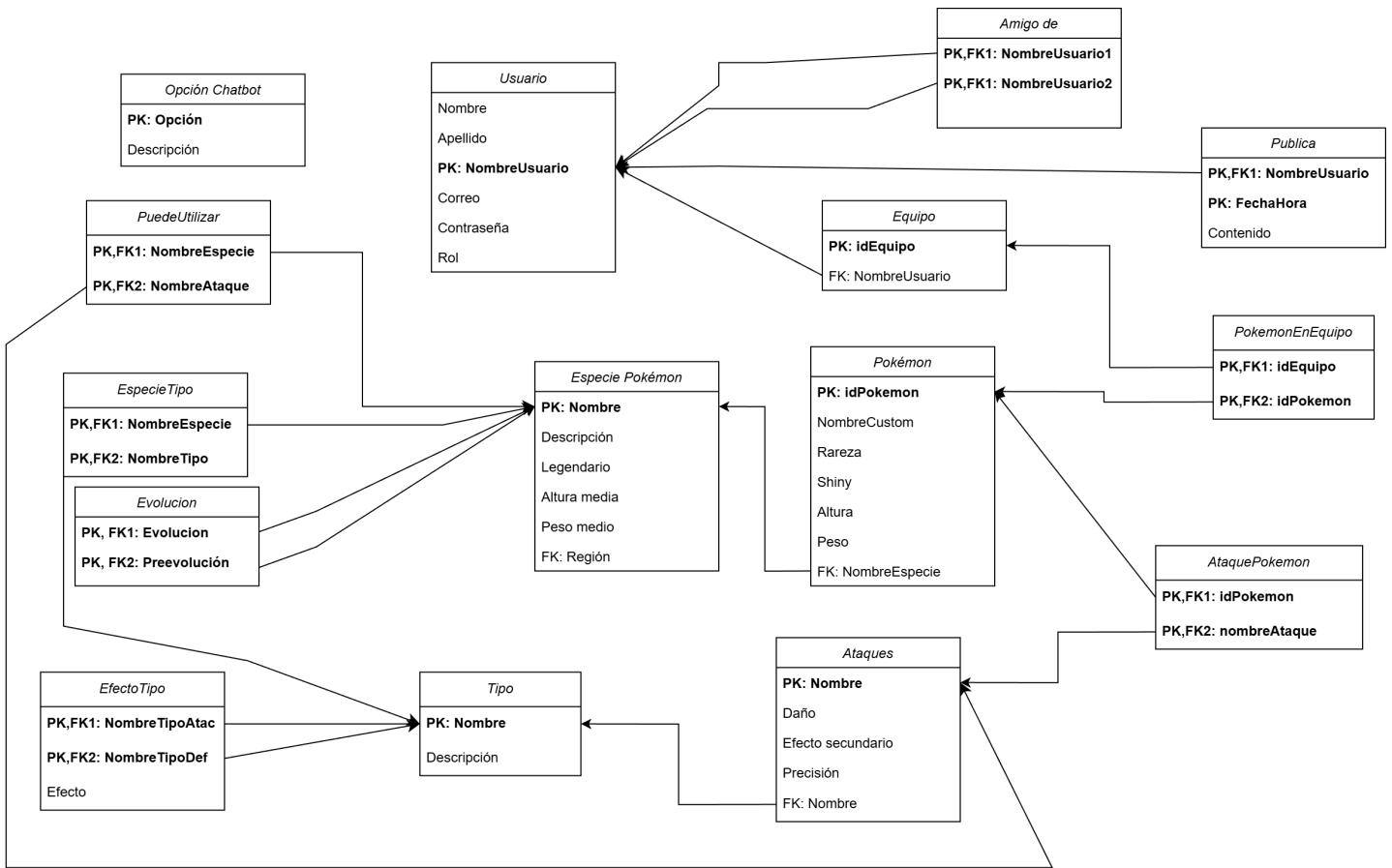


Figure 42: Base de datos

Partiendo del diagrama de modelo de dominio que habíamos obtenido con anterioridad y siguiendo los pasos de diseño para ejecutar correctamente la transformación, se ha logrado diseñar el diagrama relacional de la base de datos que vamos a utilizar. En el diagrama se aprecian todas las entidades con sus respectivos atributos y sus relaciones con otras entidades.

7 Diagrama de clases

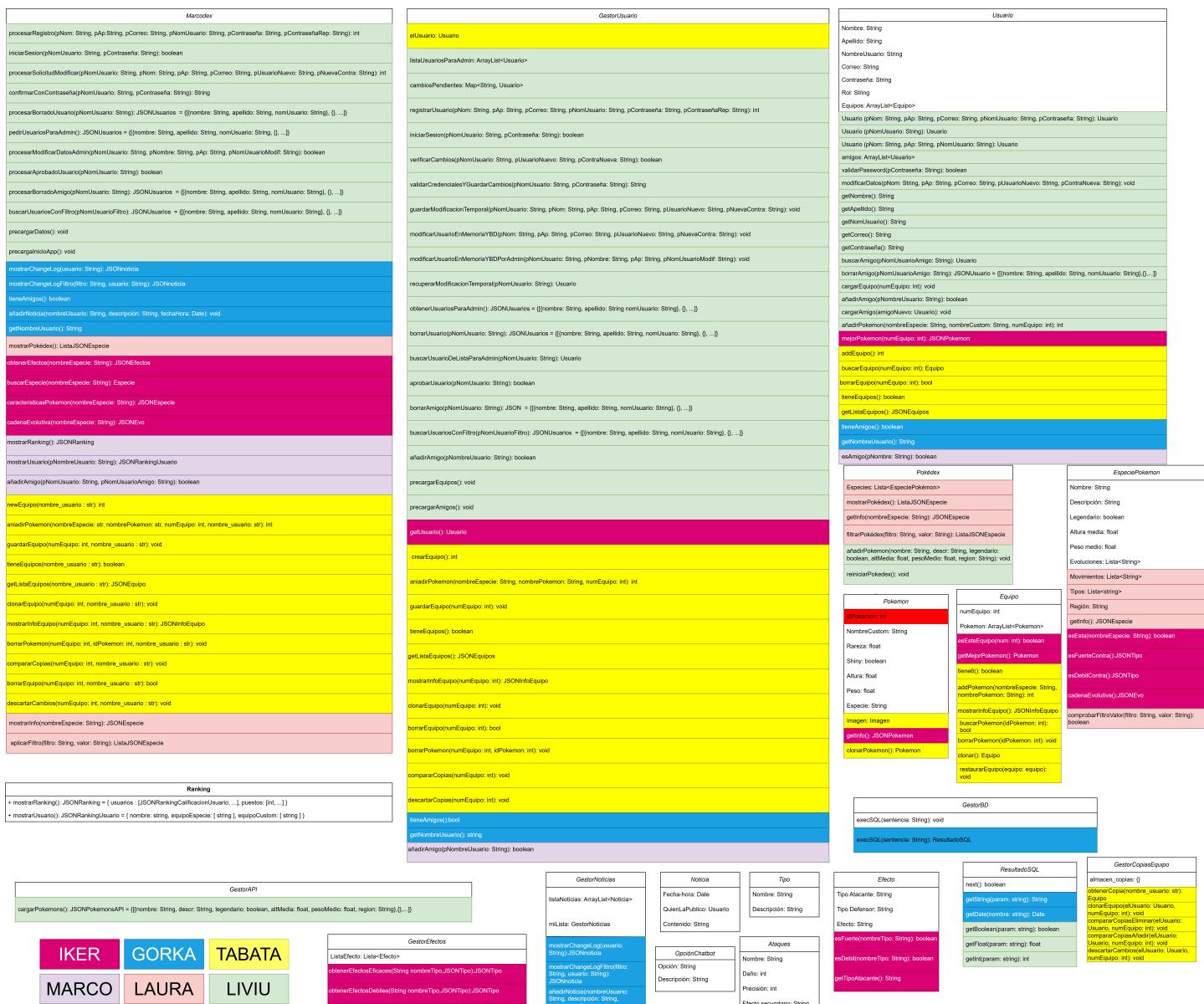
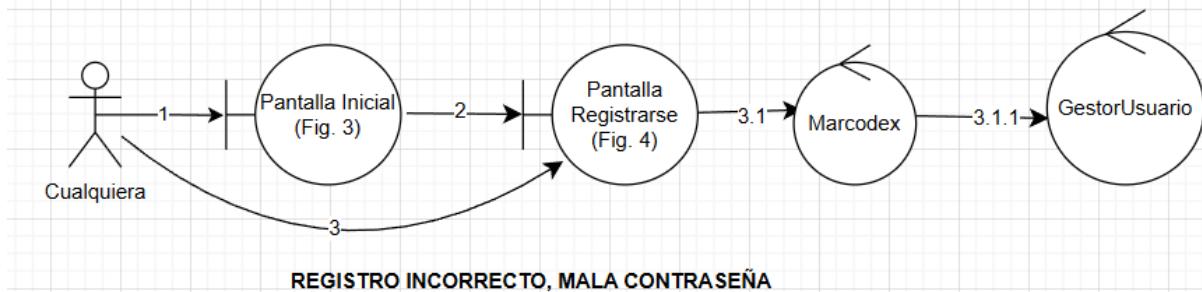


Figure 43: Diagrama de Clases

El diagrama de clases utiliza diferentes colores para indicar las aportaciones nuevas de cada integrante.

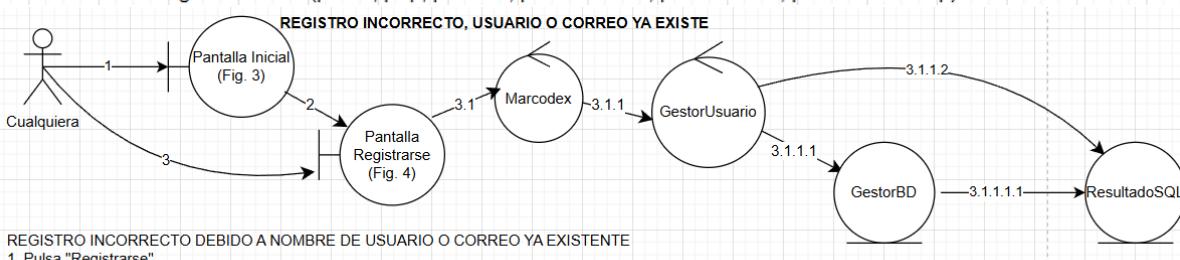
8 Diagramas de comunicación

8.1 Gestión de usuarios (Liviu)



REGISTRO INCORRECTO DEBIDO A CONTRASEÑA REPETIDA MAL ESCRITA

1. Pulsa "Registrarse"
2. new Pantalla Registrarse() (Fig. 4)
3. Introduce usuario y contraseña y Pulsa "Registrarse"
 - 3.1 procesarRegistro(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int
 - 3.1.1 registrarUsuario(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int



1. Pulsa "Registrarse"
2. new Pantalla Registrarse() (Fig. 4)
3. Introduce usuario y contraseña y Pulsa "Registrarse"
 - 3.1 procesarRegistro(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int
 - 3.1.1 registrarUsuario(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int
 - 3.1.1.1 execSQL(sql): ResultadoSQL // sql = SELECT * FROM USUARIO WHERE nombreUsuario=%pNomUsuario% OR correo=%pCorreo%"
 - 3.1.1.2 next(): boolean
 - [Si T] (significa que ya existe algún usuario con ese correo o nombre de usuario y registrarUsuario de GestorEquipos devolverá -2 para indicar mensaje de error en la interfaz)

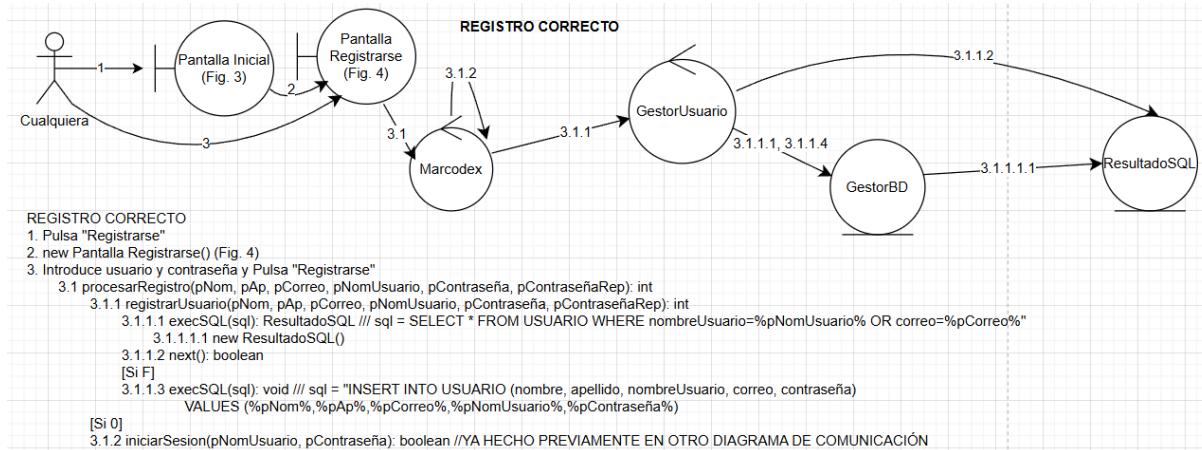
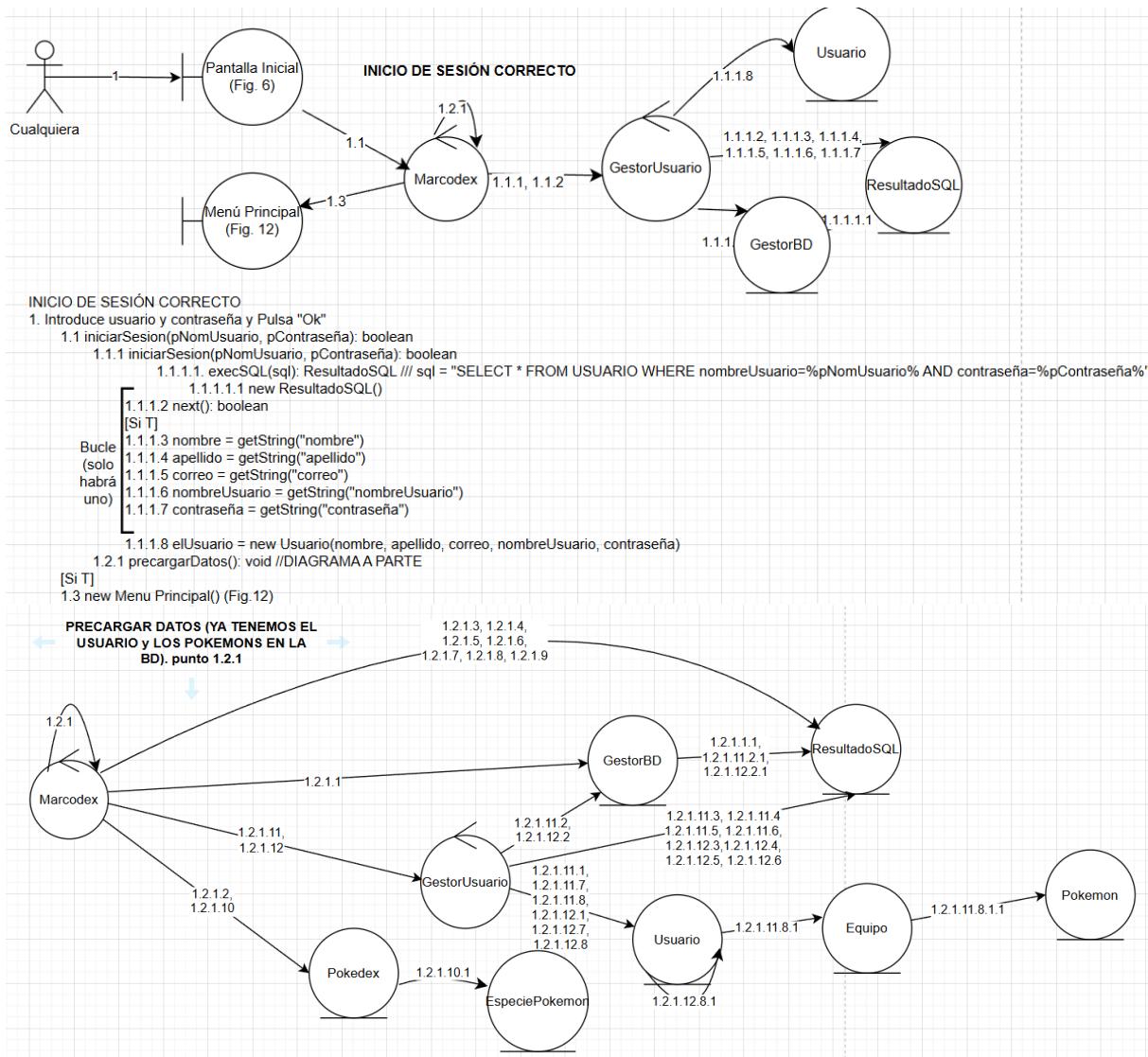


Figure 44: Diagramas Registrarse



```

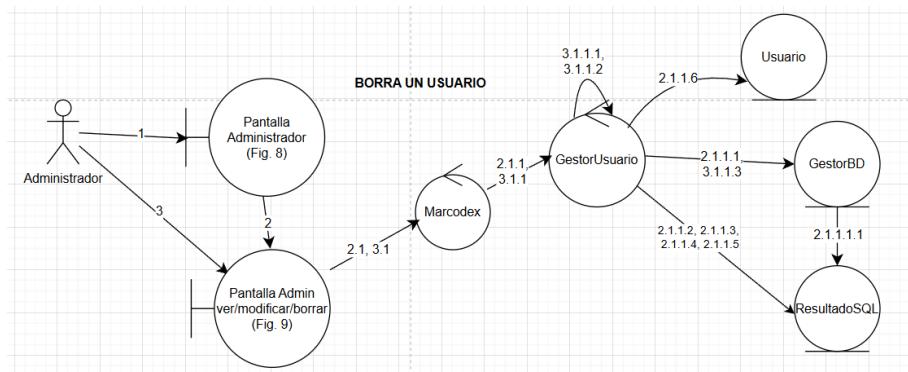
PRECARGAR DATOS
1.2.1 precargarDatos(): void
    1.2.1.1 execSQL(sql): ResultadoSQL /// sql = "SELECT * FROM EspeciePokemon E JOIN Region R ON E.Region = R.NombreRegion"
        1.2.1.1.1 new ResultadoSQL()
    1.2.1.2 reiniciarPokedex(): void
        1.2.1.3 next(): boolean
        1.2.1.4 nomPokemon = getString("nombre"): String
        1.2.1.5 descr = getString("descripcion"): String
        1.2.1.6 legendario = getBoolean("legendario"): boolean
        1.2.1.7 altMedia = getFloat("alturaMedia"): float
        1.2.1.8 pesoMedio = getFloat("pesoMedio"): float
        1.2.1.9 region = getString("region"): String
        1.2.1.10 añadirPokemon(nomPokemon, descr, legionario, altMedia, pesoMedio, region): void
            1.2.1.10.1 especies.add(new EspeciePokemon(nomPokemon, descr, legionario, altMedia, pesoMedio, region))

Bucle
    1.2.1.11 precargarEquipos(): void
        1.2.1.11.1 nombreUsuario = elUsuario.getNomUsuario(): String
        1.2.1.11.2 execSQL(sql2): ResultadoSQL
            1.2.1.11.2.1 new ResultadoSQL()
                1.2.1.11.3 next(): boolean
                    [Si T]
                        1.2.1.11.4 numEquipo = getInt("idEquipo"): int
                        1.2.1.11.5 nombreEspecie = getString("nombreEspecie"): String
                        1.2.1.11.6 nombreCustom = getString("nombreCustom"): String
                        1.2.1.11.7 cargarEquipo(numEquipo): void
                        1.2.1.11.8 añadirPokemon(nombreEspecie, nombreCustom, numEquipo): int
                            1.2.1.11.8.1 addPokemon(nombreEspecie, nombrePokemon): int
                                1.2.1.11.8.1.1 new Pokemon()

                    [Si F]
                        1.2.1.11.12 precargarAmigos(): void
                            1.2.1.12.1 nombreUsuario = elUsuario.getNomUsuario(): String
                            1.2.1.12.2 execSQL(sql3): ResultadoSQL
                                1.2.1.12.2.1 new ResultadoSQL()
                                    1.2.1.12.3 next(): boolean
                                        [Si T]
                                            1.2.1.12.4 nombreAmigo = getString("nombre"): String
                                            1.2.1.12.5 apellidoAmigo = getString("apellido"): String
                                            1.2.1.12.6 nombreUsuarioAmigo = getString("NombreUsuarioAmigo"): String
                                            1.2.1.12.7 amigoNuevo = new Usuario(nombreAmigo, apellidoAmigo, nombreUsuarioAmigo)
                                            1.2.1.12.8 elUsuario.cargarAmigo(amigoNuevo): void
                                                1.2.1.12.8.1 amigos.add(amigoNuevo)

```

Figure 45: Diagramas Iniciar Sesión



BORRA UN USUARIO

- Pulsar "Ver lista de Usuarios"
- new PantallaAdmin.ver/modificar/borrar() (Fig. 9)
 - pedirUsuariosParaAdmin(): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]
 - obtenerUsuariosParaAdmin(): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]] (en base al ArrayList, se devolverá el JSON)
 - execSQL(sql): ResultadoSQL // sql = "SELECT nombre, nombreUsuario FROM USUARIO"
 - new ResultadoSQL()

2.1.1.2 next(): boolean

Bucle [Si T]

2.1.1.3 nombre = getString("nombre"); String

2.1.1.4 apellido = getString("apellido"); String

2.1.1.5 nomUsuario = getString("nomUsuario"); String

2.1.1.6 listaUsuariosParaAdmin.add(new Usuario(nombre, apellido, nomUsuario))

- Pulsa un usuario en concreto y Pulsar "Borrar Cuenta"

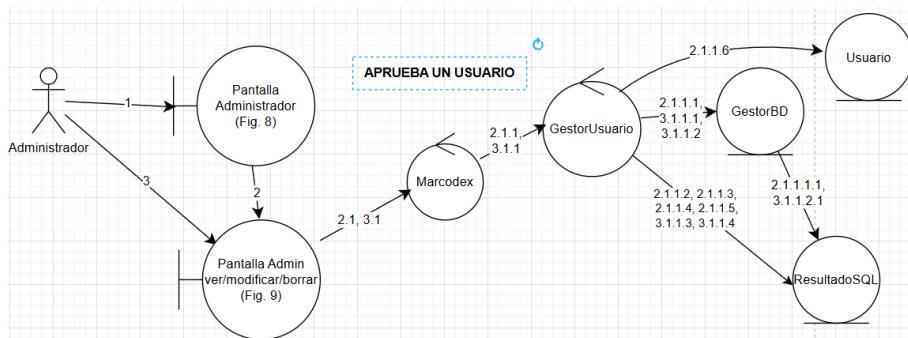
3.1 procesarBorradoUsuario(pNomUsuario): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]

3.1.1 borrarUsuario(pNomUsuario): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]] (en base al ArrayList, se devolverá el JSON)

3.1.1.1 usuarioABorrar = buscarUsuarioDeListaParaAdmin(pNomUsuario); Usuario

3.1.1.2 listaUsuariosParaAdmin.remove(usuarioABorrar) (lo pongo para que quede claro que se quiere borrar el usuario de la lista)

3.1.1.3 execSQL(sql): void // sql = "DELETE FROM USUARIO WHERE nombreUsuario=%pNomUsuario%"



APRUEBA UN USUARIO

- Pulsar "Ver lista de Usuarios"
- new PantallaAdmin.ver/modificar/borrar() (Fig. 9)
 - pedirUsuariosParaAdmin(): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]
 - obtenerUsuariosParaAdmin(): JSON = [{"nombre: String, apellido: String, nomUsuario: String}, {}...]] (en base al ArrayList, se devolverá el JSON)
 - execSQL(sql): ResultadoSQL // sql = "SELECT nombre, nombreUsuario FROM USUARIO"
 - new ResultadoSQL()

2.1.1.2 next(): boolean

Bucle [Si T]

2.1.1.3 nombre = getString("nombre")

2.1.1.4 apellido = getString("apellido")

2.1.1.5 nomUsuario = getString("nomUsuario")

2.1.1.6 listaUsuariosParaAdmin.add(new Usuario(nombre, apellido, nomUsuario))

- Pulsa un usuario en concreto y Pulsar "Aprobar Cuenta"

3.1 procesarAprobadoUsuario(pNomUsuario): boolean (este boolean sirve para mostrar un mensaje en la interfaz de que se ha aprobado la cuenta correctamente)

3.1.1 aprobarUsuario(pNomUsuario): boolean

3.1.1.1 execSQL(sql): void // sql = "UPDATE USUARIO SET rol='VERIF' WHERE nombreUsuario=%pNomUsuario%"

3.1.1.2 execSQL(sql2): ResultadoSQL // sql2 = "SELECT rol FROM USUARIO WHERE nombreUsuario=%pNomUsuario%"

3.1.1.2.1 new ResultadoSQL()

3.1.1.3 next(): boolean

3.1.1.4 rol = getString("rol") String (en base al String, se hará una comprobación si realmente ahora es VERIF, y se devolverá un boolean de true o false para la interfaz)

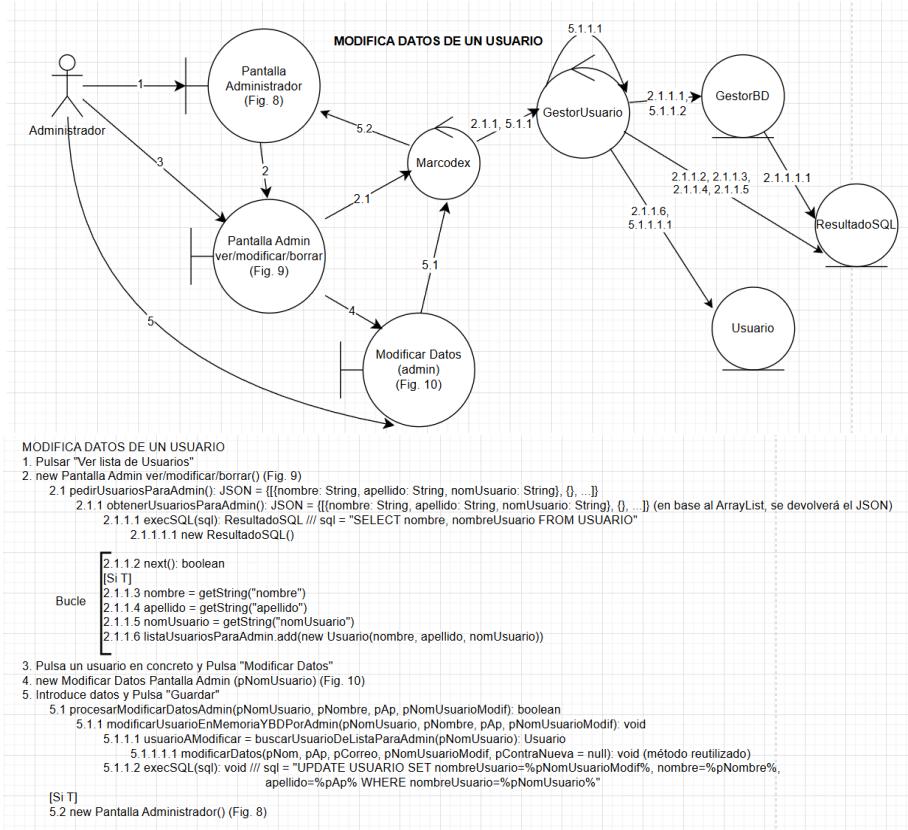


Figure 46: Diagramas Ver lista usuarios

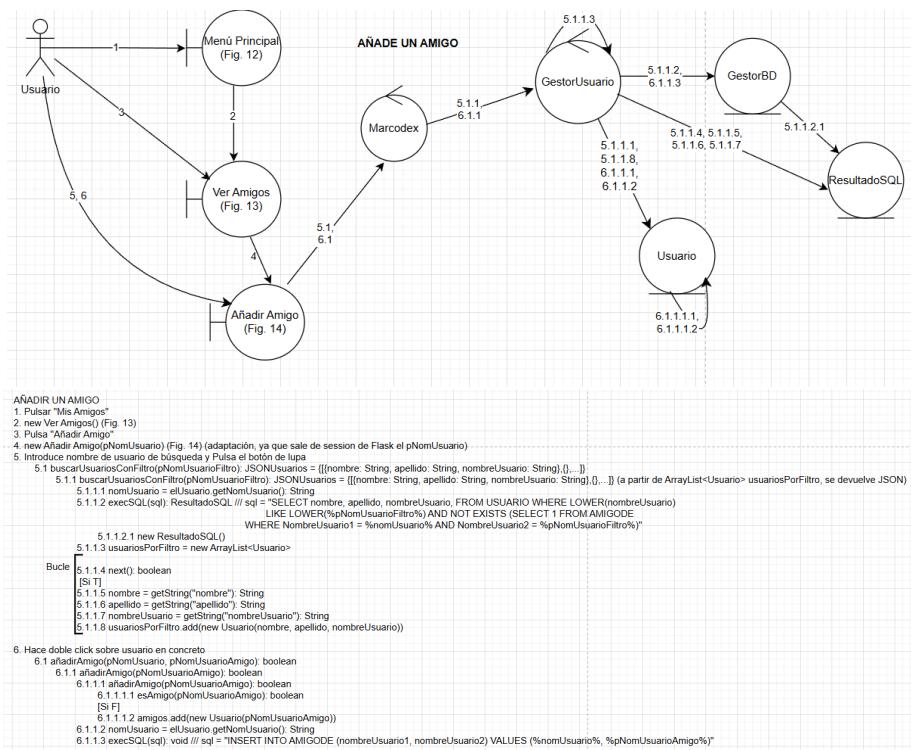
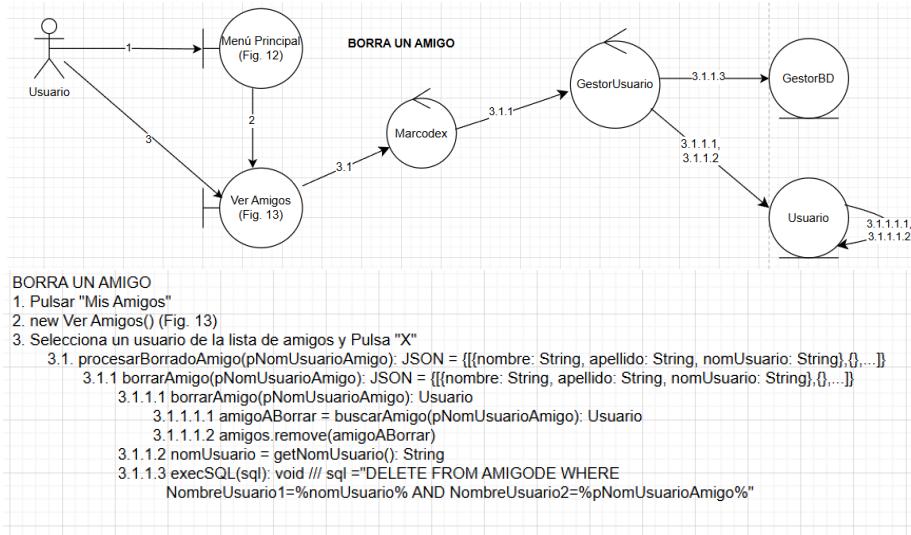
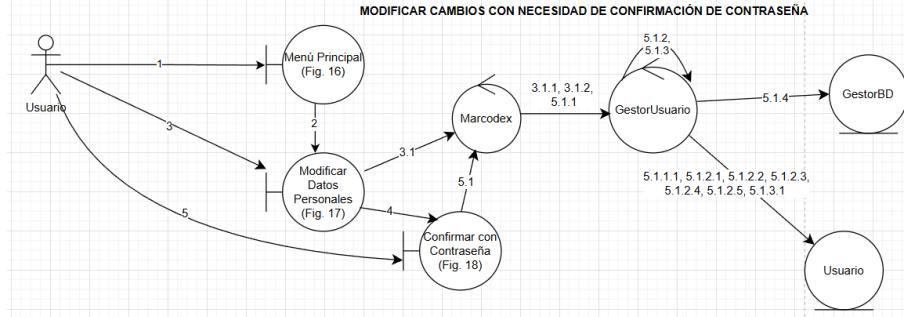
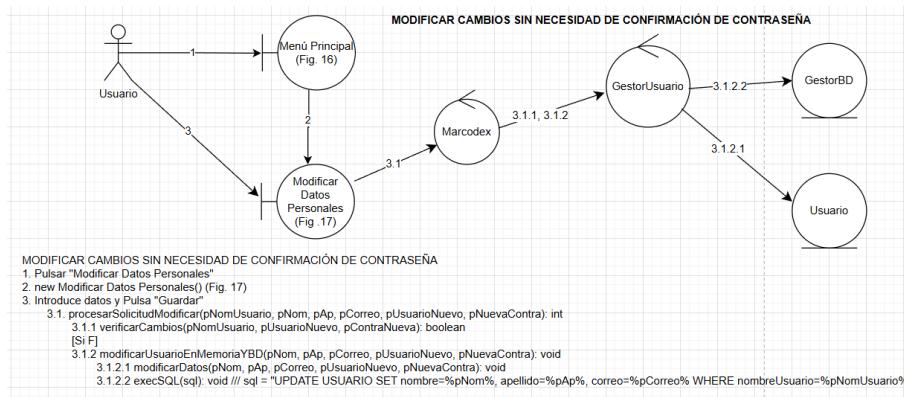


Figure 47: Diagramas Ver Amigos



MODIFICAR CAMBIOS CON NECESIDAD DE CONFIRMACIÓN DE CONTRASEÑA

- Pulsar "Modificar Datos Personales"
- new Modificar Datos Personales() (Fig. 17)
- Introduce datos y Palsa "Guardar"
 - procesarSolicitudModificar(pNomUsuario, pNom, pAp, pCorreo, pUsuarioNuevo, pNuevaContra): int
 - verificarCambios(pNomUsuario, pUsuarioNuevo, pContraNueva): boolean
[Si T]
 - guardarModificaciónTemporal(pNomUsuario, pNom, pAp, pCorreo, pUsuarioNuevo, pNuevaContra): void
- new Confirmar con Contraseña(pNomUsuario) (Fig. 18)
- Indroduce contraseña y Palsa "Confirmar"
 - confirmarConContraseña(pNomUsuario): String
[Si T]
 - validarCredencialesYGuardarCambios(pNomUsuario, pContraseña): String
 - validarPassword(pContraseña): boolean
 - datosUsuarioModif = recuperarModificacionTemporal(pNomUsuario): Usuario
 - elNombre = getNombre(): String (de datosUsuarioModif)
 - elAp = getApellido(): String (de datosUsuarioModif)
 - elCorreo = getCorreo(): String (de datosUsuarioModif)
 - elNomUsuario = getNomUsuario(): String (de datosUsuarioModif)
 - laContra = getContraseña(): String (de datosUsuarioModif)
 - modificarUsuarioEnMemoriaYBD(elNombre, elAp, elCorreo, elNomUsuario, laContra): void
 - modificarDatos(elNombre, elAp, elCorreo, elNomUsuario, laContra): void
 - execSQL(sql): void sql = "UPDATE USUARIO SET nombre=%elNombre%, apellido=%elAp%, correo=%elCorreo%, nombreUsuario=%elNomUsuario%, contraseña=%laContra% WHERE nombreUsuario=%pNomUsuario%"

Figure 48: Diagramas Modificar Datos Personales

Para la precarga de todos los pokémon obtenidos de la API, se ejecuta sólo cuando no hay una base de datos al iniciar la app por primera vez, por lo que al crearse, directamente hace las llamadas correspondientes para obtener los pokémon.

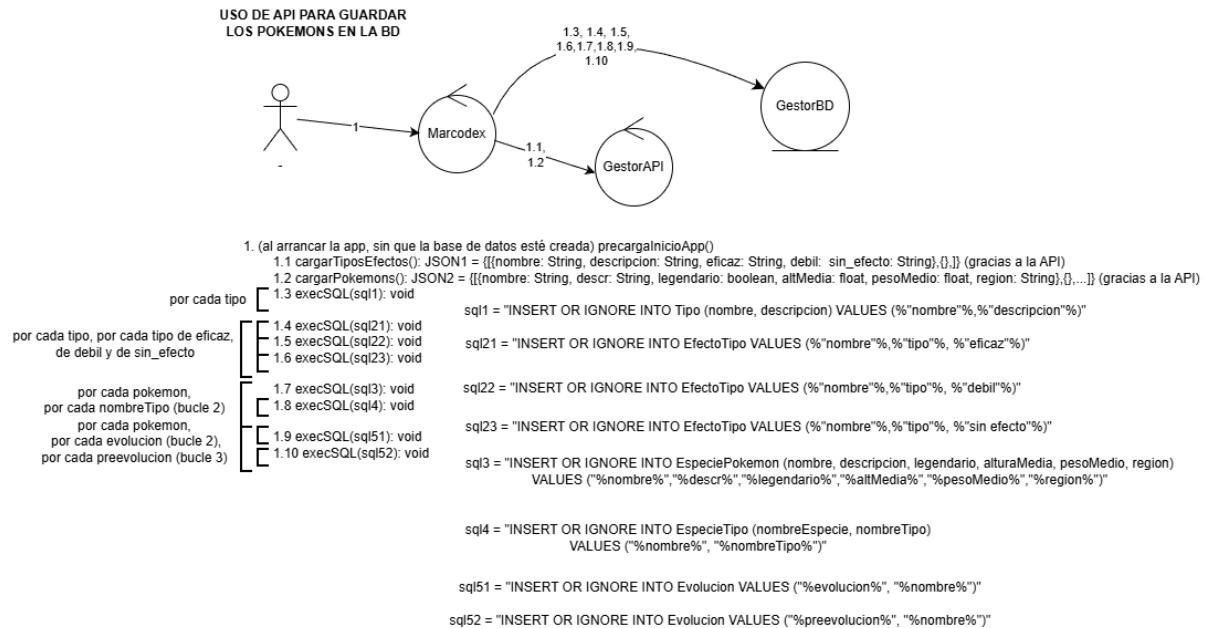


Figure 49: Diagrama Precarga Pokemons API a BD

8.2 Crear equipos Pokemon (Tabata)

Crear Equipos

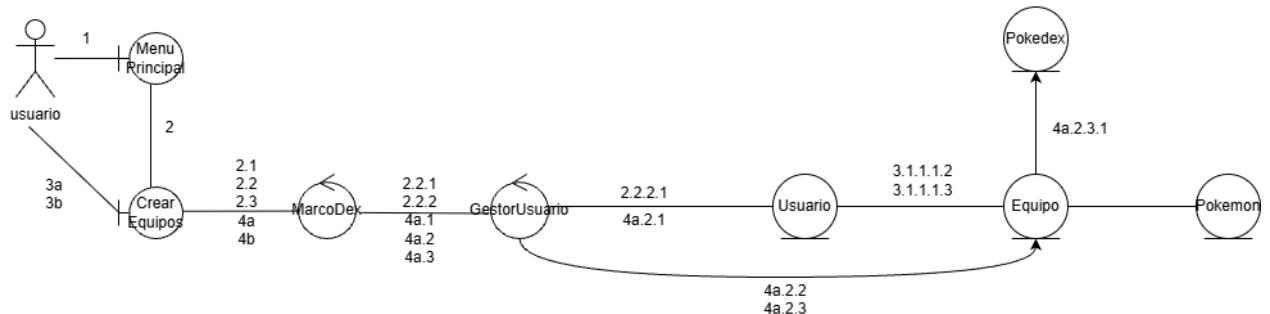


Figure 50: Crear Equipos

- 1.- Usuario pulsa "Crear equipo" del Menú Principal
- 2.- new_CrearEquipo(nombre_usuario: str)
 - 2.1.- mostrarPokedex(): JSONSpecies **DEFINIDO EN MOSTRAR POKEMONS**
 - 2.2.- newEquipo(nombre_usuario: str): int ; ese int es el nº del equipo
 - 2.2.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 2.2.2.- crearEquipo(): int
 - 2.2.2.1.- addEquipo():int
 - 2.3.- mostrarInfoPokemon(numEquipo: int): JSONInfoEquipo **DEFINIDO EN VER EQUIPOS**
- 3a.- Usuario selecciona un pokemon para añadirlo
- 4a.- aniadirPokemon(nombreEspecie: str, nombrePokemon: str, numEquipo:int): int
 - 4a.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 4a.2.- aniadirPokemon(nombreEspecie, nombrePokemon,numEquipo): int
 - 4a.2.1.- elEquipo = buscarEquipo(numEquipo: int): Equipo
 - 4a.2.2.- tiene6(): bool
 [Si False]
 - 4a.2.3.- addPokemon(nombreEspecie: str, nombrePokemon:str): int
 - 4a.2.3.1.- buscarEspecie(nombreEspecie: str): Especie
- 5a.- mostrarInfoPokemon(numEquipo): JSONEquipo **DEFINIDO EN VER EQUIPOS**
- 3b.- Usuario selecciona un pokemon para borrarlo
- 4b.- borrarPokemon(numEquipo: int, idPokemon: int): void **DEFINIDO EN VER EQUIPOS**

Figure 51: Crear Equipos 2

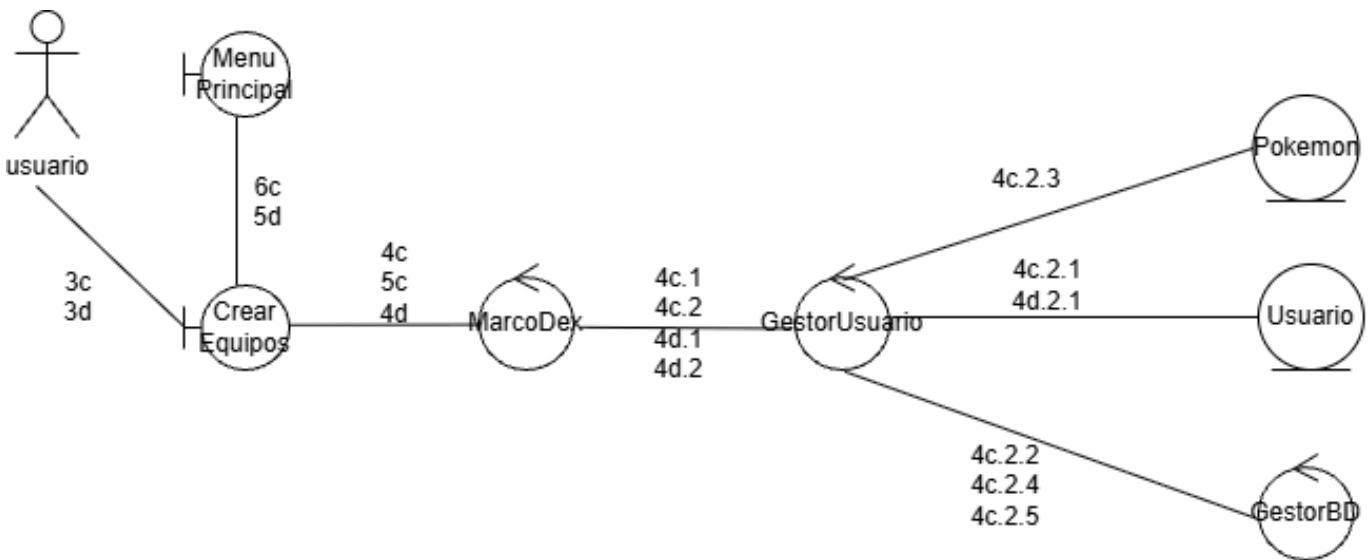


Figure 52: Crear Equipos 3

```

3c.- Usuario pulsa "Guardar" (si le da a guardar luego te manda al Menu principal)
4c.- guardarEquipo(numEquipo: int, nombre_usuario: str): void
    4c.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
    4c.2.- guardarEquipo(numEquipo: int): void
        4c.2.1.- elEquipo = buscarEquipo(numEquipo: int): Equipo
        4c.2.2.- id_bd_real = execSQL(sqlCrearEquipo): int
            (Por cada pokémon en elEquipo)
        4c.2.3.- getInfo(): JSONPokemon
        4c.2.4.- id_poki_real = execSQL(sqlInsertarPokemon): int
        4c.2.5.- execSQL(sqlInsertarRelacionPokemon)
5c.- añadirNoticia(nombreUsuario: str, descripción: str, fechaHora: Date): void
6c.- new_MenuPrincipal()
5d.- Usuario pulsa "Cancelar" (antes de darle a guardar)
4d.- descartarEquipo(numEquipo: int, nombre_usuario: str): void
    4d.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
    4d.2.- borrarEquipo(numEquipo: int): void
        4d.2.1.- elEquipo = buscarEquipo(numEquipo: int): Equipo
5d.- new_MenuPrincipal()

```

```

sqlCrearEquipo: INSERT INTO Equipo (numEquipo,
NombreUsuario) VALUES (%numEquipo%,
%nombre_usuario%)

sqlInsertarPokemon:
INSERT INTO Pokemon (numPokemon, NombreCustom, Rareza, Shiny, Altura, Peso,
NombreEspecie, Imagen) VALUES (valores de getInfo())

sqlInsertarRelacionPokemon:
INSERT INTO PokemonEnEquipo (idEquipointerno, idPokemon)
VALUES (id_bd_real, id_poki_real)

```

Figure 53: Crear Equipos 4

Se ha decidido que los equipos estén numerados en vez de ponerles un nombre, por lo tanto al crear un equipo se le asigna directamente el siguiente numero disponible en la listaEquipos del Usuario que es el tipo de dato int que devuelve el método newEquipo().

Luego cabe aclarar que el int que devuelve añadirPokemon es el id que se le pone al añadirlo al equipo en objetos y que en caso de no haberse podido añadir se devuelve un -1, -2 o -3 dependiendo de por qué no ha podido ser añadido.

Como esta lógica nos facilita la implementación en objetos pero a la hora de guardar en la base de datos habrá errores de clave primaria cada equipo y Pokémon tendrá su propia clave autoincremental interna en la base de datos. Así cuando se quiera hacer un cambio en BD simplemente se hace un SELECT para obtener el id real del objeto.

Disclaimer: La numeración se ha puesto según todas las posibilidades dentro de una misma vista Ejemplo: En la vista de CrearEquipo hay a,b,c,d posibilidades Y otra vista tendrá sus correspondientes a,b,c,etc.

Ver Equipos

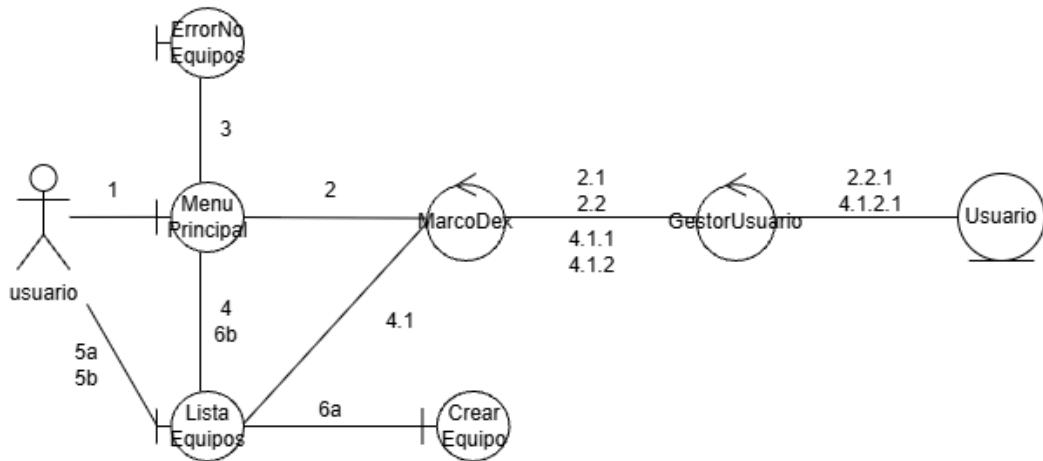


Figure 54: Ver Equipos

- 1.- Usuario pulsa "Mis Equipos" del Menú Principal
- 2.- tieneEquipos(nombre_usuario: str): bool
 - 2.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 2.2.- tieneEquipos(): bool
 - 2.2.1.- tieneEquipos(): bool
- [Si F]3.- new_ErrorNoEquipos()
- [Si T]4.- new_ListaEquipos(nombre_usuario: str)
 - 4.1.- getListEquipos(nombre_usuario: str): JSONEquipos
 - 4.1.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 4.1.2.- getListEquipos(): JSONEquipos
 - 4.1.2.1.- getListEquipos(): JSONEquipos
- 5a.- Usuario pulsa "Crear Equipo"
- 6a.- new_CrearEquipo()
- 5b.- Usuario pulsa "Volver al Menú Principal"
- 6b.- new_MenuPrincipal()

```

JSONEquipos: [
    {
        usuario: str,
        total_equipos: int,
        equipos: JSONEquipo []
    },
    ...
}

JSONEquipo: [
    {
        numEquipo: int,
        foto_principal: str,
        cantidad_pokemon: int []
    },
    ...
]
}

```

Figure 55: Ver Equipos

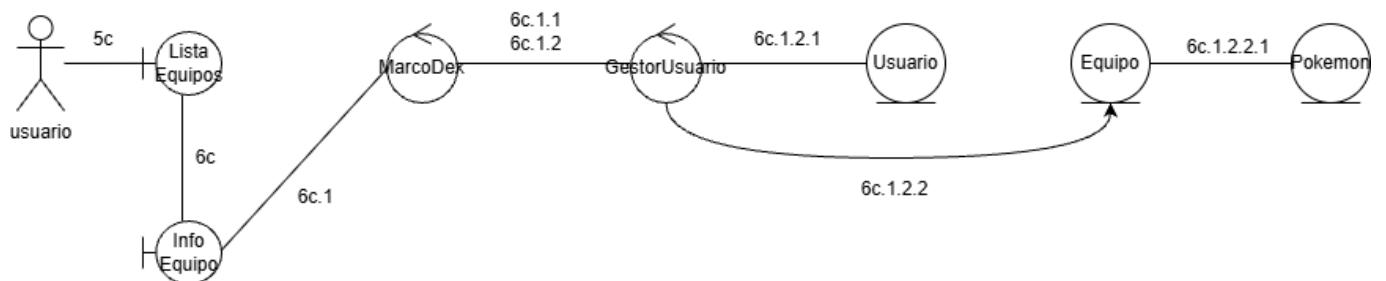


Figure 56: Ver Equipos 2

5c.- Usuario selecciona un equipo

6c.- new_InfoEquipo(numEquipo: int, nombre_usuario: str)

```
6c.1.- mostrarInfoEquipo(numEquipo: int, nombre_usuario: str): JSONInfoEquipo
    6c.1.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
    6c.1.2.- mostrarInfoEquipo(numEquipo: int): JSONInfoEquipo
        6c.1.2.1.- elEquipo = buscarEquipo(numEquipo: int): Equipo
        6c.1.2.2.- mostrarInfoEquipo(): JSONInfoEquipo
            6c.1.2.2.1.- getInfo(): JSONPokemon
```

```
JSONInfoEquipo: {
    [
        pokemon_id: int,
        nombre_custom: str,
        shiny: bool,
        especie: str,
        imagen: str
    ],
    [
        ...
    ]
}
```

Figure 57: Ver Equipos 3

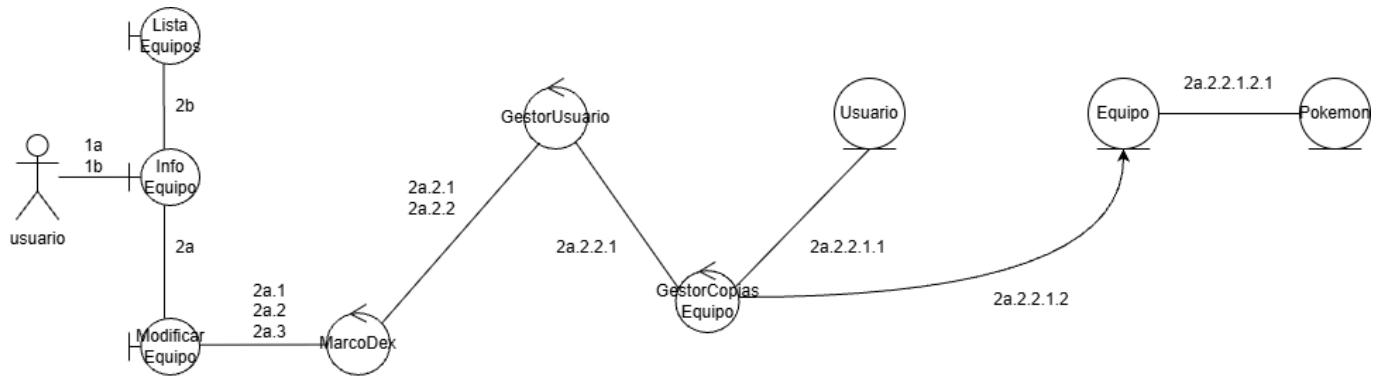


Figure 58: Ver Equipos 4

1a.- Usuario pulsa "Modificar equipo"

2a.- new_ModificarEquipo(numEquipo: int, nombre_usuario: str)

```
2a.1.- mostrarPokedex(): JSONSpecies DEFINIDO EN VER POKEMONS
2a.2.- clonarEquipo(numEquipo: int): void
    2a.2.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
    2a.2.2.- clonarEquipo(numEquipo: int): void
        2a.2.2.1.- clonarEquipo(elUsuario: Usuario, numEquipo: int): void
            2a.2.2.1.1.- equipoOriginal = buscarEquipo(numEquipo: int): Equipo
            2a.2.2.1.2.- copiaOriginal = clonar(EquipoOriginal: Equipo): Equipo
                (Por cada pokemon en el equipo)
            2a.2.2.1.2.1.- clonarPokemon(): Pokemon
```

2a.3.- mostrarInfoEquipo(numEquipo): JSONInfoEquipo DEFINIDO EN VER EQUIPOS (JUSTO ARRIBA)

1b.- Usuario da a volver

2b.- new_ListaEquipos(nombre_usuario: str)

Figure 59: Ver Equipos 5

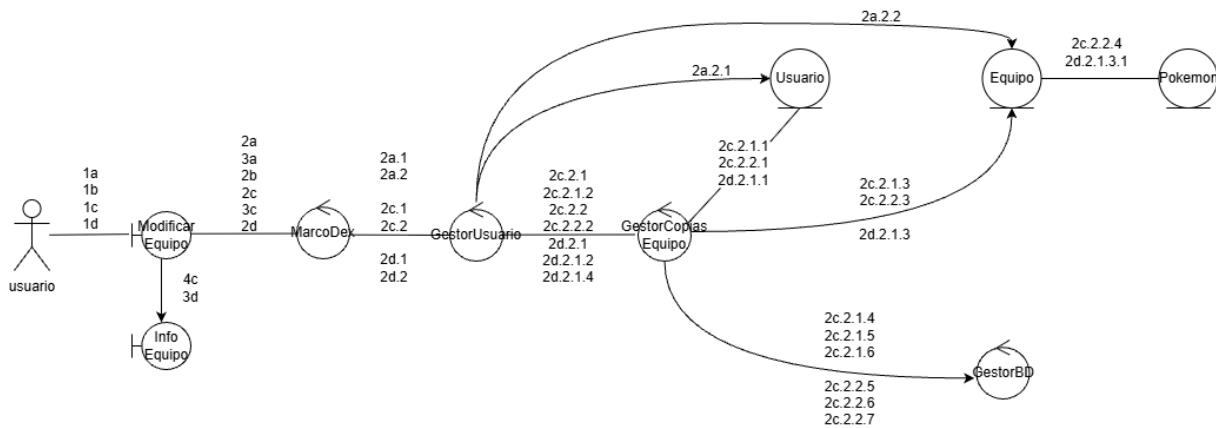


Figure 60: Ver Equipos 6

- 1a.- Usuario selecciona pokemon para borrarlo
- 2a.- borrarPokemon(numEquipo: int, idPokemon: int): void
 - 2a.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 2a.2.- borrarPokemon(numEquipo: int, idPokemon: int): void
 - 2a.2.1.- elEquipo = buscarEquipo(numEquipo: int): Equipo
 - 2a.2.2.- borrarPokemon(idPokemon: int): void
- 3a.- mostrarInfoEquipo(numEquipo): JSONInfoEquipo

- 1b.- Usuario selecciona un pokemon para añadirlo
- 2b- añadirPokemon(nombreEspecie: str, nombrePokemon: str, numEquipo: str): int **DEFINIDO EN CREAR EQUIPO**

- 1c.- Usuario pulsa "Modificar"
- 2c.- compararCopias(numEquipo: int, nombre_usuario: str): void
 - 2c.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 - 2c.2.- compararCopias(numEquipo: int): void
 - 2c.2.1.- compararCopiasEliminar(elUsuario: Usuario, numEquipo: int, db: Connection): void
 - 2c.2.1.1.- equipoEditado = buscarEquipo(numEquipo: int): Equipo
 - 2c.2.1.2.- copia = obtener_copia(elUsuario.nombre_usuario : str): Equipo

(Por cada pokemon en la copia)
 - 2c.2.1.3.- equipoEditado.buscarPokemon(pokemon.pokemon_id: int): bool
 [If False]
 - 2c.2.1.4.- execSQL(sqlObtenerIdPokemon)
 - 2c.2.1.5.- execSQL(sqlBorrarRelacionPokemon)
 - 2c.2.1.6.- execSQL(sqlBorrarPokemon)
 - 2c.2.2.- compararCopiasAñadir(elUsuario: Usuario, numEquipo: int): void
 - 2c.2.2.1.- equipoEditado = buscarEquipo(numEquipo: int): Equipo
 - 2c.2.2.2.- copia = obtener_copia(elUsuario.nombre_usuario: str): Equipo

(Por cada pokemon en equipoEditado)
 - 2c.2.2.3.- copia.buscarPokemon(pokemon.pokemon_id: int): bool
 - 2c.2.2.4.- info = getInfo()
 - 2c.2.2.5.- id_poki_real = execSQL(sqlInsertarPokemon)
 - 2c.2.2.6.- id_bd_real = execSQL(sqlObtenerId)
 - 2c.2.2.7.- execSQL(sqlInsertarRelacionPokemon)
- 3c.- añadirNoticia(nombreUsuario: str, descripcion: str, fechaHora: Date): void
- 4c.- new_InfoEquipo(numEquipo: int)

Figure 61: Ver Equipos 7

1d.- Usuario pulsa "Cancelar"
2d.- descartarEquipo(numEquipo: int): void
 2d.1.- getMyGestorUsuario(nombre_usuario: str, db: Connection): gestorUsuario
 2d.2.- descartarEquipo(numEquipo: int): void
 2d.2.1.- descartarEquipo(elUsuario: Usuario, numEquipo: int): void
 2d.2.1.1.- elEquipo = buscarEquipo(numEquipo: int): void
 2d.2.1.2.- copia = obtener_copia(elUsuario.nombre_usuario : str): Equipo
 2d.2.1.3.- elEquipo.restaurarEquipo(copia: Equipo): void
 (Por cada Pokemon en copia)
 2d.2.1.3.1.- pokemon.clonarPokemon(): void
 2d.2.1.4.- finalizarEdicion(nombre_usuario: str): void
3d.- new_InfoEquipo(numEquipo: int)

sqlBorrarRelacionPokemon: DELETE FROM PokemonEnEquipo WHERE numPokemon = %pokemon_id%

sqlBorrarPokemon: DELETE FROM Pokemon WHERE numPokemon = %pokemon_id%

sqlInsertarPokemon:
INSERT INTO Pokemon (numPokemon, NombreCustom, Rareza, Shiny, Altura, Peso, NombreEspecie, Imagen) VALUES (valores de getInfo())

sqlInsertarRelacionPokemon:
INSERT INTO PokemonEnEquipo (idEquipointerno, idPokemon)
VALUES (%id_bd_real%, %id_poki_real%)

sqlObtenerIdEquipo:
SELECT idEquipo FROM Equipo WHERE numEquipo = %numEquipo%
AND NombreUsuario = %nombre_usuario%

sqlObtenerIdPokemon:
SELECT idPokemon FROM Pokemon WHERE numPokemon = %pokemon_id% AND NombreCustom = %nombre_custom%

Figure 62: Ver Equipos 8

Aquí cabe destacar que se ha creado una copia profunda del equipo original antes de que el usuario edite el equipo, esto con el fin de poder restaurarlo en caso de que el usuario pulse "Cancelar". Además, se ha añadido una clase controlador GestorCopiasEquipo para poder manejar mejor la lógica de detección de cambios para luego así reflejarlos en la base de datos comparando los dos equipos.

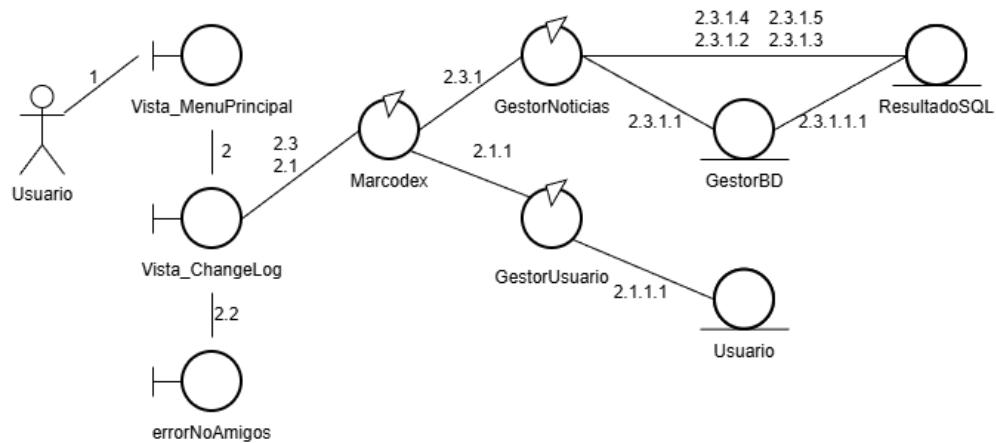
Por ello, el borrado y añadido de pokémon se sigue manejando desde el gestorUsuario, efectuando los cambios que el usuario hace sobre el equipo de la lista del usuario. Así las modificaciones ya están implementadas en objetos antes de hacer los cambios en la base de datos.

*Se ha añadido una interfaz nueva llamada Modificar Equipo

Relaciones Vista-Figura:

1. Menú Principal: *Figure 12*
2. Crear Equipos: *Figure 19*
3. Error No Equipos: *Figure 21*
4. Lista Equipos: *Figure 22*
5. Info Equipo: *Figure 23*
6. Modificar Equipo: *Figure 24*

8.3 ChangeLog (Gorka)



1. El usuario pulsa el botón "ChangeLog" en el menú principal
2. new Vista_ChangeLog()
 - 2.1. tieneAmigos(NombreUsuario: string):bool
 - 2.1.1. tieneAmigos(NombreUsuario: string):bool
 - 2.1.1.1. tieneAmigos(NombreUsuario: string):bool
 - [Si F] 2.2. new errorNoAmigos()
 - [Si T] 2.3. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia
 - 2.3.1. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia
 - 2.3.1.1. execSQL(sqlNoticia): ResultadoSQL
 - 2.3.1.1.1. new ResultadoSQL

- 2.3.1.2. next():bool
 - 2.3.1.3. getString("NombreUsuario"): string
 - 2.3.1.4. getDate("FechaHora"): Date
 - 2.3.1.5. getString("Contenido"): string

```

sqlNoticia:    SELECT NombreUsuario, FechaHora, Contenido
               FROM Publica
               WHERE FechaHora >= DATE_SUB(CURDATE(), INTERVAL 1
                                              MONTH)
               AND NombreUsuario IN (
               SELECT nombreUsuario1
               FROM AmigoDe
               WHERE nombreUsuario2 = %nombreUsuario%
               AND NombreUsuario LIKE % filtro% %
               ORDER BY FechaHora DESC;

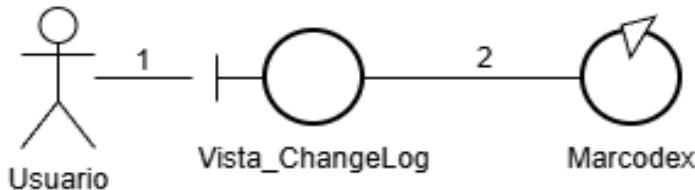
```

JSONnoticia: { [date,string,string] }

Figure 63: ChangeLog mostrar noticias

En este diagrama de comunicación se muestra como mostrar las noticias que se hayan creado en el último mes. Se ha decidido mostrar solo el último mes porque mostrar todo sería trabajar con una cantidad demasiado elevada de datos, además, creemos que con los cambios de un mes es suficiente.

En el diagrama, primero, se ve si el usuario tiene amigos. Si no los tiene, se muestra un mensaje de error y si los tiene, se procede a cargar las noticias de la base de datos.



1. El usuario pone algo en la barra de búsqueda y le da al botón de búsqueda
2. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia

```

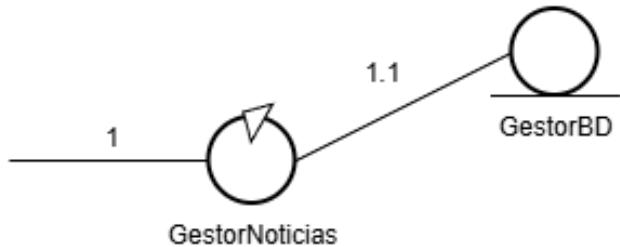
sqlNoticia:    SELECT NombreUsuario, FechaHora, Contenido
               FROM Publica
               WHERE FechaHora >= DATE_SUB(CURDATE(),
                                              INTERVAL 1 MONTH)
               AND NombreUsuario LIKE % filtro% %
               ORDER BY FechaHora DESC;

```

JSONnoticia: { [date,string,string] }

Figure 64: ChangeLog-filtro de usuarios

En este diagrama se muestra como filtrar los usuarios por el nombre del usuario. Es muy parecido al anterior, pero esta vez no hace falta ver si tiene amigos porque si ha podido acceder a esta vista quiere decir que tiene amigos y en la SELECT se añade un WHERE con el filtro.



1. añadirNoticia(nombreUsuario, descripción)

 1.1. ExecSQL(sqlNuevaNoticia): void

sqlNuevaNoticia: INSERT INTO Publica (nombreUsuario, FechaHora, Contenido)
 VALUES (%nombreUsuario%, NOW(), %descripcion%)

Figure 65: ChangeLog-añadir noticia

En este diagrama se muestra como se añade una nueva noticia a la base de datos. Este método puede ser llamado desde diversos sitios como al crear un equipo o al modificarlo, por eso, se ha puesto el actor "cambio" que representa todas esas llamadas.

8.4 Lista Pokémon y filtros (Laura)

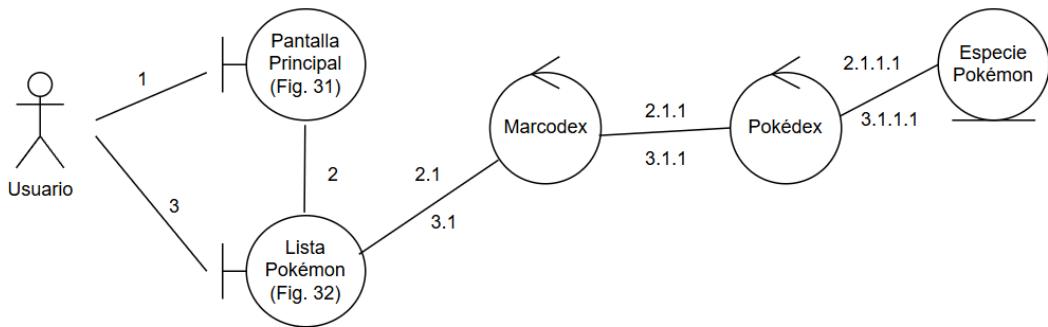


Figure 66: Ver Lista Pokémon

1. Usuario pulsa "Lista Pokémon" en la pantalla principal
2. new Lista_Pokémon()
2.1 mostrarPokédex(): ListaJSONEspecie
2.1.1 mostrarPokédex(): ListaJSONEspecie
[Bucle]
2.1.1.1 getInfo(): JSONEspecie
3. Usuario selecciona un Pokémon de la Lista Pokémon
3.1 mostrarInfo(nombreEspecie): JSONEspecie
3.1.1 getInfo(nombreEspecie): JSONEspecie
IF [el nombre es igual]
T 3.1.1.1 getInfo(): JSONEspecie
F Devuelve "Pokemon no encontrado"

Este diagrama de comunicación se corresponde con la funcionalidad de ver todos los pokémon de la Pokédex y de visualizar los datos sobre un Pokémon en específico, que elija el usuario.

JSONEspecie:

```
{
  "nombre": "string",
  "descripcion": "string",
  "legendario": "bool",
  "alturaMedia": "float",
  "peso": "float",
  "movimientos": ["string", "int", "int", "string"],
  "tipo": ["string"],
  "imagen": "string"
}
```

*JSONEspecie devuelve la información de Especie Pokémon, en concreto: nombre, descripción, legendario, altura media, peso medio, una lista de ataques y una lista de tipos.

ListaJSONEspecie:

```
{
  "nombre": "string",
  "descripcion": "string",
  "legendario": "bool",
  "alturaMedia": "float",
  "peso": "float",
  "movimientos": ["string", "int", "int", "string"],
  "tipo": ["string"],
  "imagen": "string"
}
```

*ListaJSONEspecie devuelve una lista de JSONEspecie

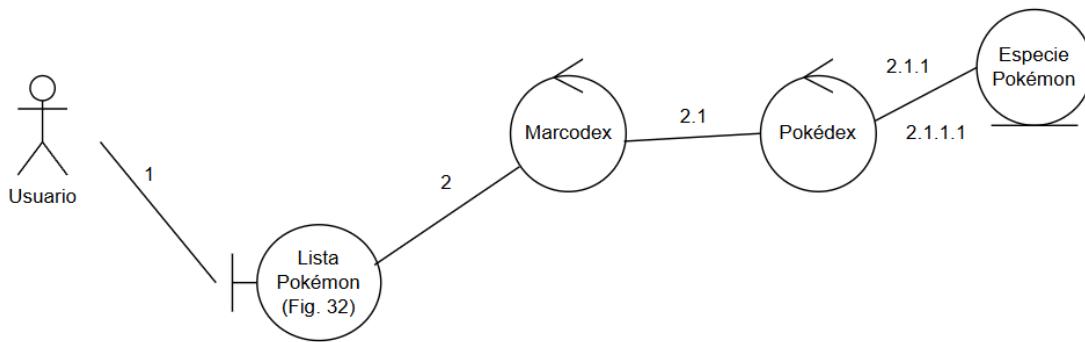


Figure 67: Filtrar Lista Pokémon

1. Usuario selecciona "Filtrar por nombre" / "Filtrar por tipo" en la pantalla de la Pokédex e introduce un valor de filtro (String)
2. aplicarFiltro(String filtro, String valor): ListaJSONEspecie
 - 2.1 filtrarPokedex(String filtro, String valor): ListaJSONEspecie
 [Bucle para cada Especie Pokémon]
 - 2.1.1 comprobarFiltroValor(String filtro, String valor): bool
 [Si True]
 - 2.1.1.1 añadir JSONEspecie a la lista filtrada

Este diagrama representa la funcionalidad de filtrar pokémon (teniendo en cuenta que el usuario ya está en la vista de Ver Lista Pokémon). Para filtrar, el usuario tendrá que elegir si quiere filtrar por nombre, por tipo o por habilidad y después introducirá un nombre, un tipo o una habilidad y buscará el/los pokémon. En caso de que no exista el valor que introduzca el usuario, la lista de pokémon saldrá vacía.

8.5 Chatbot (Iker)

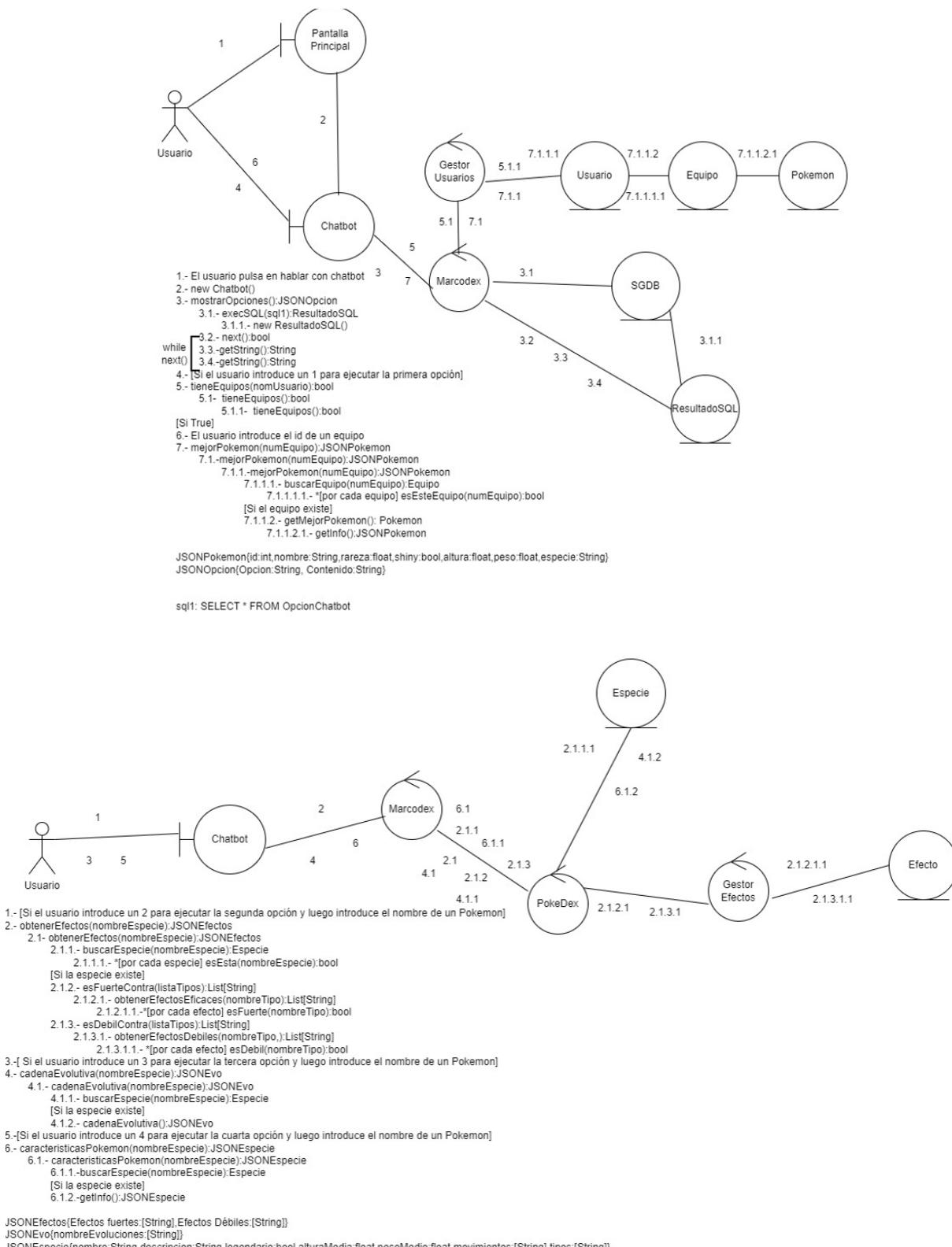


Figure 68: ChatBot

En estos diagrama de comunicación se muestra la funcionalidad de la interfaz del ChatBot y las formas en las que el usuario puede interactuar dentro de ella. Como ya es sabido, el usuario dispone de cinco opciones con las que puede visualizar diferentes estadísticas, como por ejemplo el Pokémon más fuerte de un equipo o las habilidades de un Pokémon en concreto. En el diagrama mismo se recogen todas estas opciones. Cabe recalcar que las cinco opciones que muestra el ChatBot tienen que ser precargadas desde la base de datos cada vez que se accede a la página para que se las pueda mostrar al usuario.

8.6 Ranking (Marco)

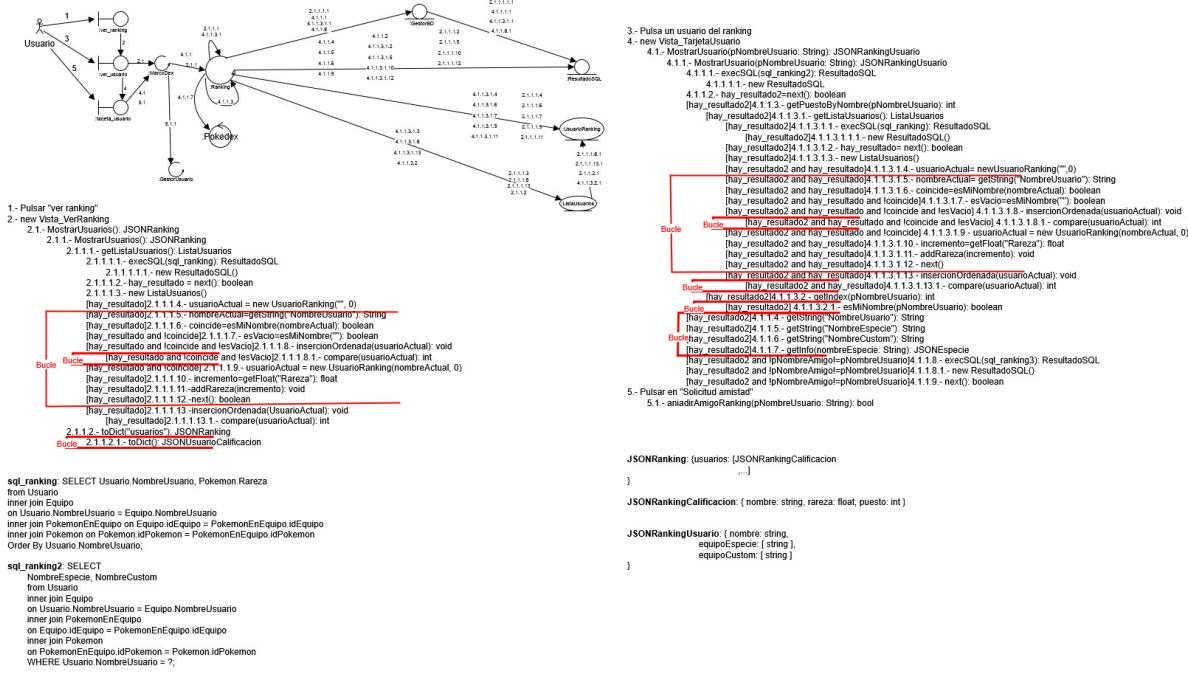
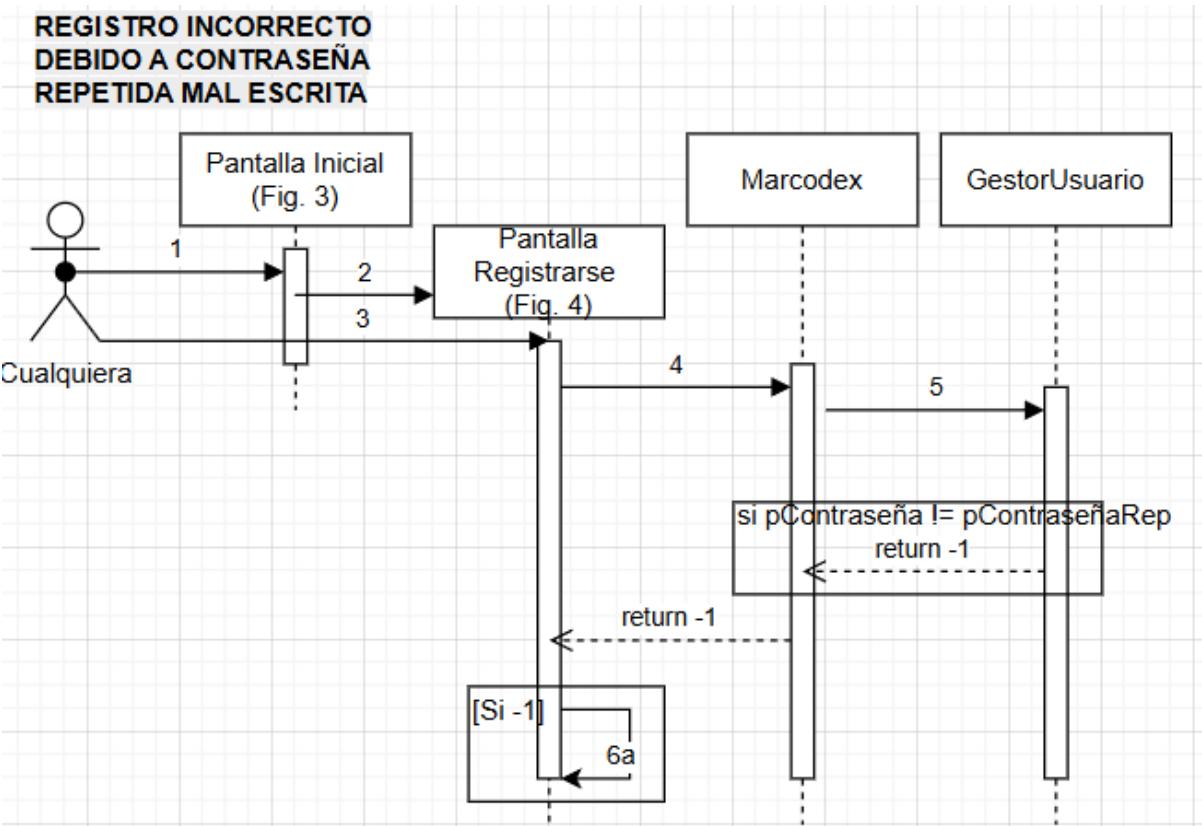


Figure 69: Ranking

En esta figura se puede observar el diagrama de comunicación de la funcionalidad del ranking. Como se puede ver, es necesario precargar los usuarios de la base de datos para posteriormente visualizarlos en el ranking. El diagrama, además del ranking, también captura los métodos tanto para visualizar perfiles como para añadir nuevos amigos. Cabe recalcar que también es necesario precargar los datos de un usuario desde la base de datos cuando se quiere ver el perfil de este. Como se ve en el diagrama, la base de datos es modificada cuando un usuario añade un nuevo amigo, representando este cambio en la base de datos del usuario.

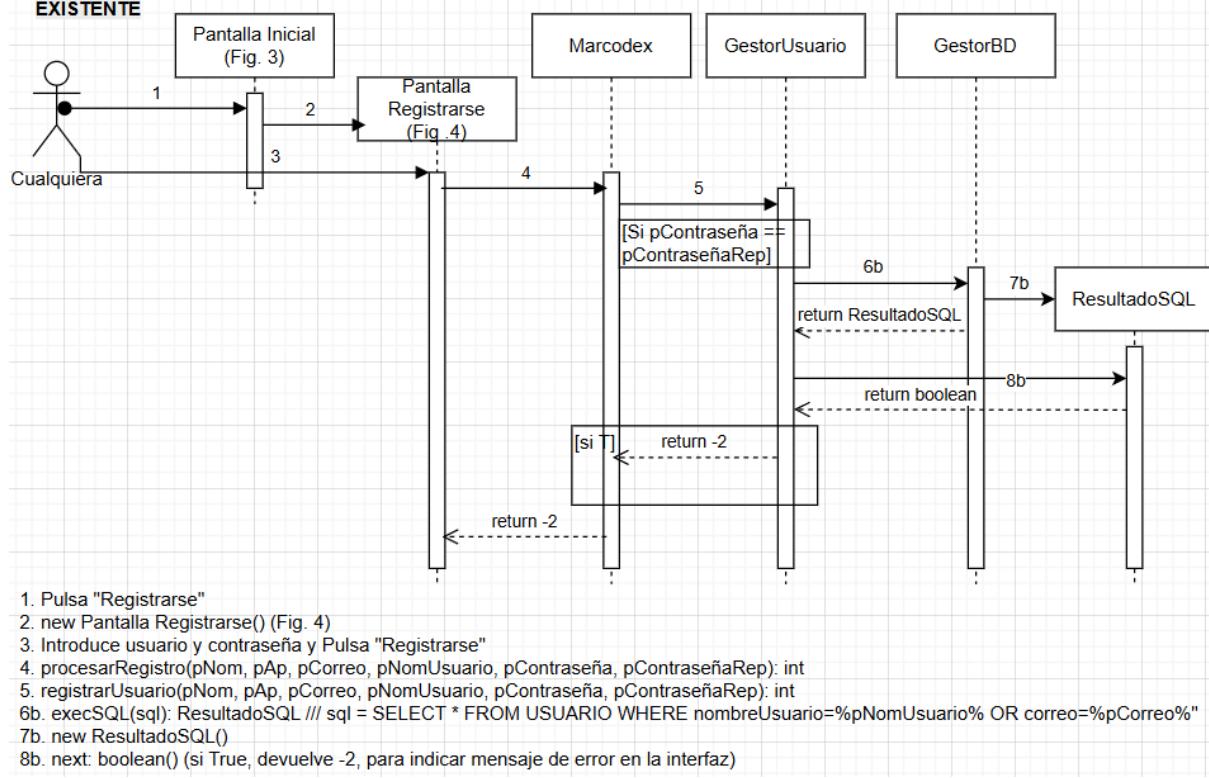
9 Diagramas de secuencia

9.1 Gestión de usuarios (Liviu)



1. Pulsa "Registrarse"
2. new Pantalla Registrarse() (Fig. 4)
3. Introduce usuario y contraseña y Pulsa "Registrarse"
4. procesarRegistro(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int
5. registrarUsuario(pNom, pAp, pCorreo, pNomUsuario, pContraseña, pContraseñaRep): int
- 6a. redirigir Pantalla Registrarse (Fig. 4)

**REGISTRO INCORRECTO
DEBIDO A NOMBRE DE
USUARIO O CORREO YA
EXISTENTE**



REGISTRO CORRECTO

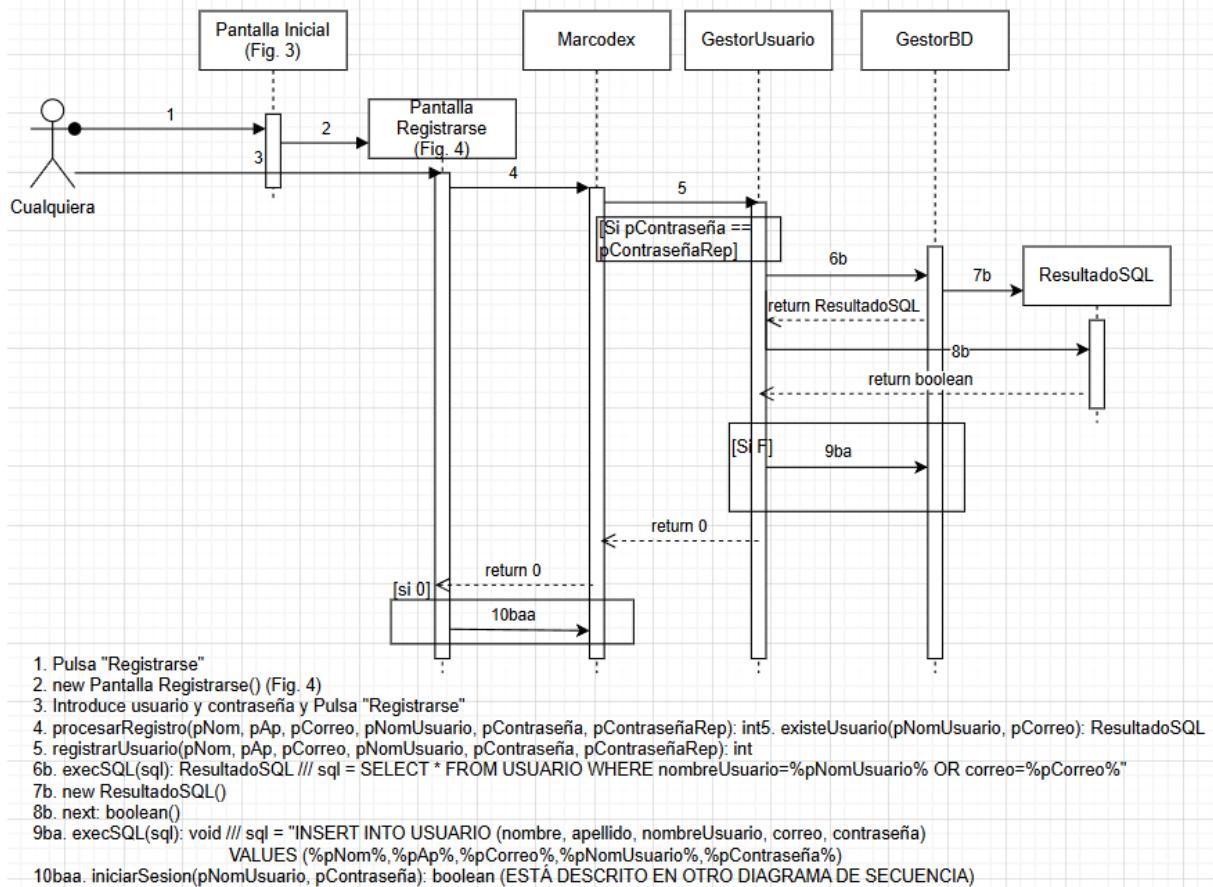
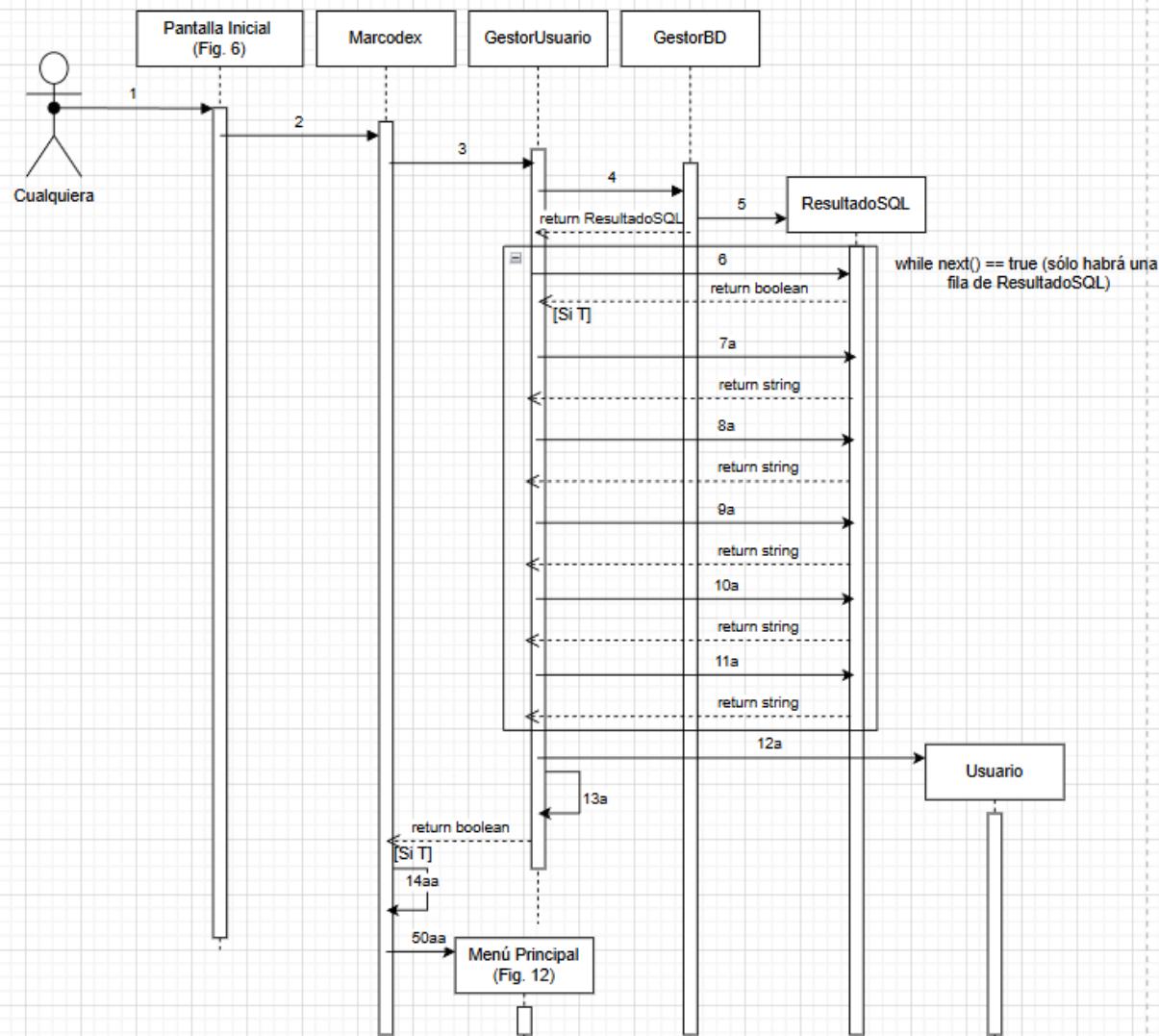
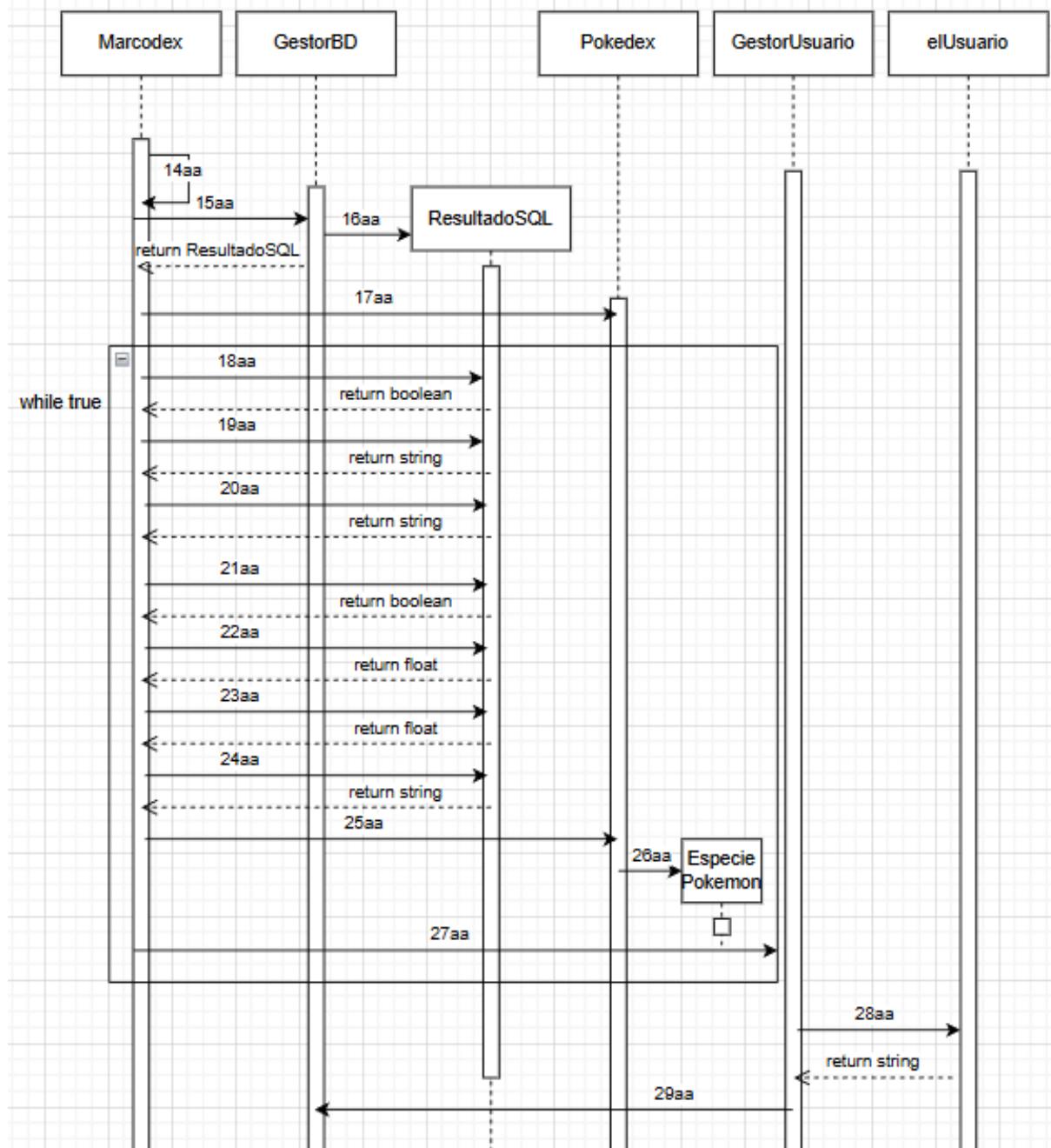


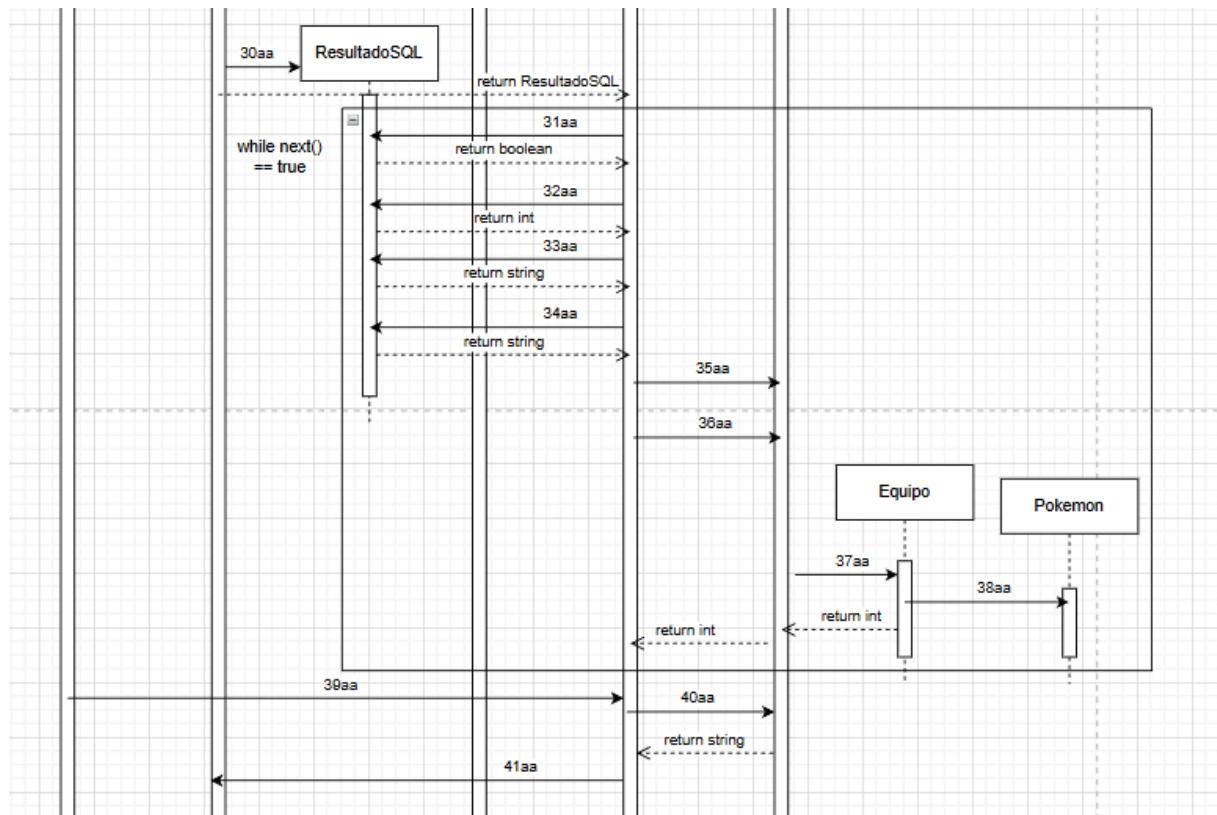
Figure 70: Diagramas Registrarse

INICIO DE SESIÓN CORRECTO



PRECARGAR DATOS (YA TENEMOS EL USUARIO y LOS POKEMONS EN LA BD), punto 1.2.1 del diagrama de comunicación





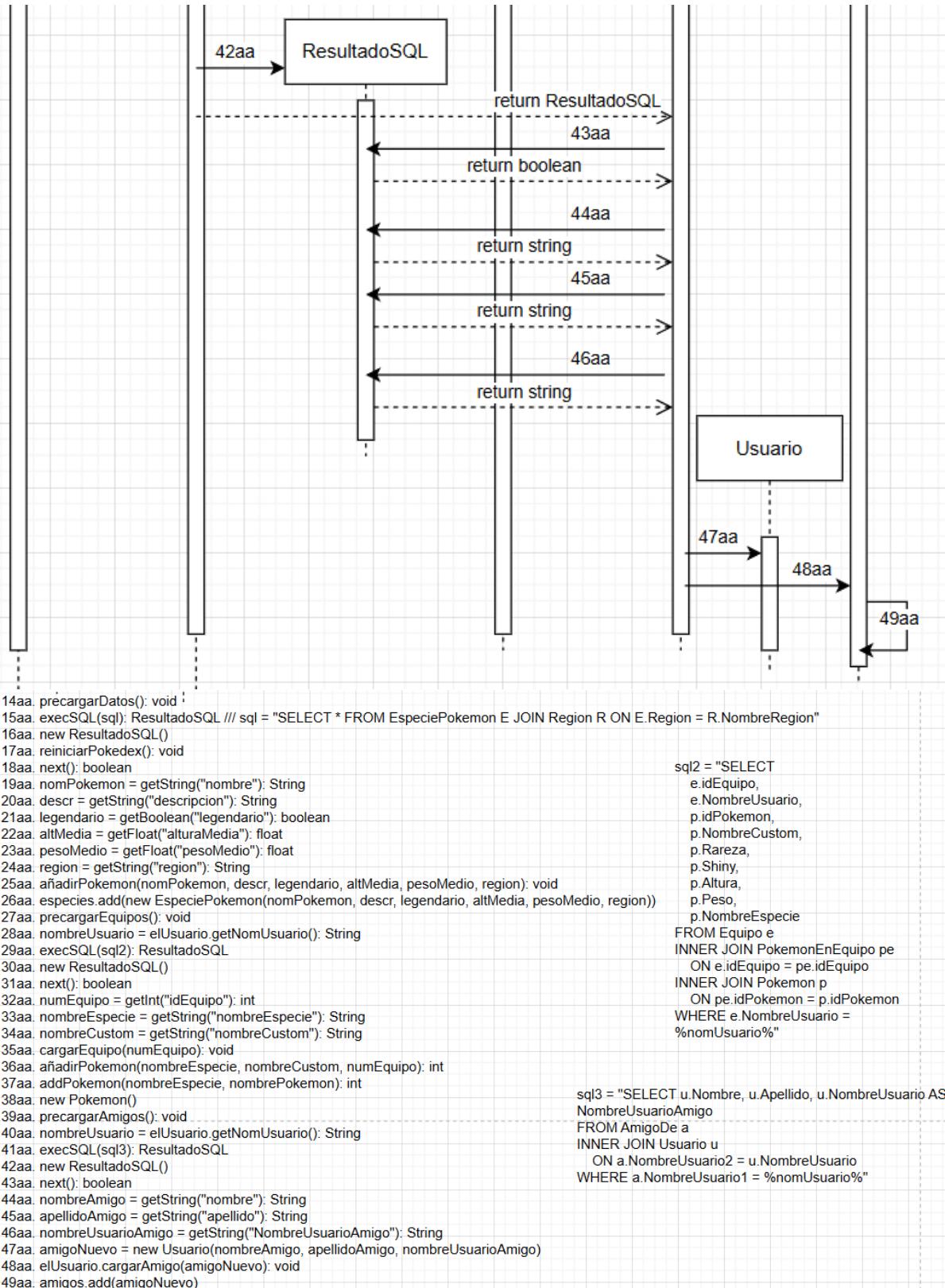
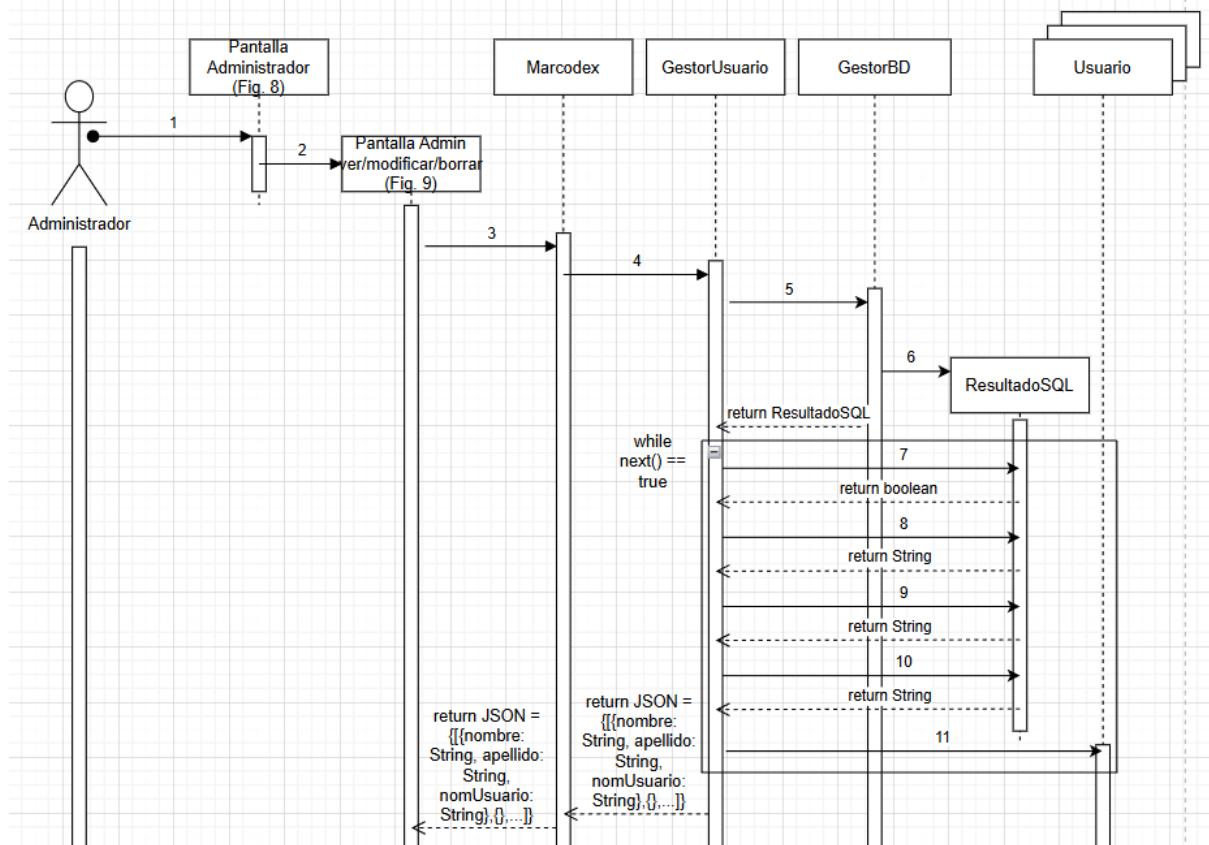
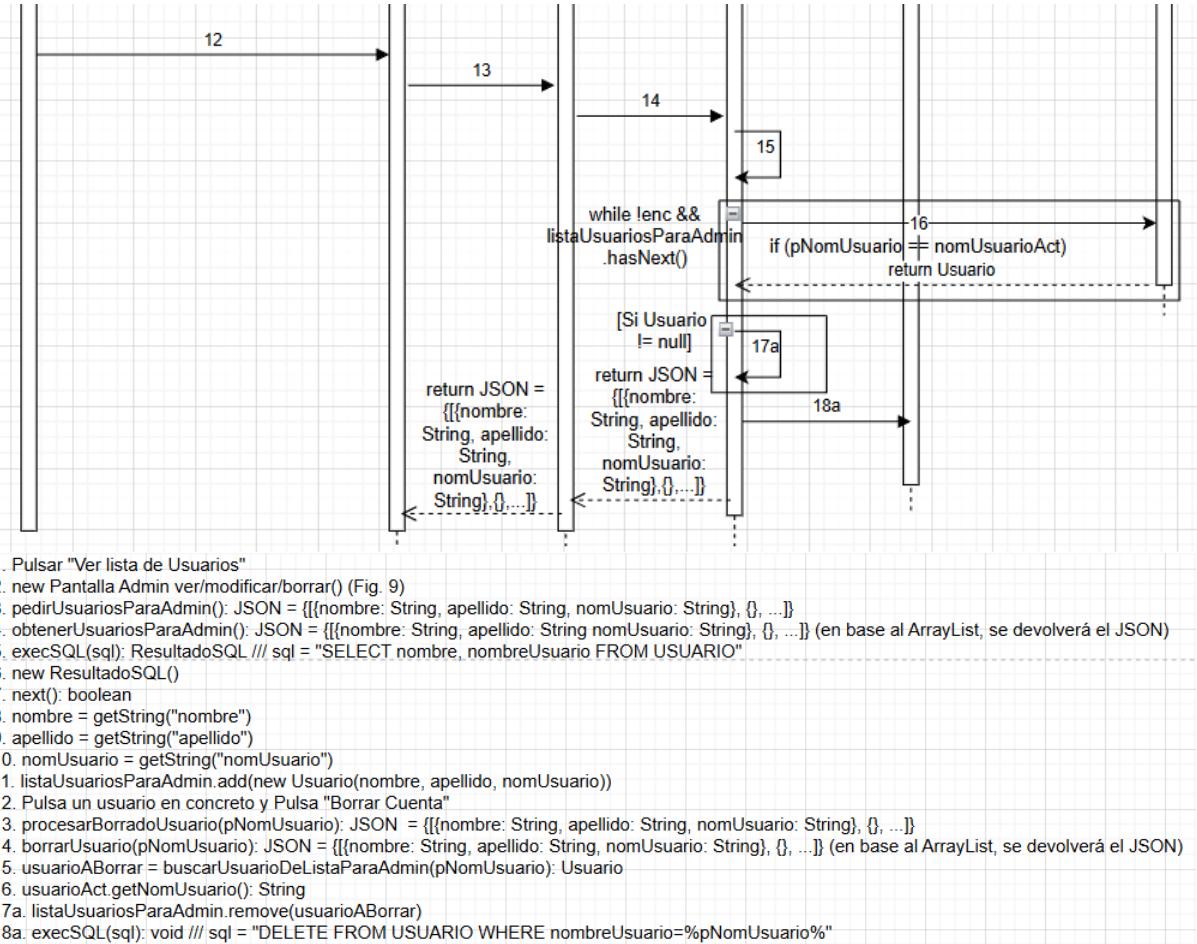


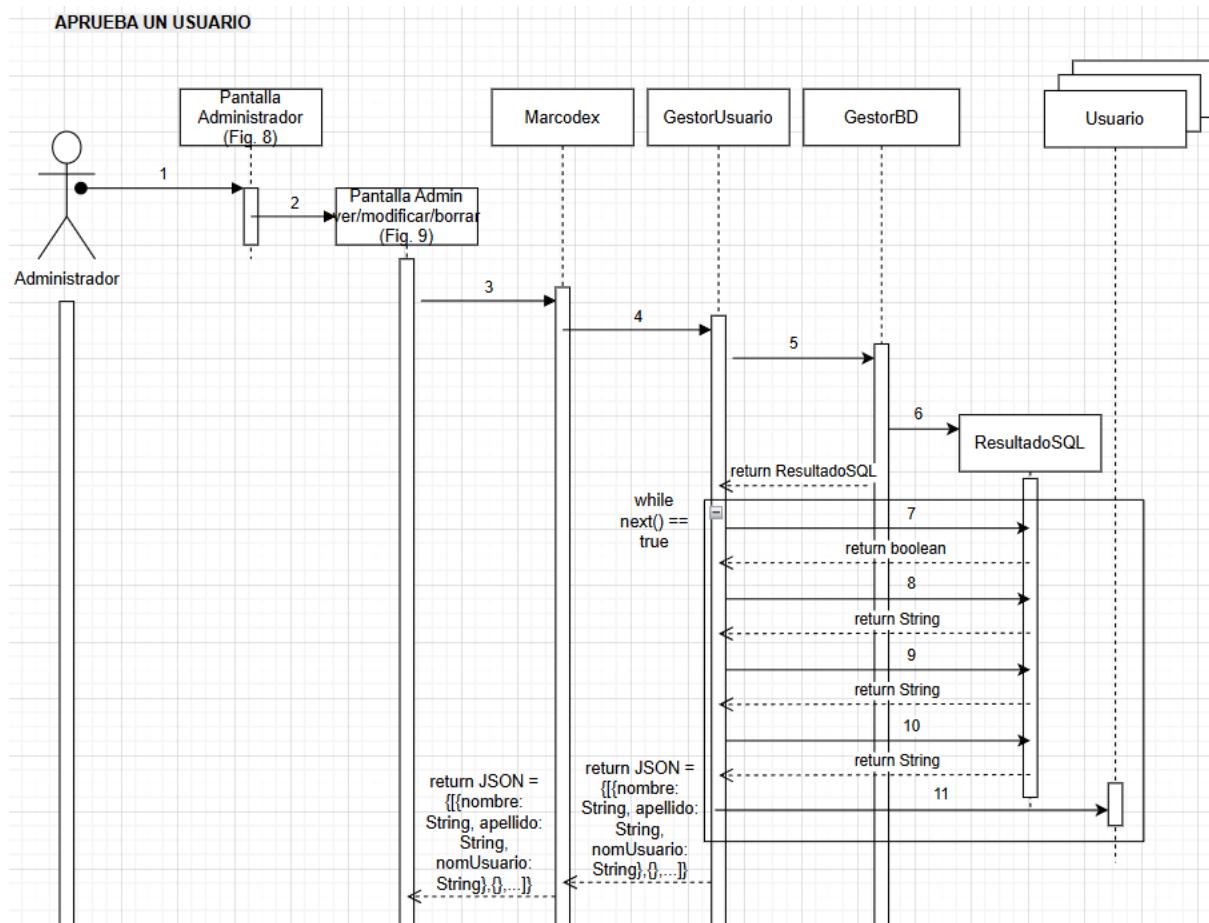
Figure 71: Diagramas Iniciar Sesión

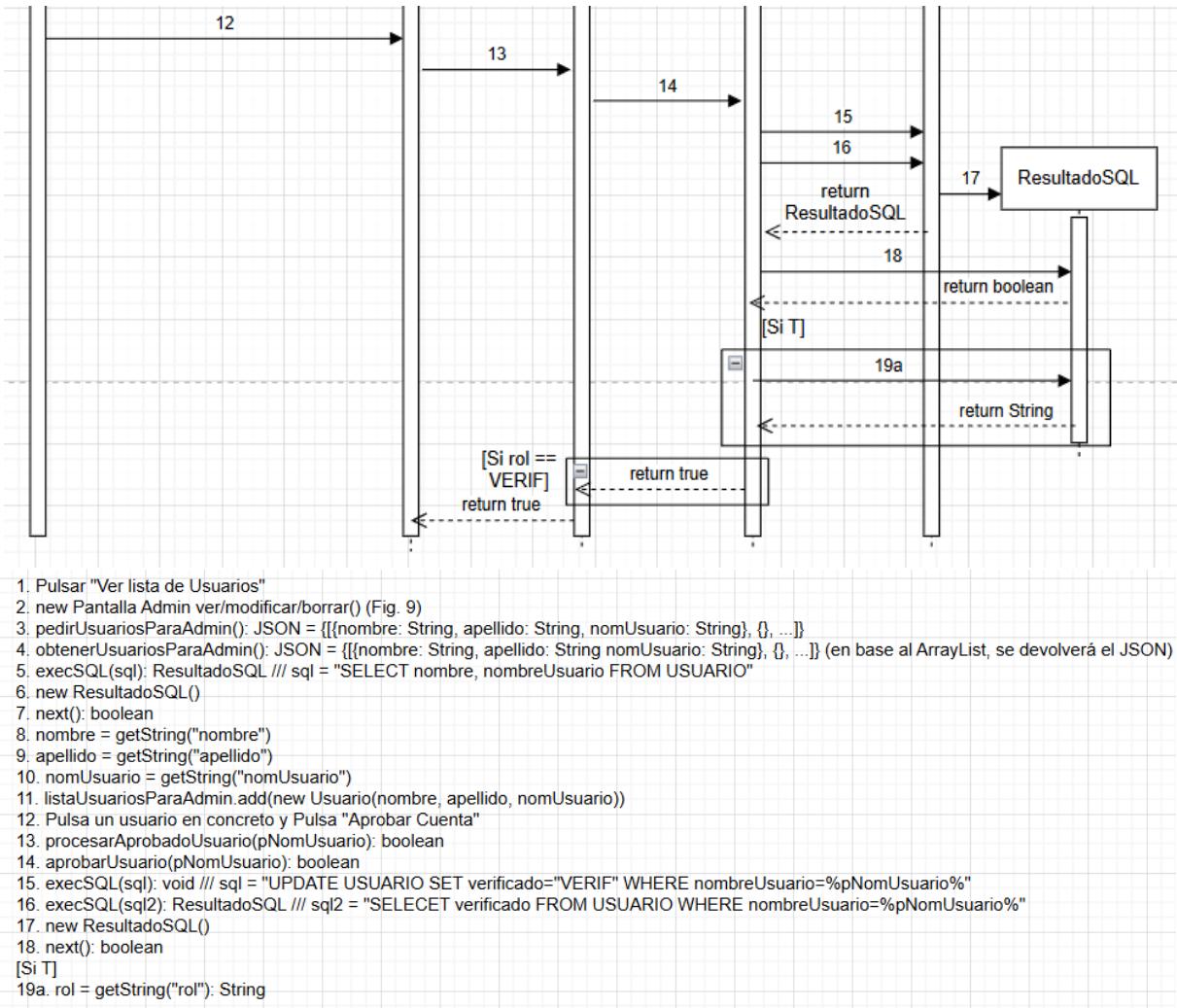
BORRA UN USUARIO



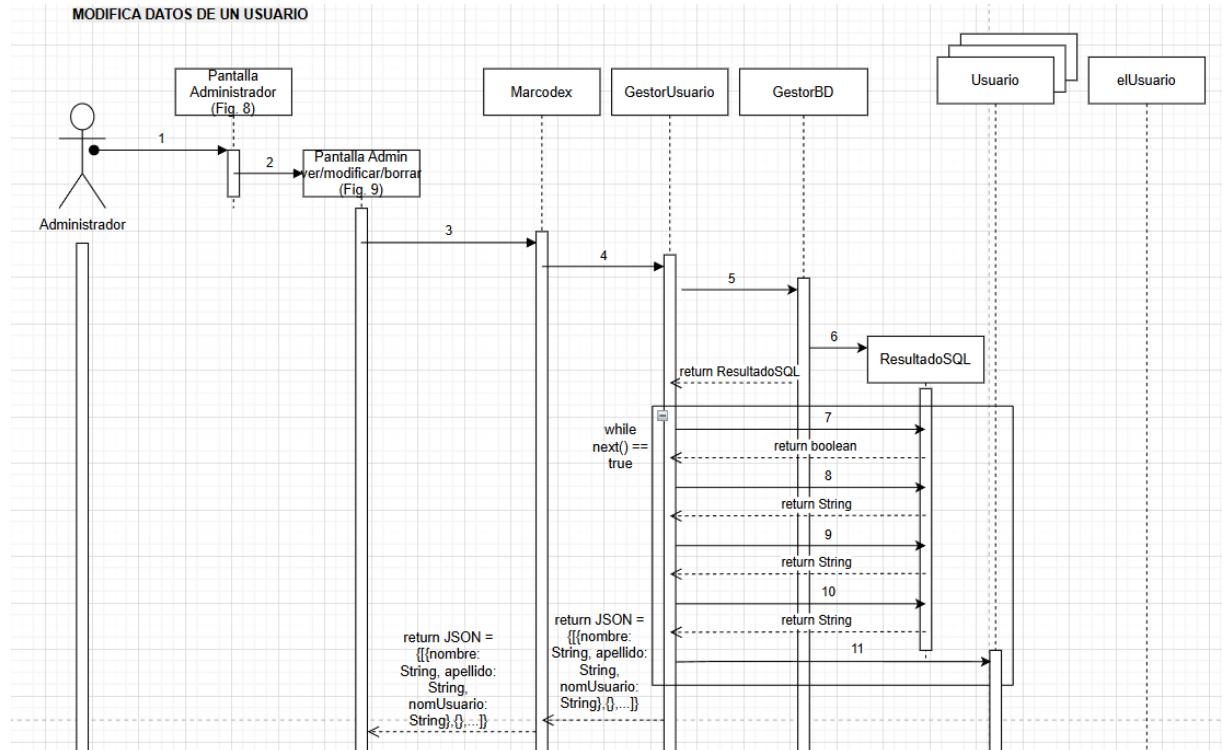


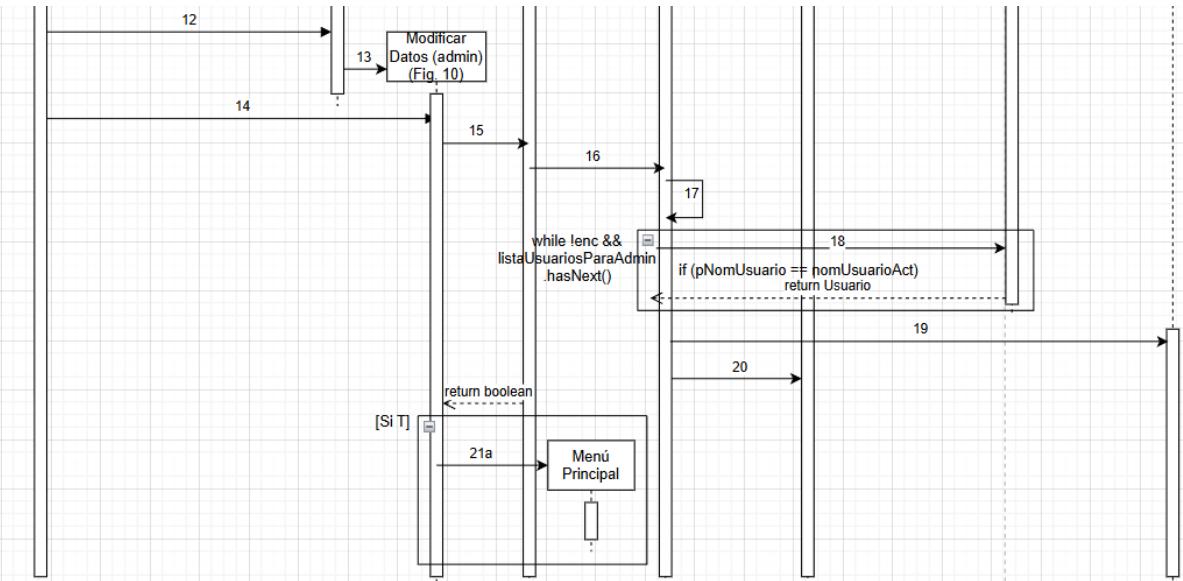
APRUEBA UN USUARIO





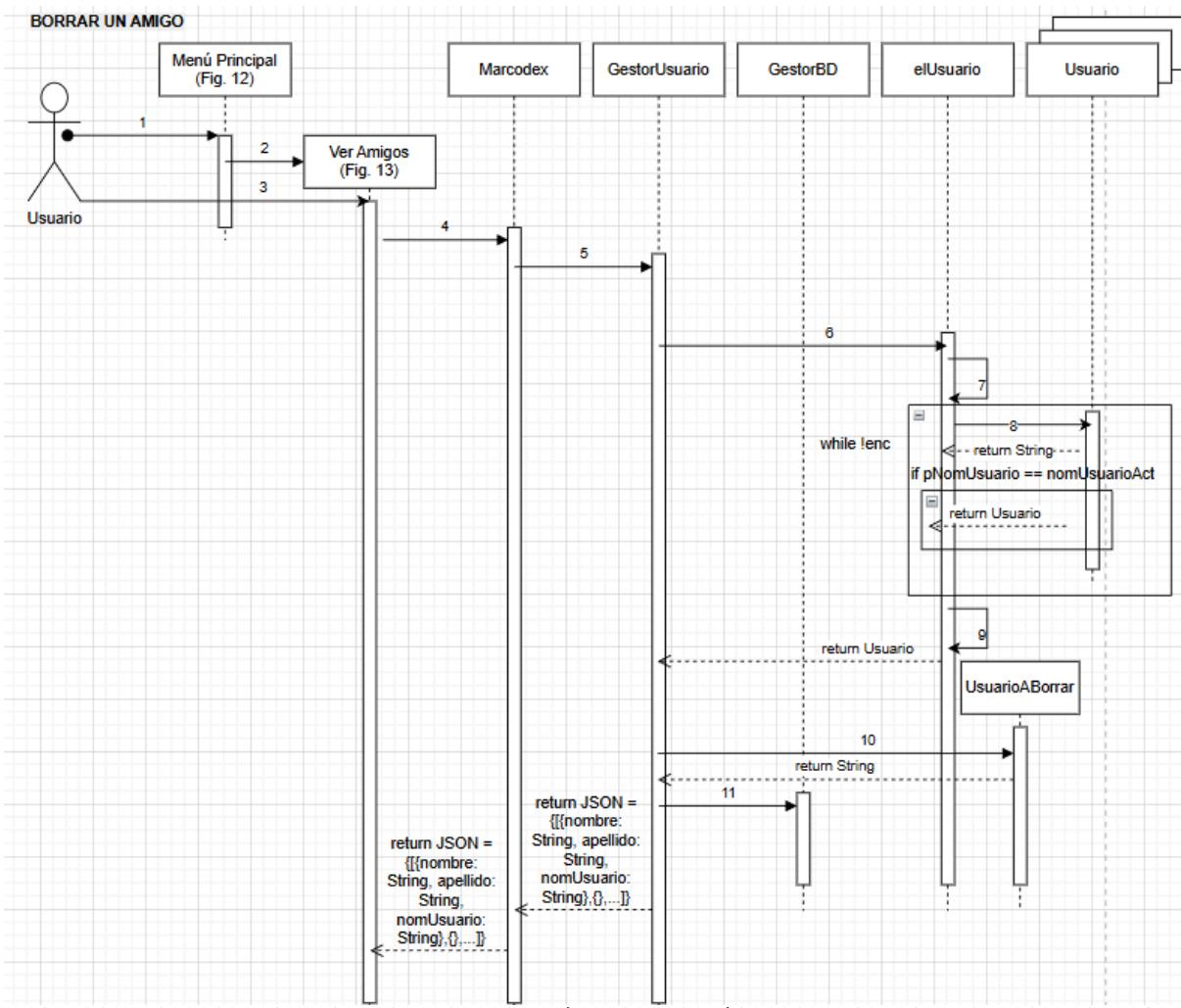
MODIFICA DATOS DE UN USUARIO



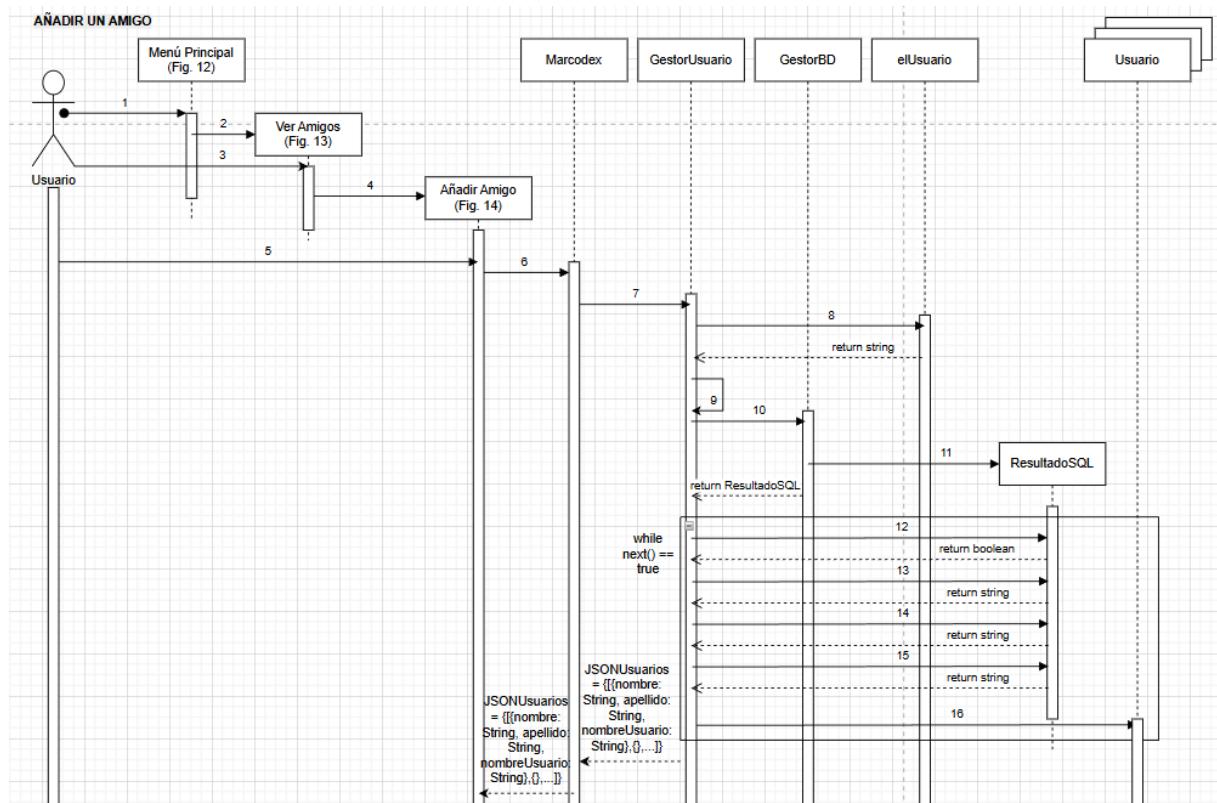


1. Pulsar "Ver lista de Usuarios"
 2. new Pantalla Admin ver/modificar/borrar() (Fig. 9)
 3. pedirUsuariosParaAdmin(): JSON = [[[nombre: String, apellido: String, nomUsuario: String}, {}, ...]]
 4. obtenerUsuariosParaAdmin(): JSON = [[[nombre: String, apellido: String nomUsuario: String}, {}, ...]] (en base al ArrayList, se devolverá el JSON)
 5. execSQL(sql): ResultadoSQL /// sql = "SELECT nombre, nombreUsuario FROM USUARIO"
 6. new ResultadoSQL()
 7. next(): boolean
 8. nombre = getString("nombre")
 9. apellido = getString("apellido")
 10. nomUsuario = getString("nomUsuario")
 11. listaUsuariosParaAdmin.add(new Usuario(nombre, apellido, nomUsuario))
 12. Pulsar un usuario en concreto y Pulsar "Modificar Datos"
 13. new Modificar Datos Pantalla Admin (pNomUsuario) (Fig. 10)
 14. Introduce datos y Pulsar "Guardar"
 15. procesarModificarDatosAdmin(pNomUsuario, pNombre, pAp, pNomUsuarioModif): boolean
 16. modificarUsuarioEnMemoriaPorAdmin(pNomUsuario, pNombre, pAp, pNomUsuarioModif): void
 17. usuarioAModificar = buscarUsuarioDeListaParaAdmin(pNomUsuario): Usuario
 18. usuarioAct.getNomUsuario(): String
 19. modificarDatos(pNom, pAp, pCorreo, pNomUsuarioModif, pContraNueva = null): void (método reutilizado)
 20. execSQL(sql): void /// sql = "UPDATE USUARIO SET nombreUsuario=%pNomUsuarioModif%, nombre=%pNombre%, apellido=%pAp% WHERE nombreUsuario=%pNomUsuario%" (atributos del usuarioAModificar)
- [Si T]
21. new Pantalla Administrador() (Fig. 8)

Figure 72: Diagramas Ver Lista usuarios



1. Pulsar "Mis Amigos"
2. new Ver Amigos() (Fig. 13)
3. Selecciona un usuario de la lista de amigos y Pulta "X"
4. procesarBorradoAmigo(pNomUsuarioAmigo): JSON = {[{nombre: String, apellido: String, nomUsuario: String}, ...]}
5. borrarAmigo(pNomUsuarioAmigo): JSON = {[{nombre: String, apellido: String, nomUsuario: String}, ...]}
6. borrarAmigo(pNomUsuarioAmigo): Usuario
7. amigoABorrar = buscarAmigo(pNomUsuarioAmigo): Usuario
8. nomUsuarioAct = usuarioAct.getNomUsuario(): String
9. amigos.remove(amigoABorrar)
10. nomUsuario = getNomUsuario(): String
11. execSQL(sql): void // sql ="DELETE FROM AMIGODE WHERE NombreUsuario1=%nomUsuario% AND NombreUsuario2=%pNomUsuarioAmigo%"



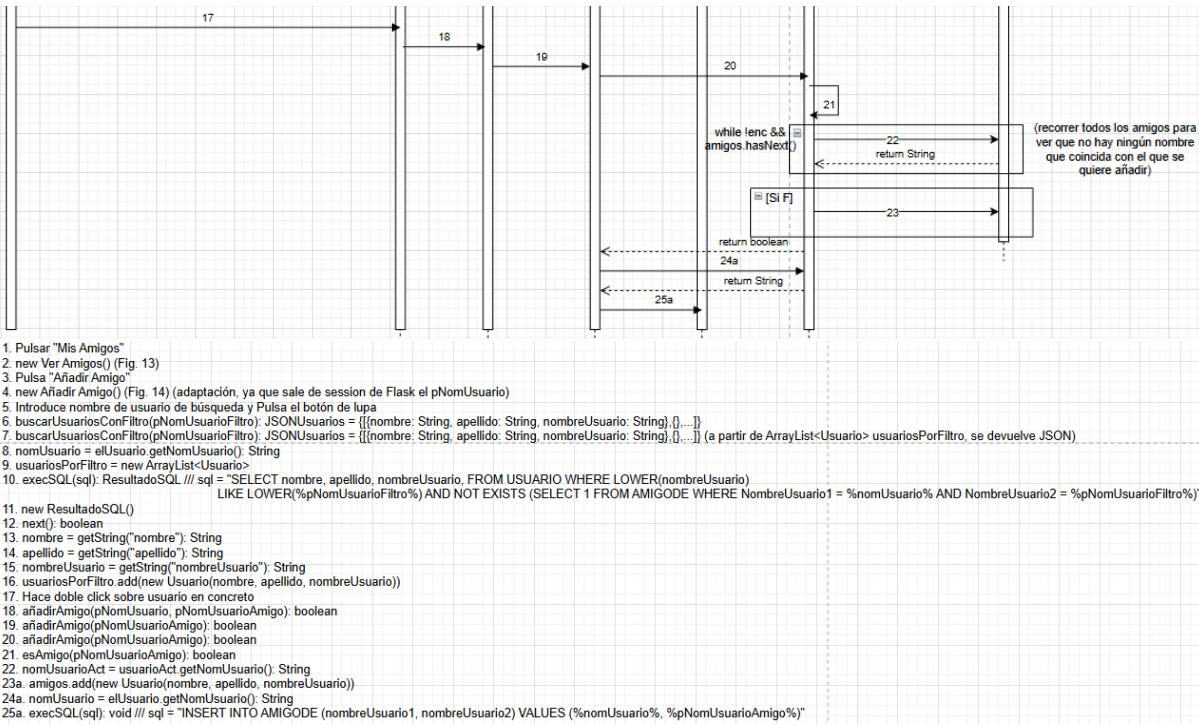
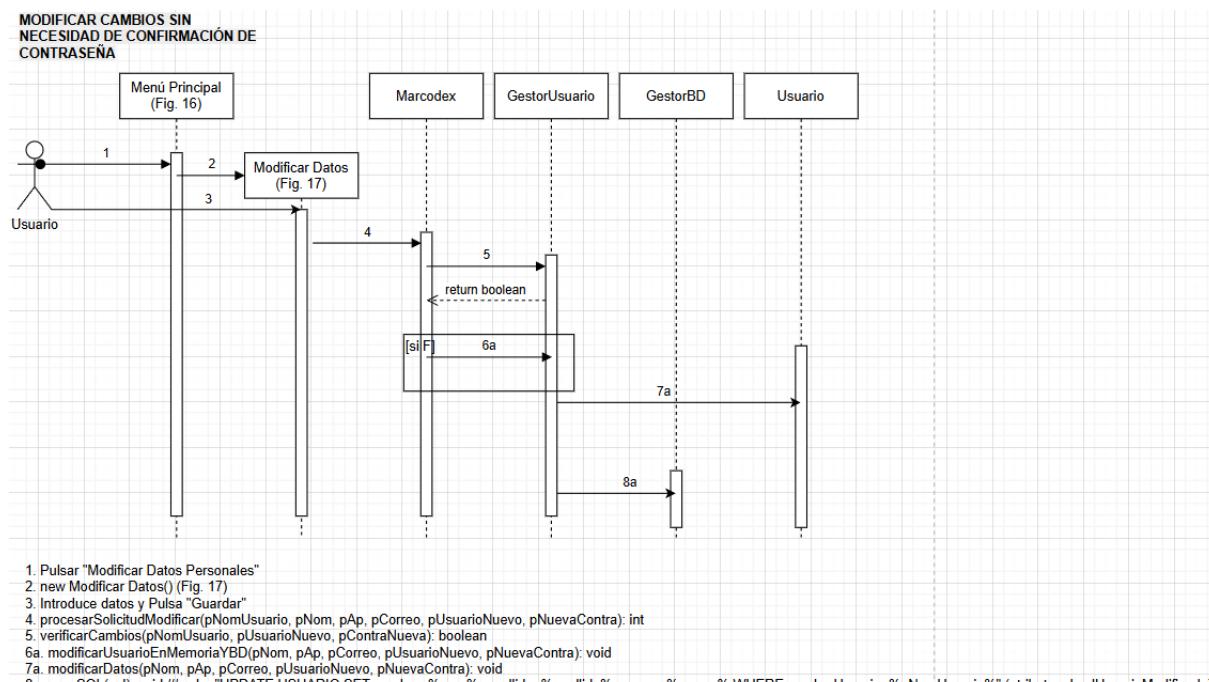
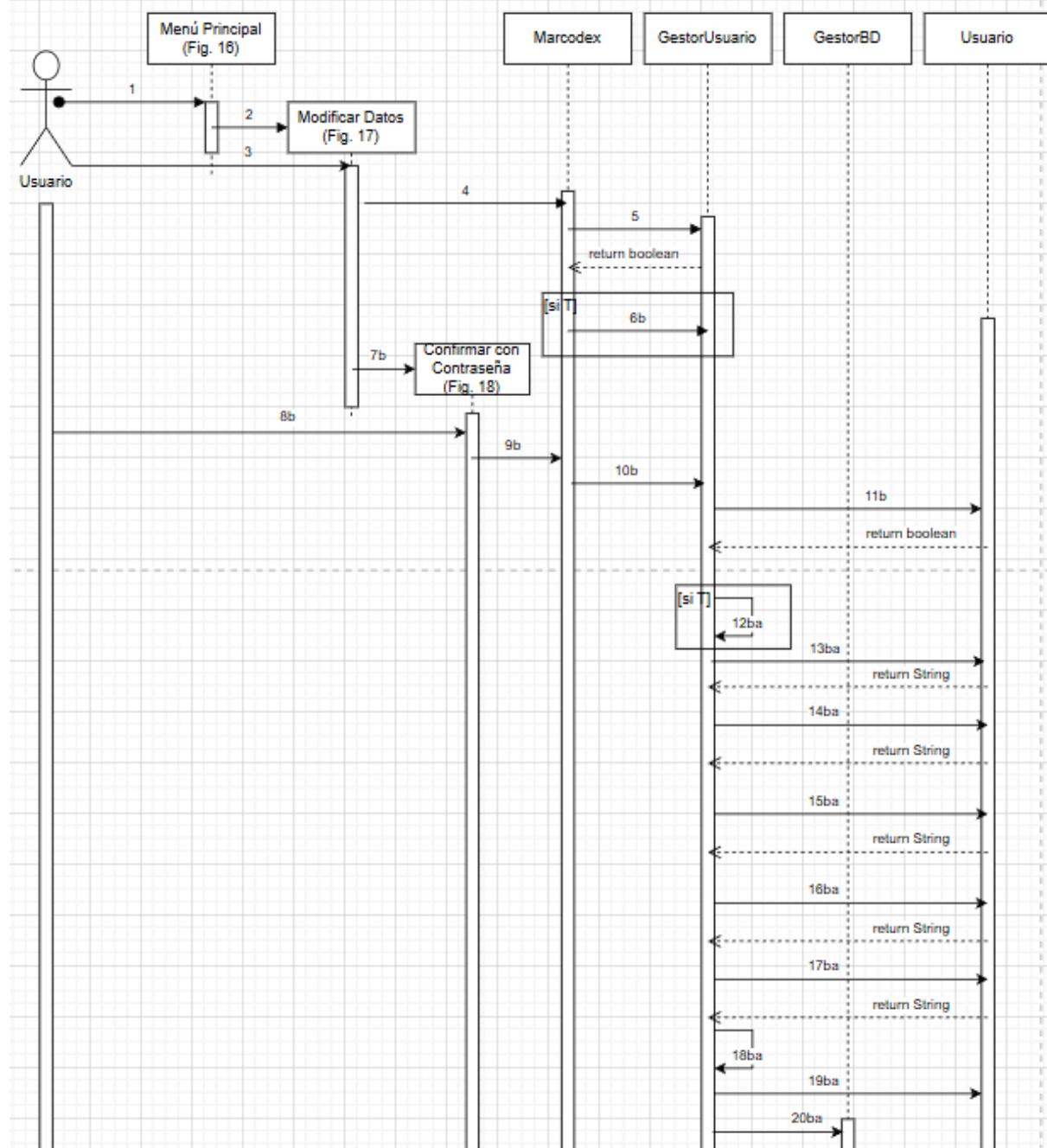


Figure 73: Diagramas Ver Amigos

**MODIFICAR CAMBIOS SIN
NECESIDAD DE CONFIRMACIÓN DE
CONTRASEÑA**



MODIFICAR CAMBIOS CON NECESIDAD DE CONFIRMACIÓN DE CONTRASEÑA



```

1. Pulsar "Modificar Datos Personales"
2. new Modificar Datos() (Fig. 17)
3. Introduce datos y Pulsa "Guardar"
4. procesarSolicitudModificar(pNomUsuario, pNom, pAp, pCorreo, pUsuarioNuevo, pNuevaContra): int
5. verificarCambios(pNomUsuario, pUsuarioNuevo, pContraNueva): boolean
6b. guardarModificaciónTemporal(pNomUsuario, pNom, pAp, pCorreo, pUsuarioNuevo, pNuevaContra): void
7b. new Confirmar con Contraseña(pNomUsuario) (Fig. 18)
8b. Introduce contraseña y Pulsa "Confirmar"
9b. confirmarConContraseña(pNomUsuario, pContraseña): String
10b. validarCredencialesYGuardarCambios(pNomUsuario, pContraseña): String
11b. validarPassword(pContraseña): boolean
12ba. datosUsuarioModif = recuperarModificacionTemporal(pNomUsuario): Usuario
13ba. elNombre = getNombre(): String
14ba. elAp = getApellido(): String
15ba. elCorreo = getCorreo(): String
16ba. elNomUsuario = getNomUsuario(): String
17ba. laContra = getContraseña(): String
18ba. modificarUsuarioEnMemoriaYBD(elNombre, elAp, elCorreo, elNomUsuario, laContra): void
19ba. modificarDatos(elNombre, elAp, elCorreo, elNomUsuario, laContra): void
20ba. execSQL(sql):void sql = "UPDATE USUARIO SET nombre=%nombre%, apellido=%apellido%, correo=%correo%,
    nombreUsuario=%nombreUsuarioModif%, contraseña=%contraseña% WHERE nombreUsuario=%pNomUsuario%"
```

Figure 74: Diagramas Modificar Datos Personales

9.2 Crear equipos Pokemon (Tabata)

Los números y por tanto el flujo es el mismo que el de los diagramas de comunicación

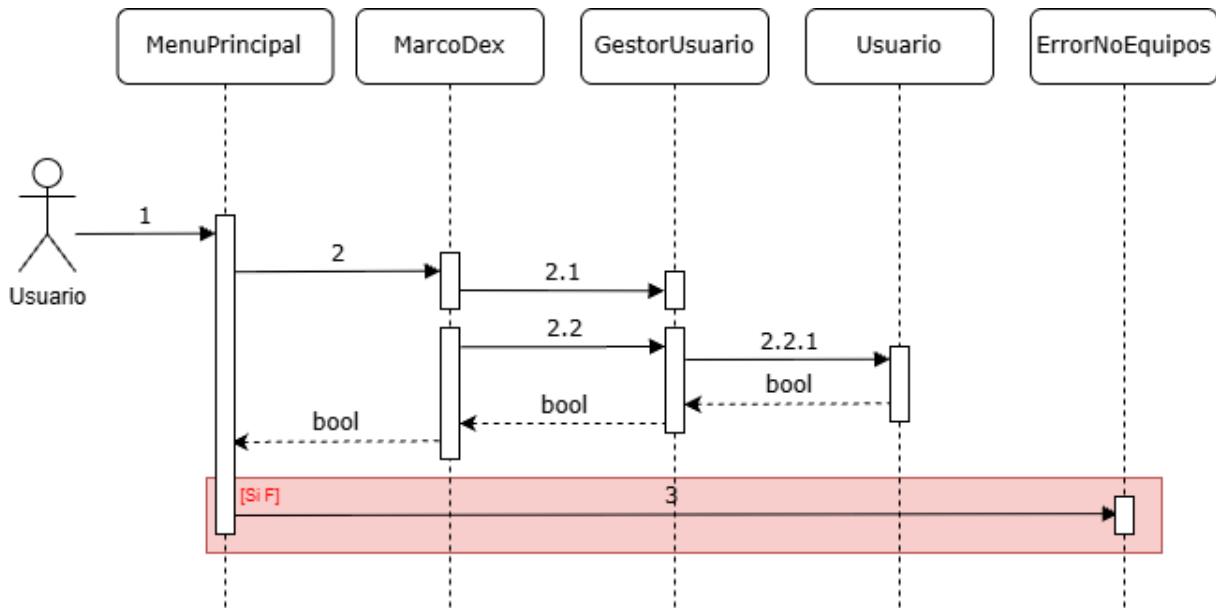


Figure 75: Ver Equipos - Pulsar Mis Equipos

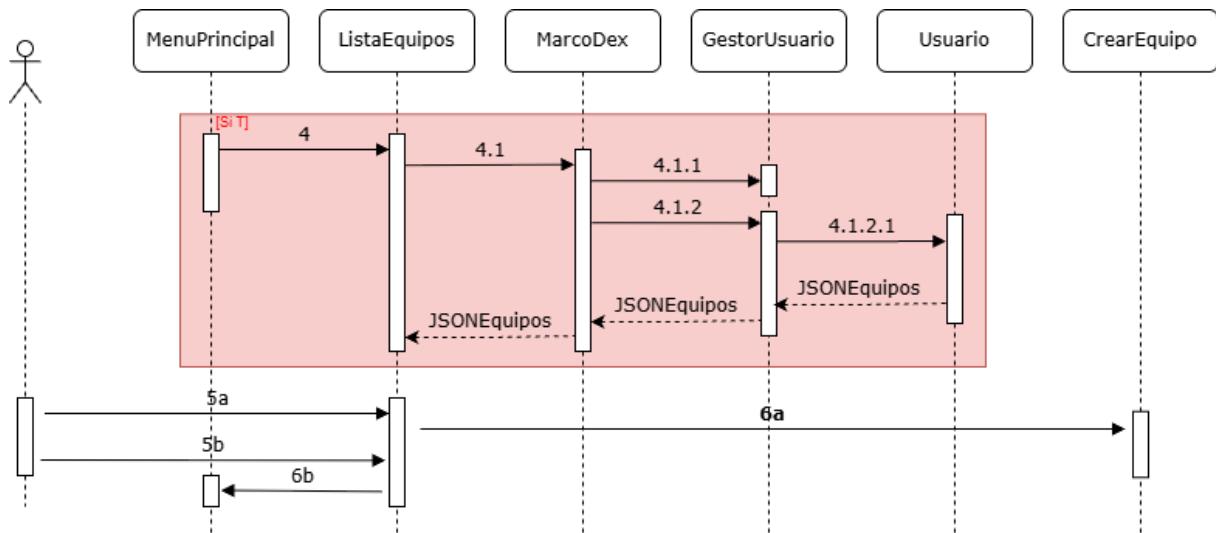


Figure 76: Ver Equipos - Mis Equipos lista

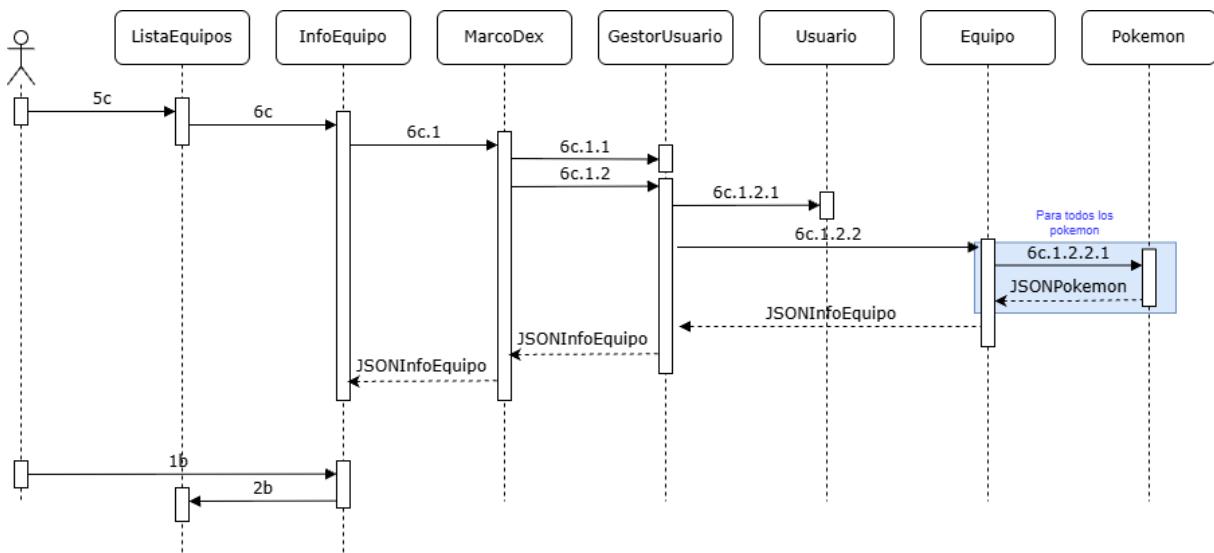


Figure 77: Ver Equipos - Pulsa Equipo

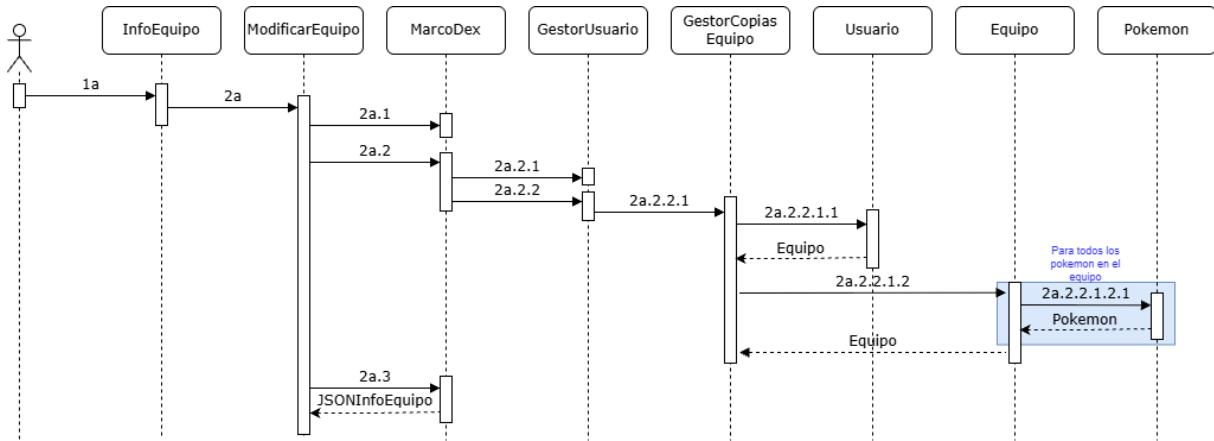


Figure 78: Ver Equipos - Pulsa Modificar

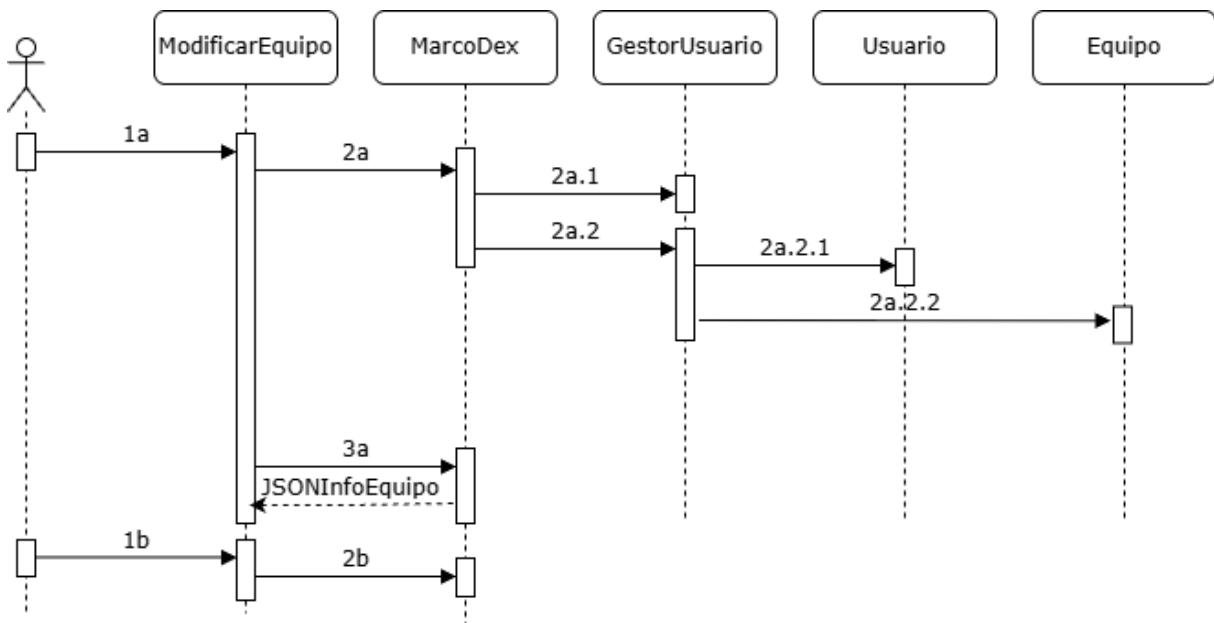


Figure 79: Modificar - Añadir/Borrar

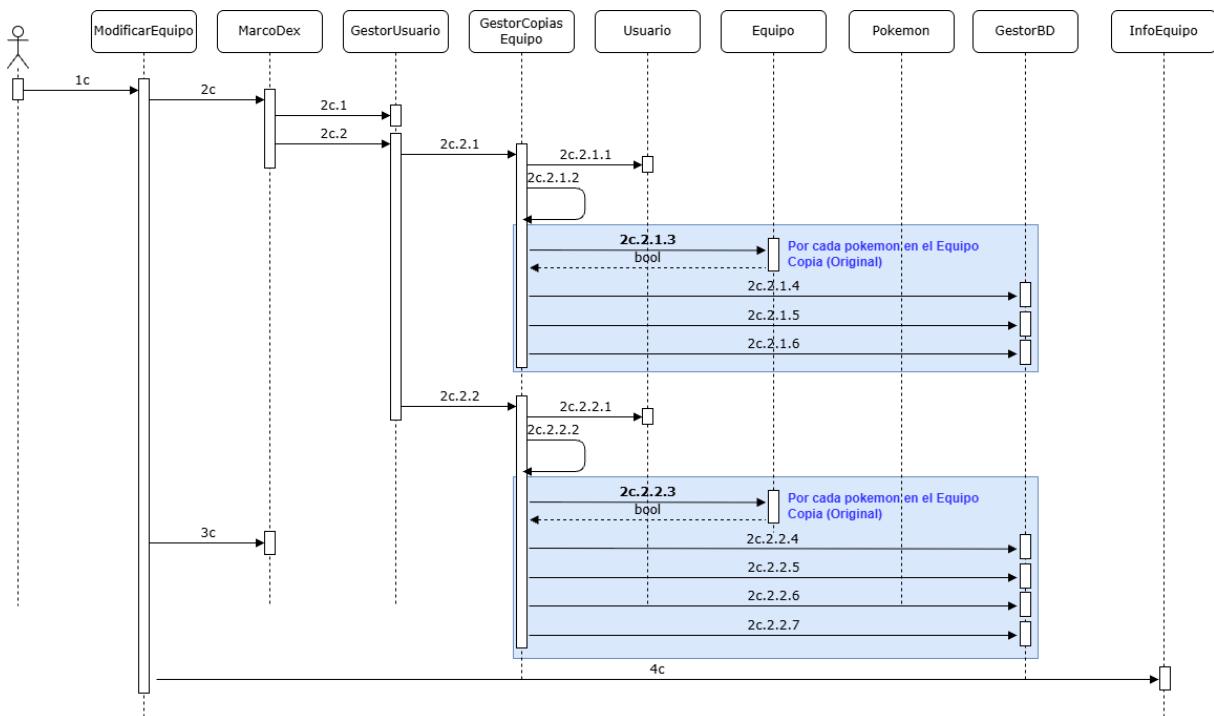


Figure 80: Modificar - Guardar

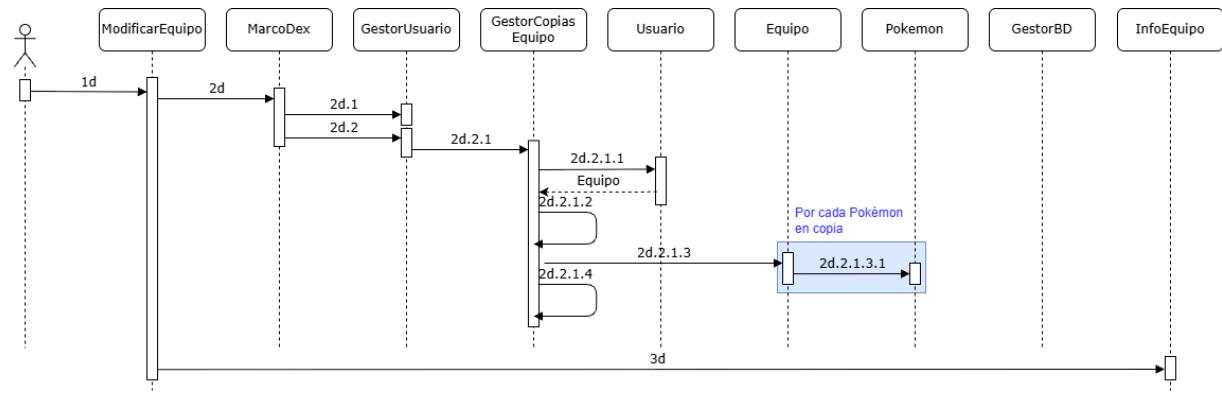


Figure 81: Modificar - Cancelar

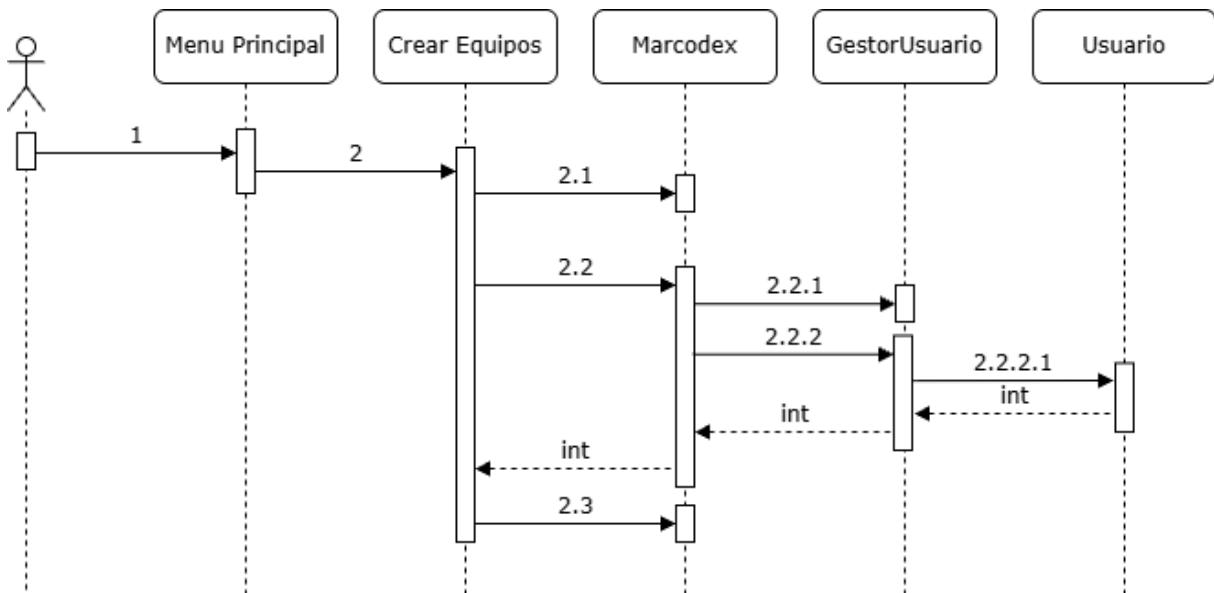


Figure 82: Crear Equipo - Pulsar Crear Equipo

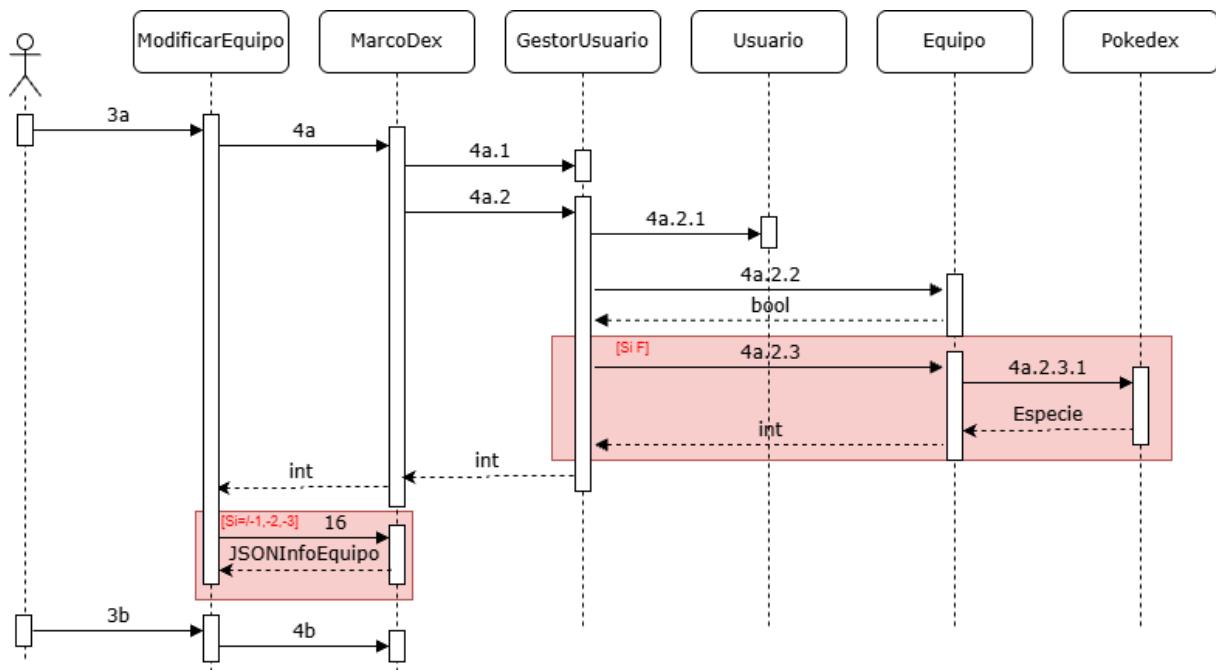


Figure 83: Crear Equipo - Añadir/Borrar

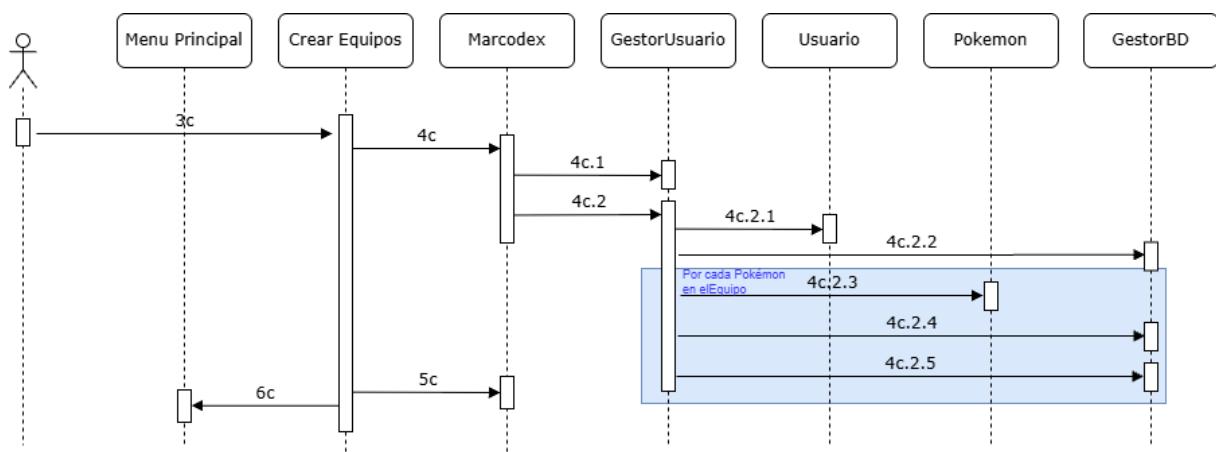


Figure 84: Crear Equipo - Guardar

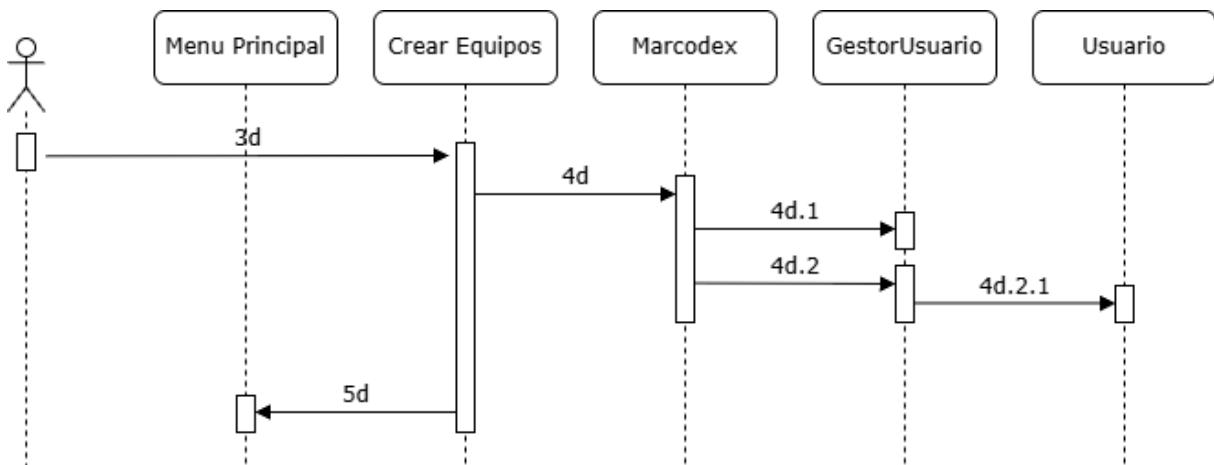
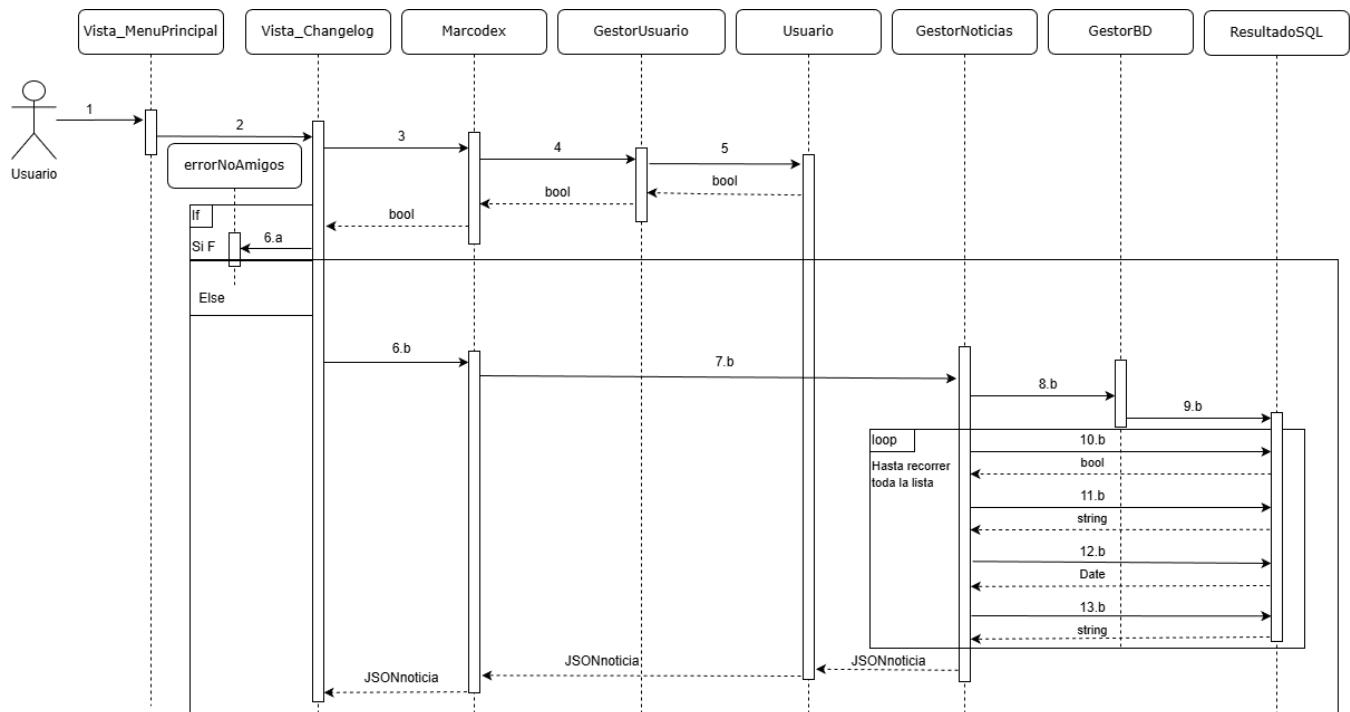


Figure 85: Crear Equipo - Cancelar

9.3 ChangeLog (Gorka)



1. El usuario pulsa el botón "ChangeLog" en el menú principal
2. new Vista_Changelog()
3. tieneAmigos(NombreUsuario: string):bool
4. tieneAmigos(NombreUsuario: string):bool
5. tieneAmigos(NombreUsuario: string):bool
- [Si F]
 - 6.a. new errorNoAmigos()
- [Else]
 - 6.b. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia
 - 7.b. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia
 - 8.b. execSQL(sqlNoticia): ResultadoSQL
 - 9.b. new ResultadoSQL()
 - 10.b. next():bool
 - 11.b. getString("NombreUsuario"): string
 - 12.b. getDate("FechaHora"): Date
 - 13.b. getString("Contenido"): Date

Figure 86: mostrar Changelog

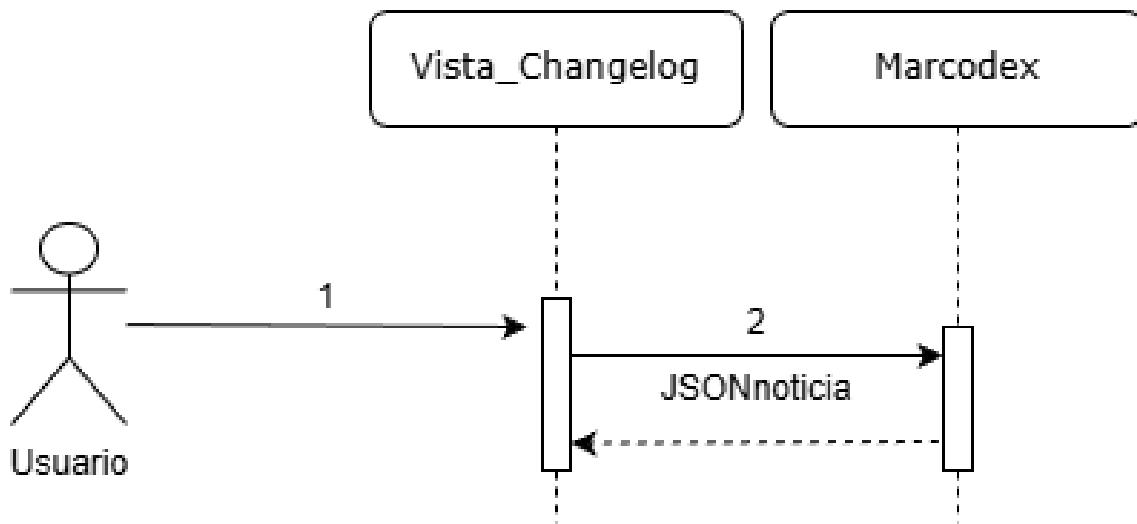
```

sqlNoticia:    SELECT NombreUsuario, FechaHora, Contenido
                FROM Publica
                WHERE FechaHora >= DATE_SUB(CURDATE(), INTERVAL 1
                                              MONTH)
                AND NombreUsuario IN (
                    SELECT nombreUsuario1
                    FROM AmigoDe
                    WHERE nombreUsuario2 = %nombreUsuario%
                )
                AND NombreUsuario LIKE % filtro% %
                ORDER BY FechaHora DESC;

```

JSONnoticia: { [{ nombreUsuario: string, fecha: date, contenido: string }] }

Figure 87: mostrar Changelog



1. El usuario pone algo en la barra de búsqueda y le da al botón de búsqueda
2. mostrarChangeLog(NombreUsuario: string, filtro: string):JSONnoticia

Figure 88: Filtrar Changelog

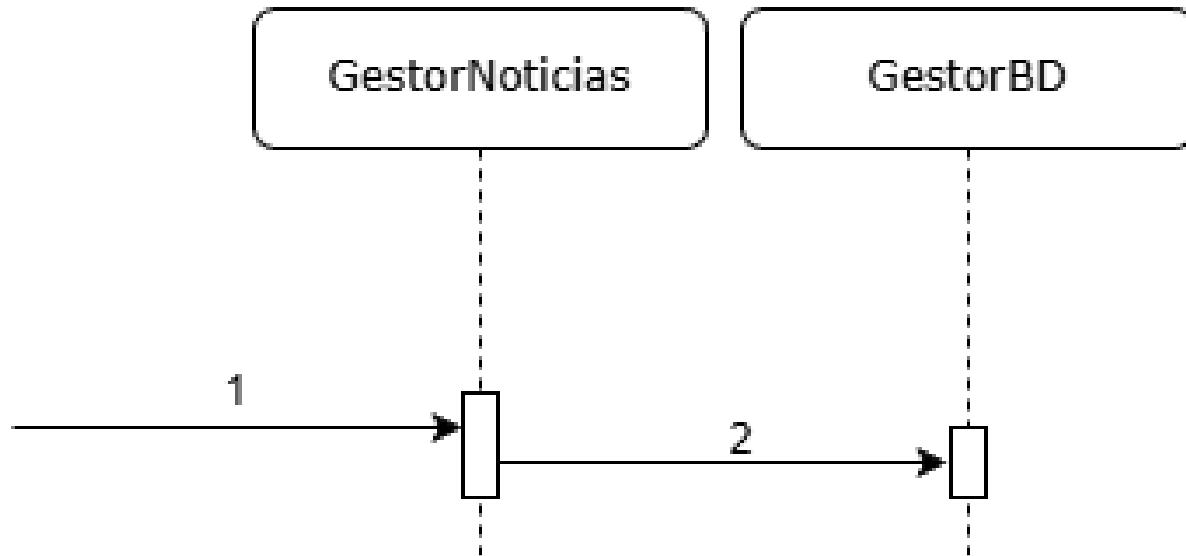


Figure 89: Añadir noticia

1. añadirNoticia(nombreUsuario, descripción)
2. ExecSQL(sqlNuevaNoticia): void

sqlNuevaNoticia: `INSERT INTO Publica (nombreUsuario, FechaHora, Contenido)
VALUES (%nombreUsuario%, NOW(), %descripcion%)`

9.4 Lista Pokémon y filtros (Laura)

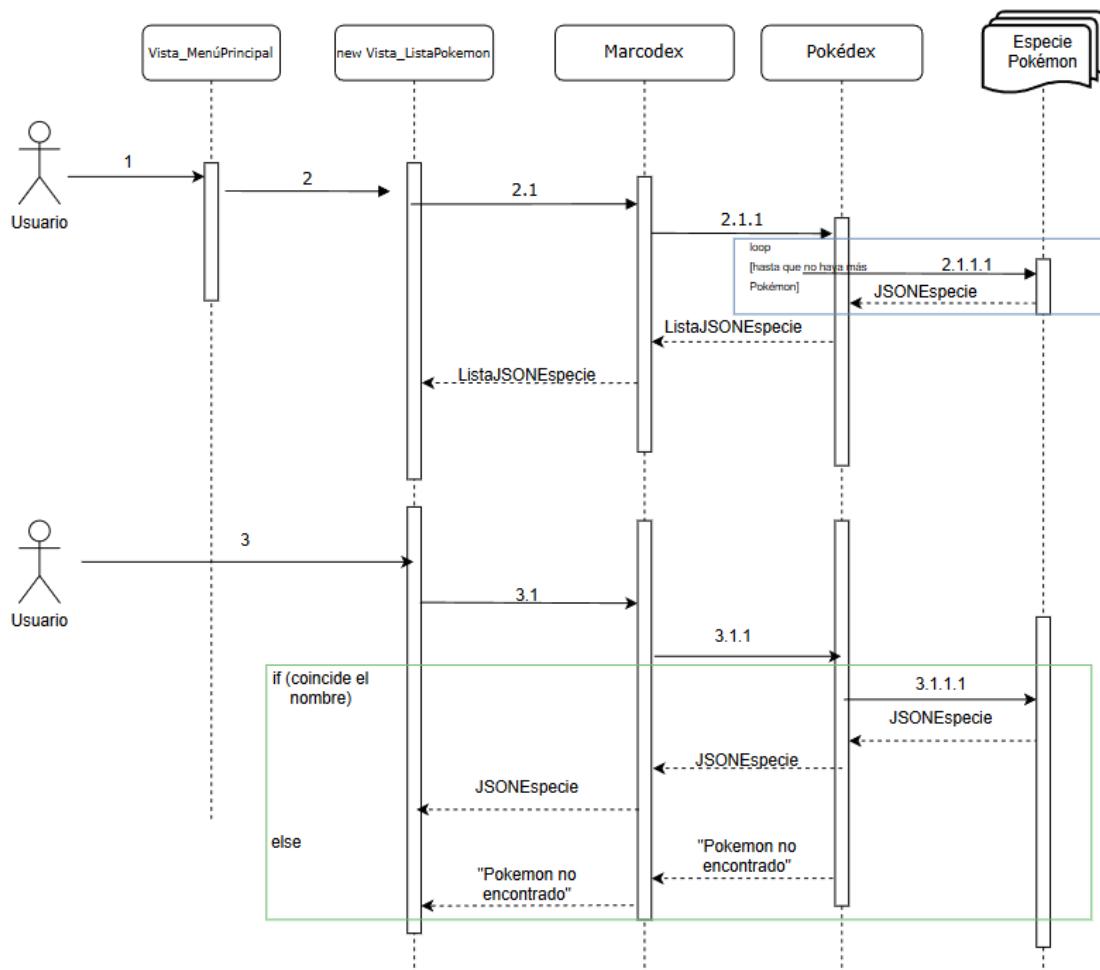


Figure 90: Ver Lista y detalles Pokémon

1. Usuario pulsa "Lista Pokémon" en la pantalla principal
2. new Lista_Pokémon()
 - 2.1 mostrarPokédex(): ListaJSONEspecie
 - 2.1.1 mostrarPokédex(): ListaJSONEspecie
 - [Bucle]
 - 2.1.1.1 getInfo(): JSONEspecie
3. Usuario selecciona un Pokémon de la Lista Pokémon
 - 3.1 mostrarInfo(nombreEspecie): JSONEspecie
 - 3.1.1 getInfo(nombreEspecie): JSONEspecie
 - IF [el nombre es igual]
 - T 3.1.1.1 getInfo(): JSONEspecie
 - F Devuelve "Pokemon no encontrado"

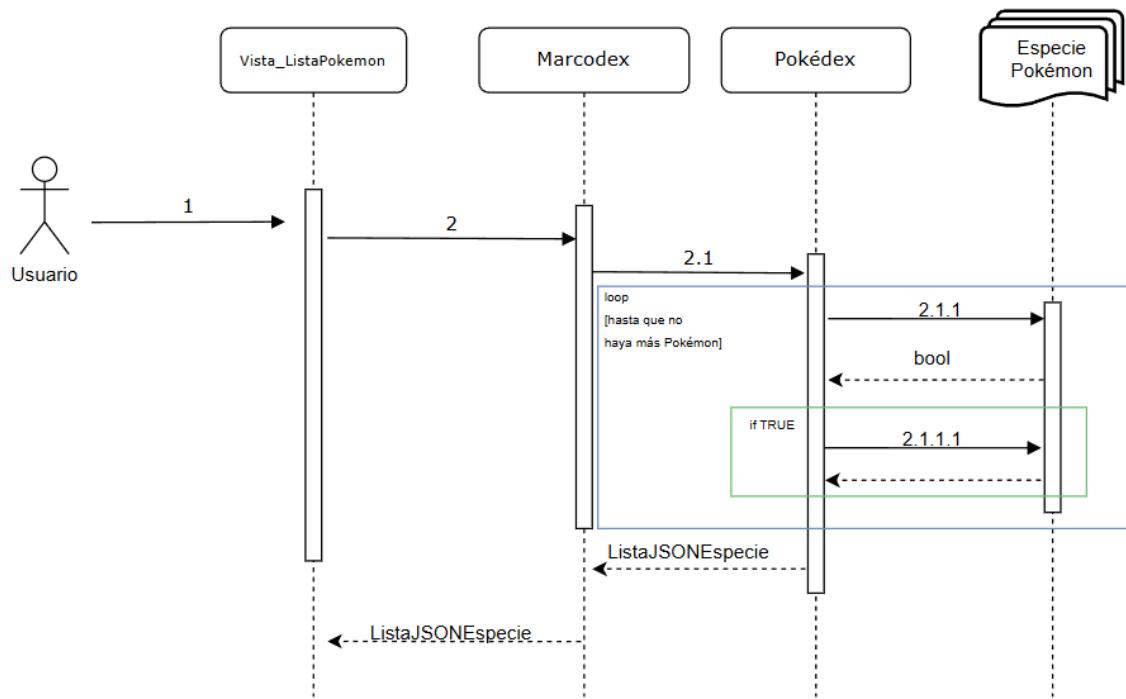


Figure 91: Filtrar Lista Pokémon

1. Usuario selecciona "Filtrar por nombre" / "Filtrar por tipo" en la pantalla de la Pokédex e introduce un valor de filtro (String)
2. aplicarFiltro(String filtro, String valor): ListaJSONEspecie
 - 2.1 filtrarPokedex(String filtro, String valor): ListaJSONEspecie
 - [Bucle para cada Especie Pokémon]
 - 2.1.1 comprobarFiltroValor(String filtro, String valor): bool
 - [Si True]
 - 2.1.1.1 añadir JSONEspecie a la lista filtrada

9.5 Chatbot (Iker)

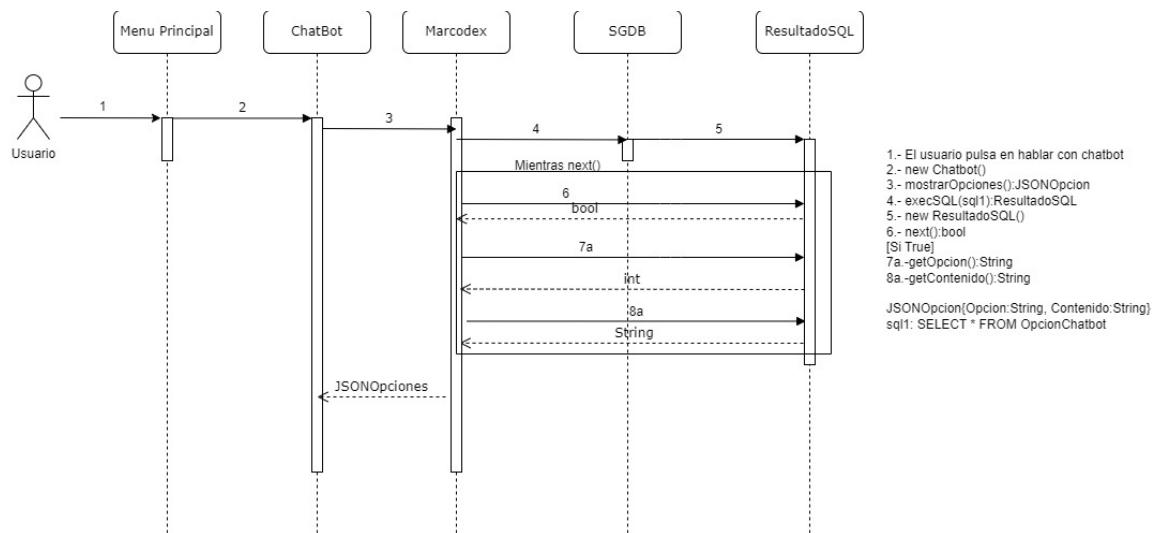


Figure 92: mostrar ChatBot

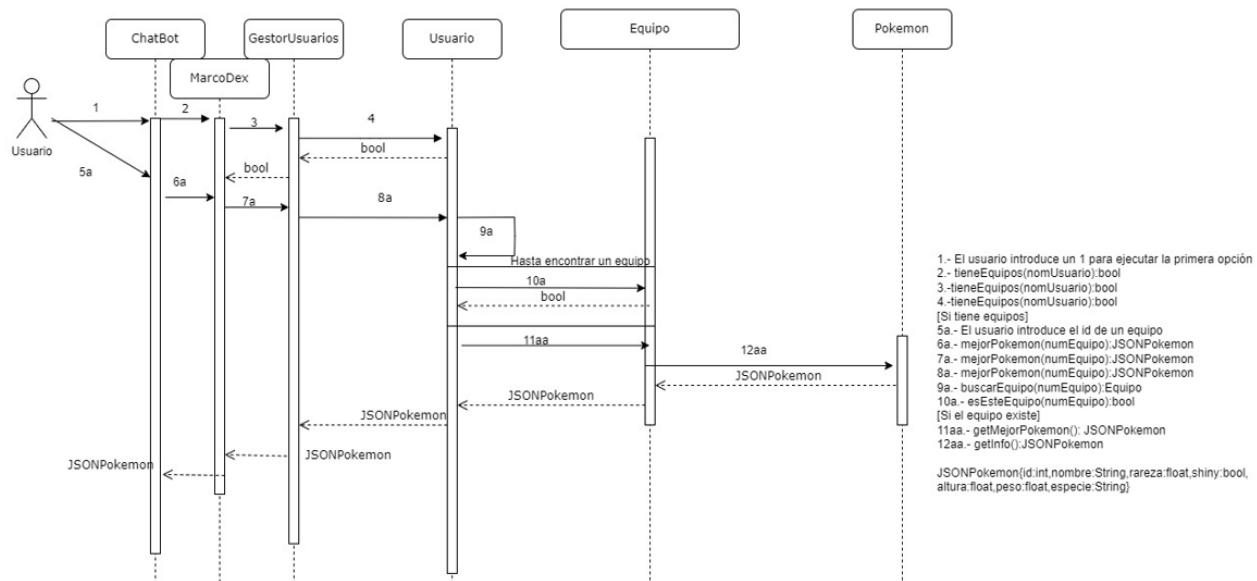


Figure 93: Opción 1

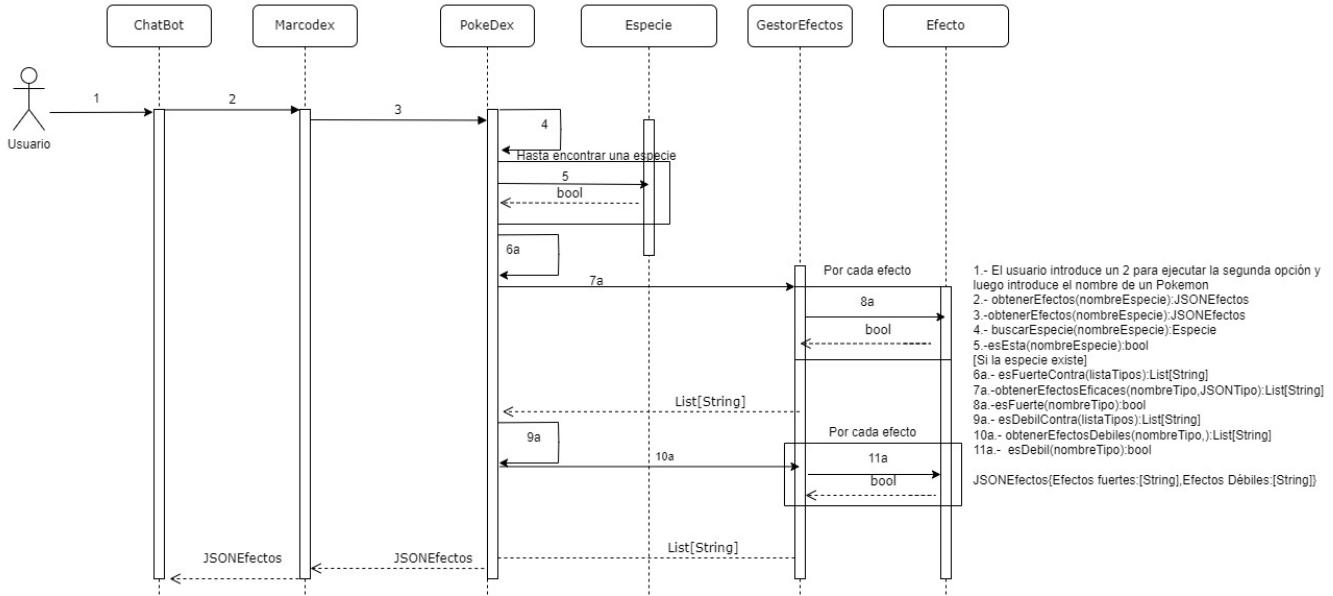


Figure 94: Opción 2

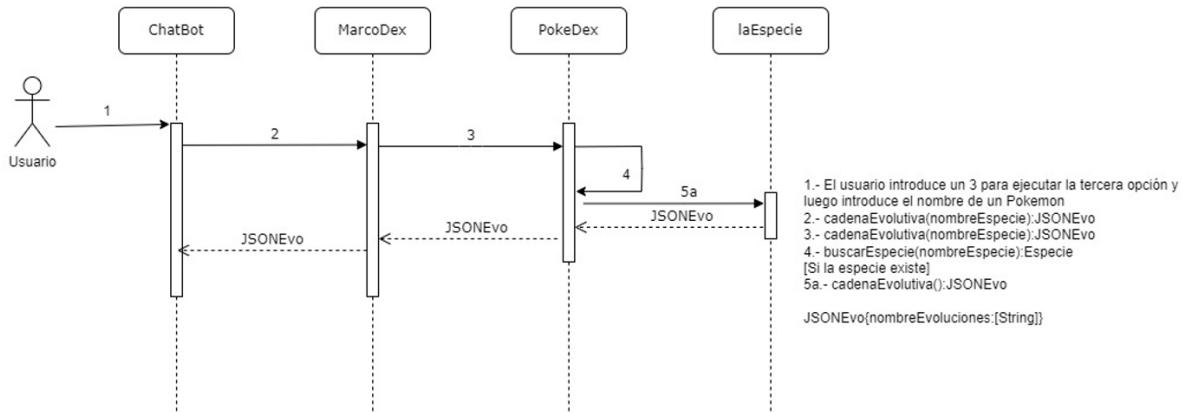


Figure 95: Opción 3

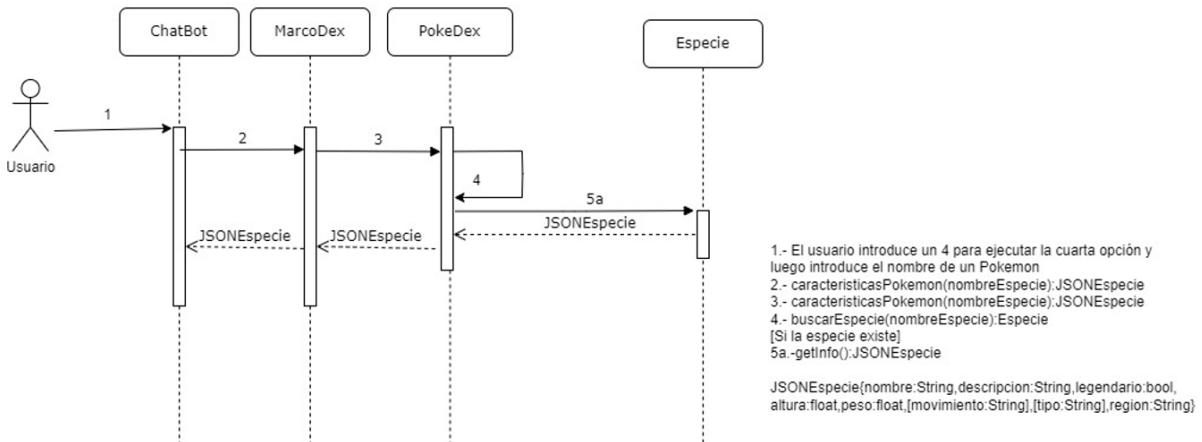


Figure 96: Opción 4

9.6 Ranking (Marco)

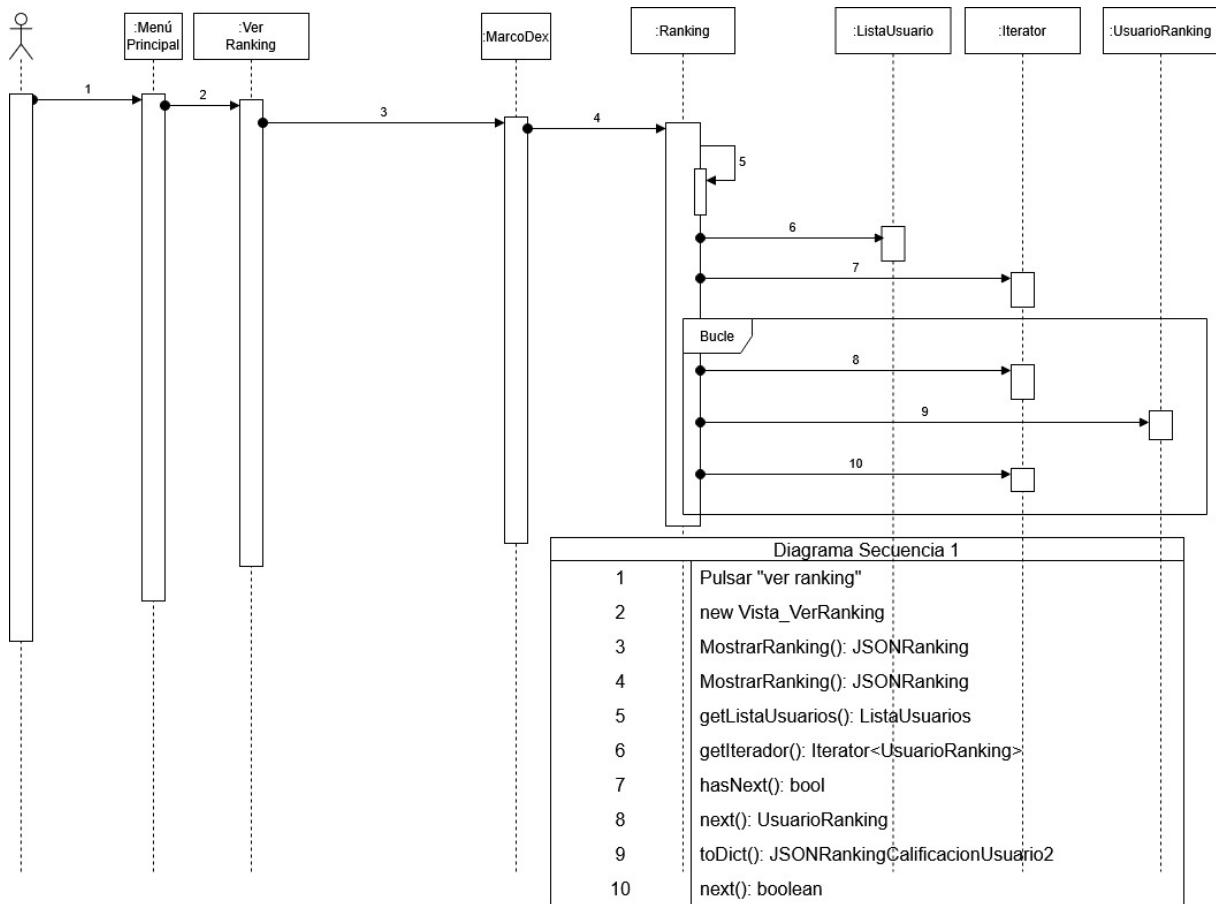


Figure 97: VerRanking - mostrar el ranking

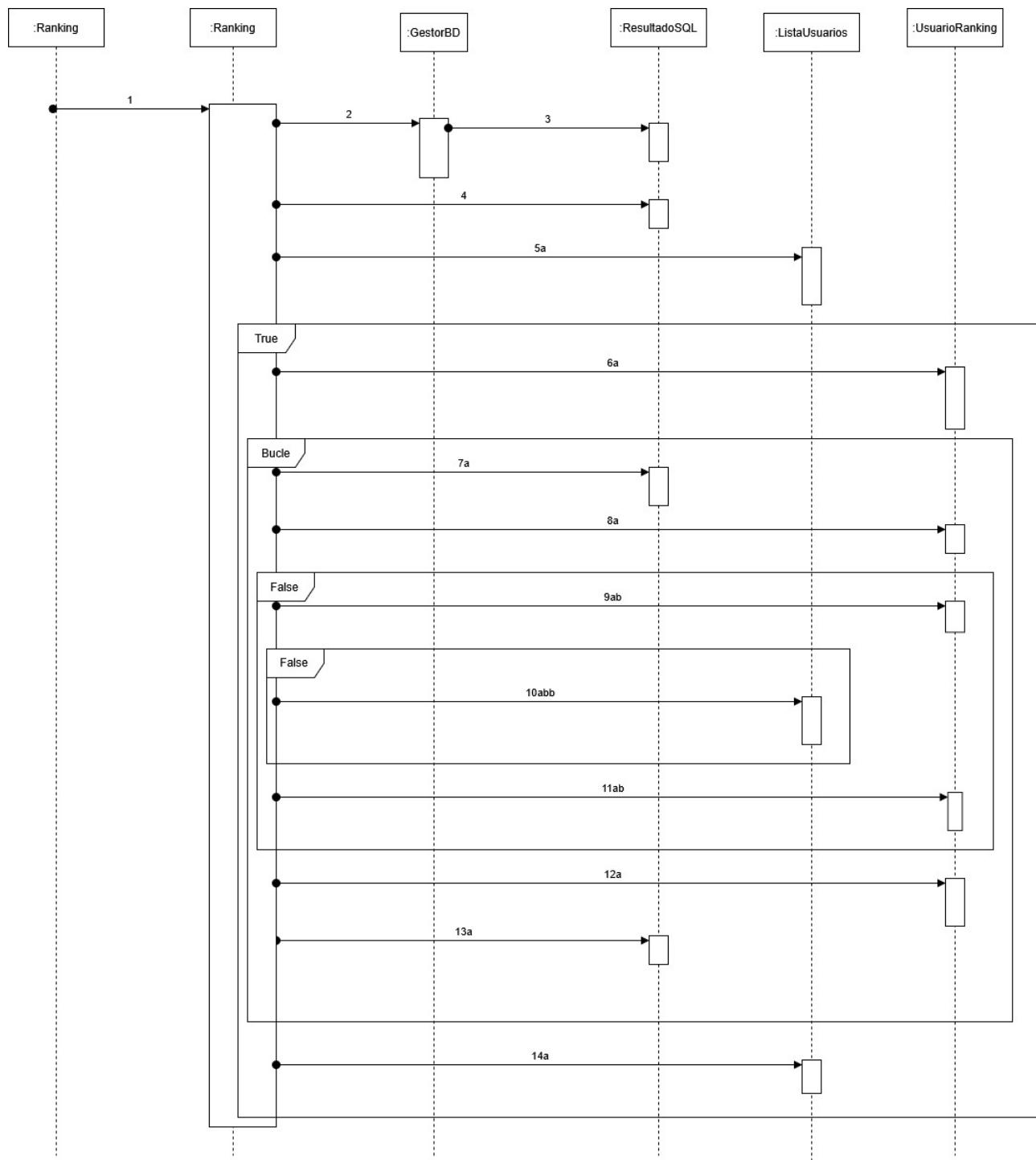


Figure 98: Ranking - getListaUsuarios

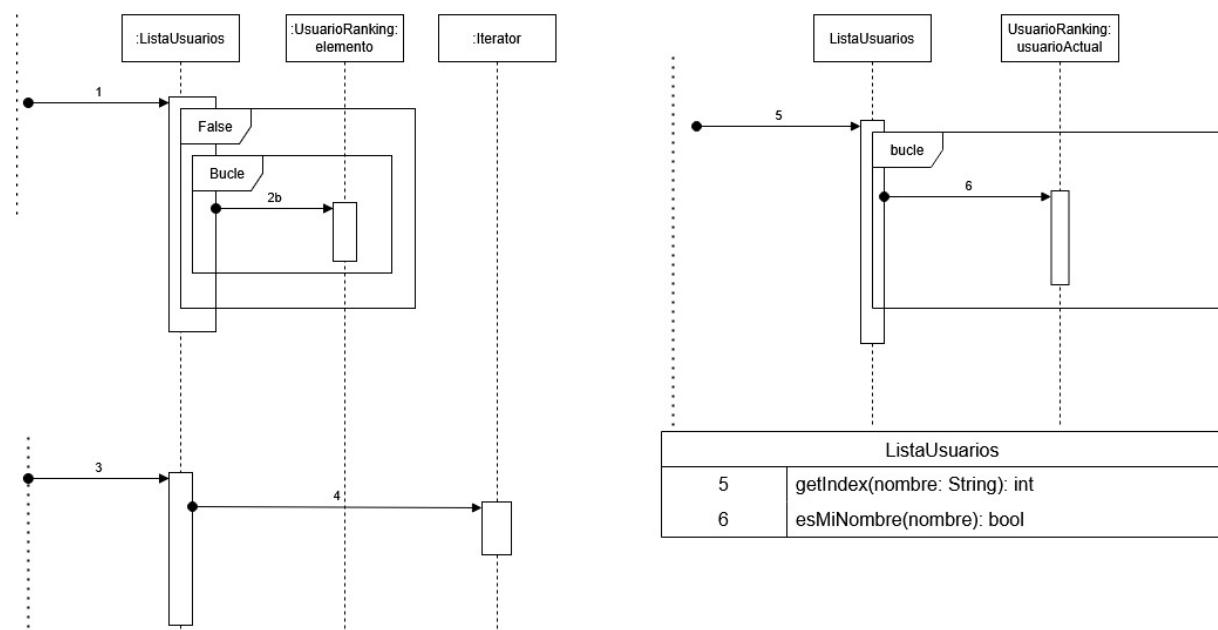
```

sql_ranking: SELECT Usuario.NombreUsuario, Pokemon.Rareza
from Usuario
inner join Equipo
on Usuario.NombreUsuario = Equipo.NombreUsuario
inner join PokemonEnEquipo on Equipo.idEquipo = PokemonEnEquipo.idEquipo
inner join Pokemon on Pokemon.idPokemon = PokemonEnEquipo.idPokemon
Order By Usuario.NombreUsuario;

```

Table	
1	getListaUsuarios(): ListaUsuarios
2	execSQL(sql_ranking): ResultadoSQL
3	new ResultadoSQL()
4	next(): bool
5a	new ListaUsuarios()
6a	usuario_actual = new UsuarioRanking("", 0)
7a	nombre_actual=getString("NombreUsuario"): String
8a	es_mi_nombre(nombre_actual): bool
9ab	es_mi_nombre("") : bool
10abb	insercion_ordenada(usuario_actual): void
11ab	usuario_actual = new UsuarioRanking(nombre_actual, 0)
12a	total+=getFloat("Rareza"): float
13a	next(): bool
14a	insercion_ordenada(usuario_actual): void

Figure 99: Ranking - getListaUsuarios



ListaUsuarios	
1	insercionOrdenada(elemento: UsuarioRanking): void
2b	compare(usuario_actual): int
3	getIterator(): Iterator<UsuarioRanking>
4	new Iterador(lista: ListaDeUsuariosRanking)
JSONRanking	{usuarios: [JSONRankingCalificacionUsuario1, ...]}
JSONRankingCalificacionUsuario2	{ nombre: string, rareza: float }
JSONRankingCalificacionUsuario1	{"nombre": string, "rareza: float, "posicion":}

Figure 100: ListaUsuarios

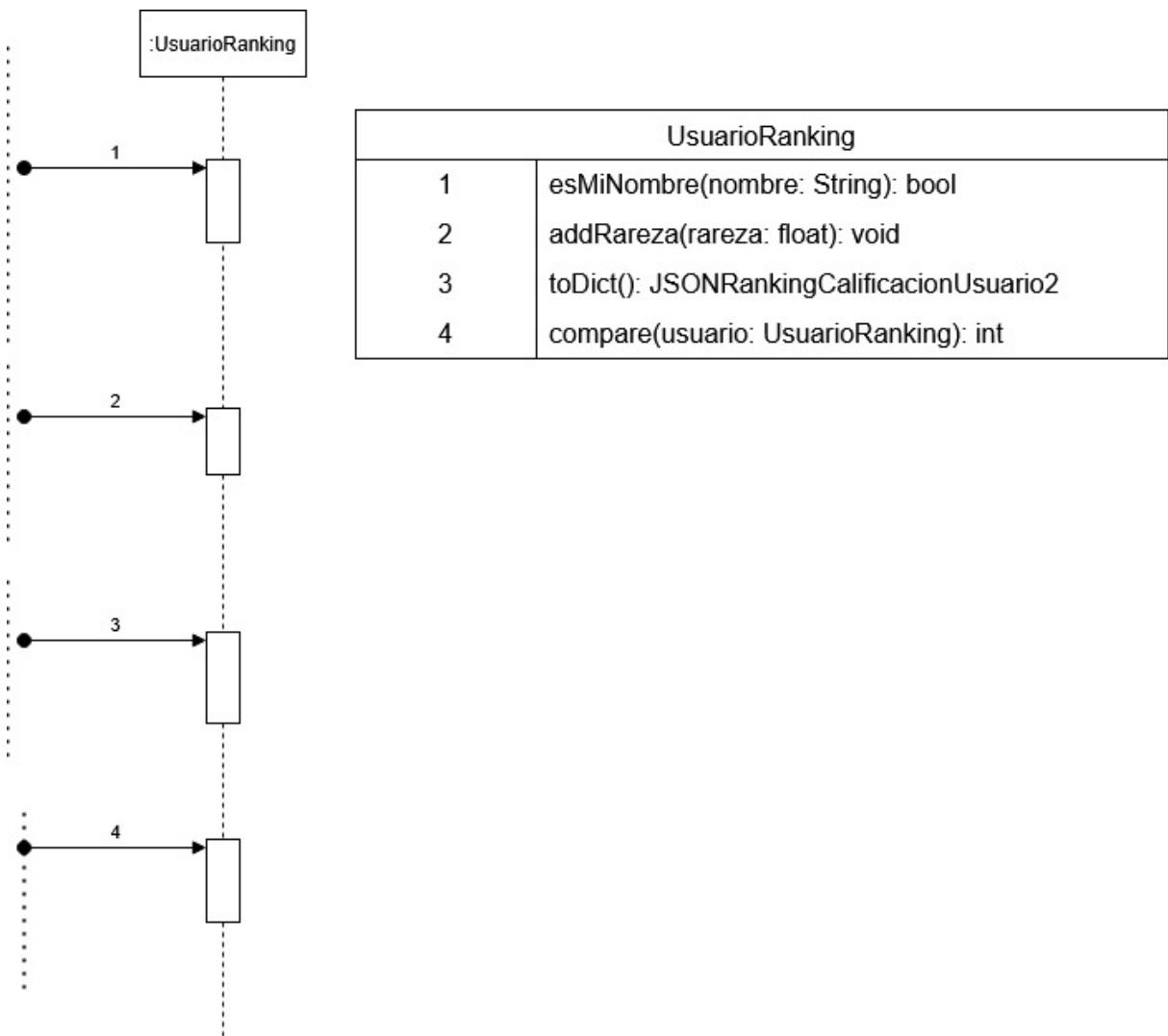


Figure 101: `UsuarioRanking`

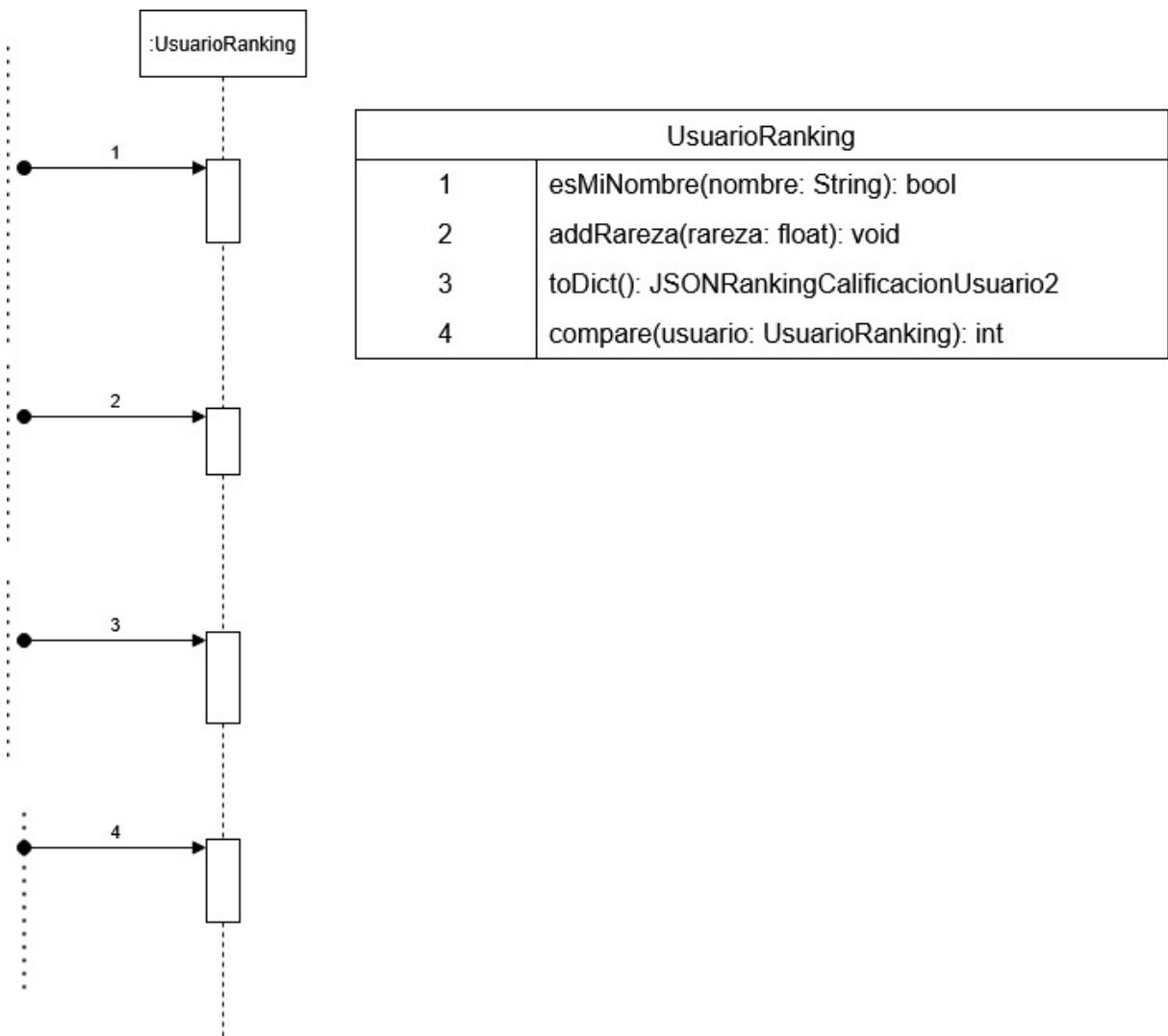


Figure 102: `UsuarioRanking`

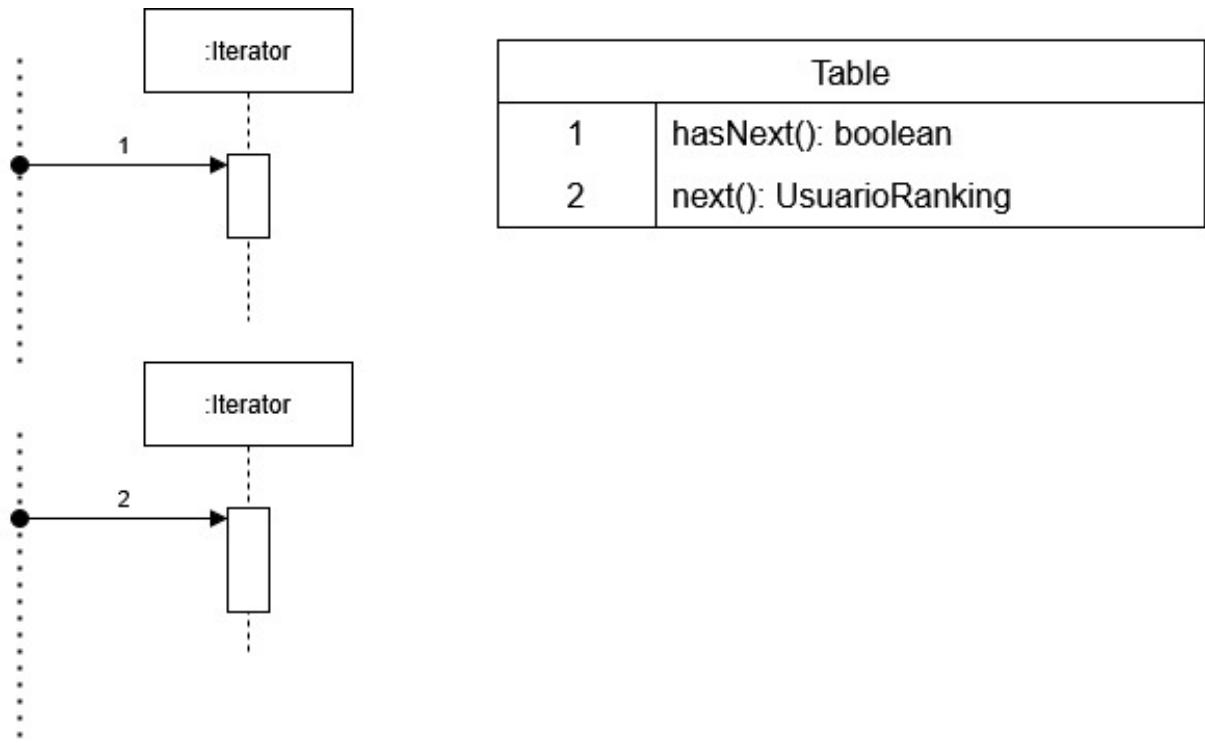


Figure 103: UsuarioRanking

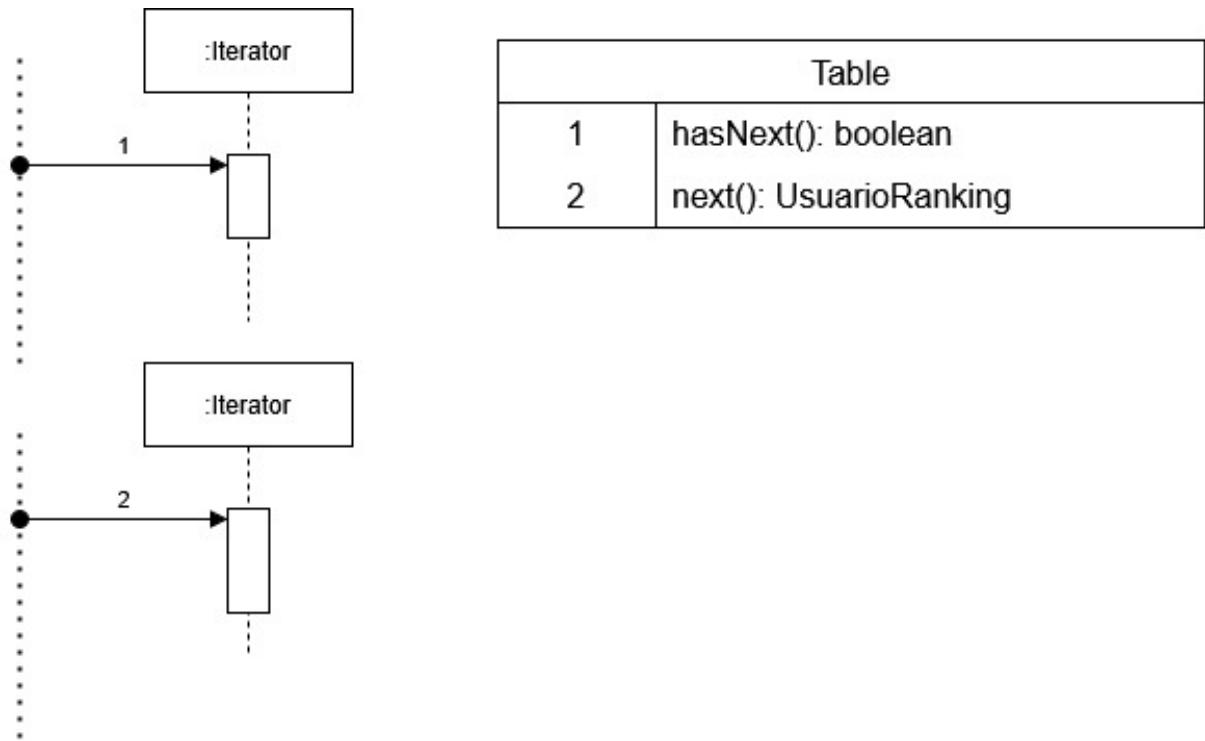


Figure 104: UsuarioRanking

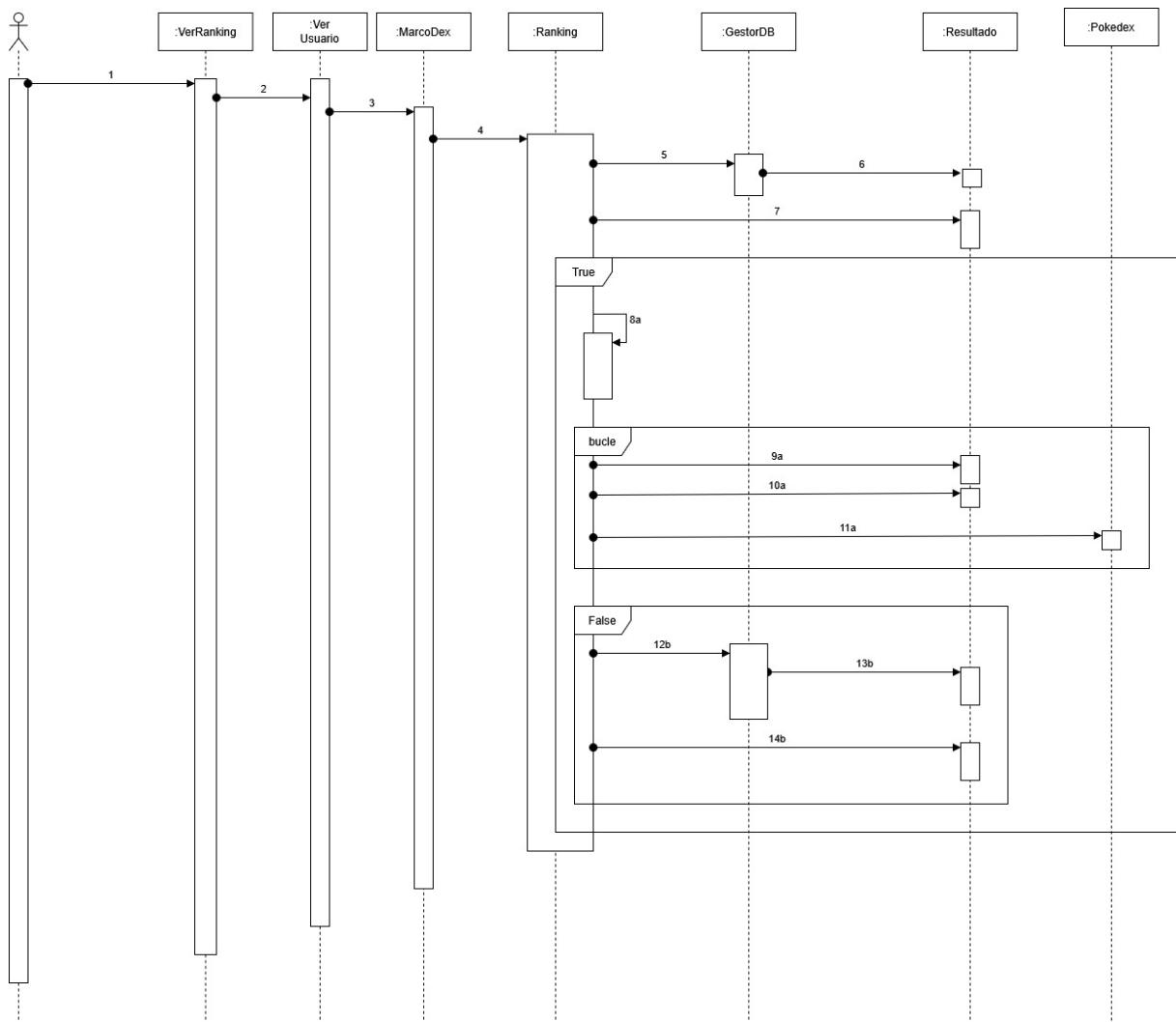


Figure 105: VerUsuario - mostrar Usuario

TarjetaUsuario	
1	Pulsa un usuario del ranking
2	new Vista_VerUsuario()
3	mostrarUsuario(pNombreUsuario: String): JSONRankingUsuario
4	mostrarUsuario(pNombreUsuario: String): JSONRankingUsuario
5	execSQL(sql_ranking2): ResultadoSQL
6	new ResultadoSQL()
7	next(): bool
8a	getPuestoByNombre(pNombreUsuario): Int
9a	getString("NombreEspecie"): String
10a	getString("NombreCustom"): String
11a	getInfo(nombreEspecie: string): JSONEspecie
12b	execSQL(sql_ranking3): ResultadoSQL
13b	new ResultadoSQL()
14b	next(): boolean

```

sql_ranking2: SELECT
    NombreEspecie, NombreCustom
  from Usuario
  inner join Equipo
  on Usuario.NombreUsuario = Equipo.NombreUsuario
  inner join PokemonEnEquipo
  on Equipo.idEquipo = PokemonEnEquipo.idEquipo
  inner join Pokemon
  on PokemonEnEquipo.idPokemon = Pokemon.idPokemon
  WHERE Usuario.NombreUsuario = ?;

sql_ranking3: SELECT * FROM AmigoDe
  INNER JOIN Usuario
  ON Usuario.NombreUsuario = AmigoDe.NombreUsuario1
  WHERE ( Usuario.NombreUsuario = ? AND AmigoDe.NombreUsuario2 = ? );

JSONRankingUsuario: {
    "nombre": string,
    "equipoEspecie": [ String ],
    "equipoCustom": [ String ],
    "fotoPokemon": [ String ],
    "puesto": int,
    "estado_amigo": int
}

datos = {
    "nombre": String,
    "descripcion": String,
    "legendario": self.legendario,
    "alturaMedia": self.alturaMedia,
    "pesoMedio": self.pesoMedio,
    "movimientos": self.movimientos,
    "tipos": self.tipos,
    "Imagen": String
}
}

```

Figure 106: VerUsuario - mostrar Usuario

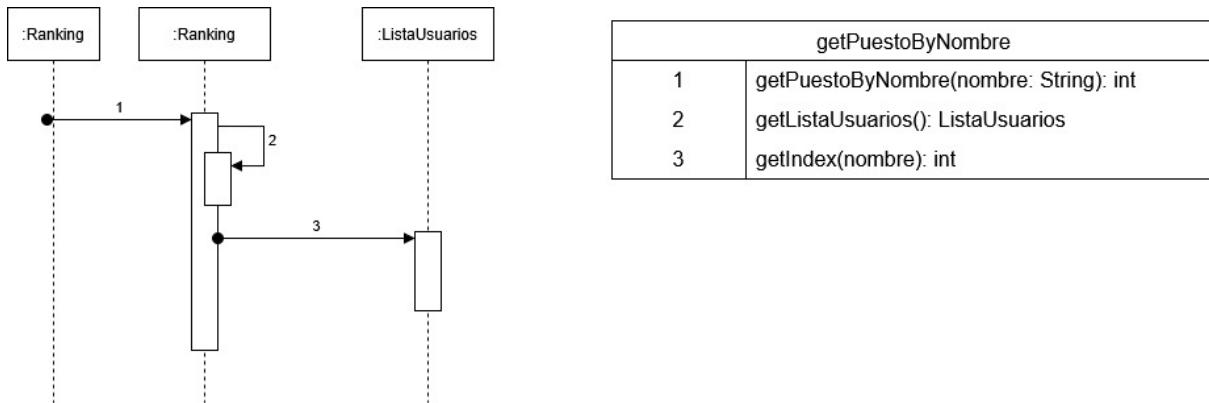
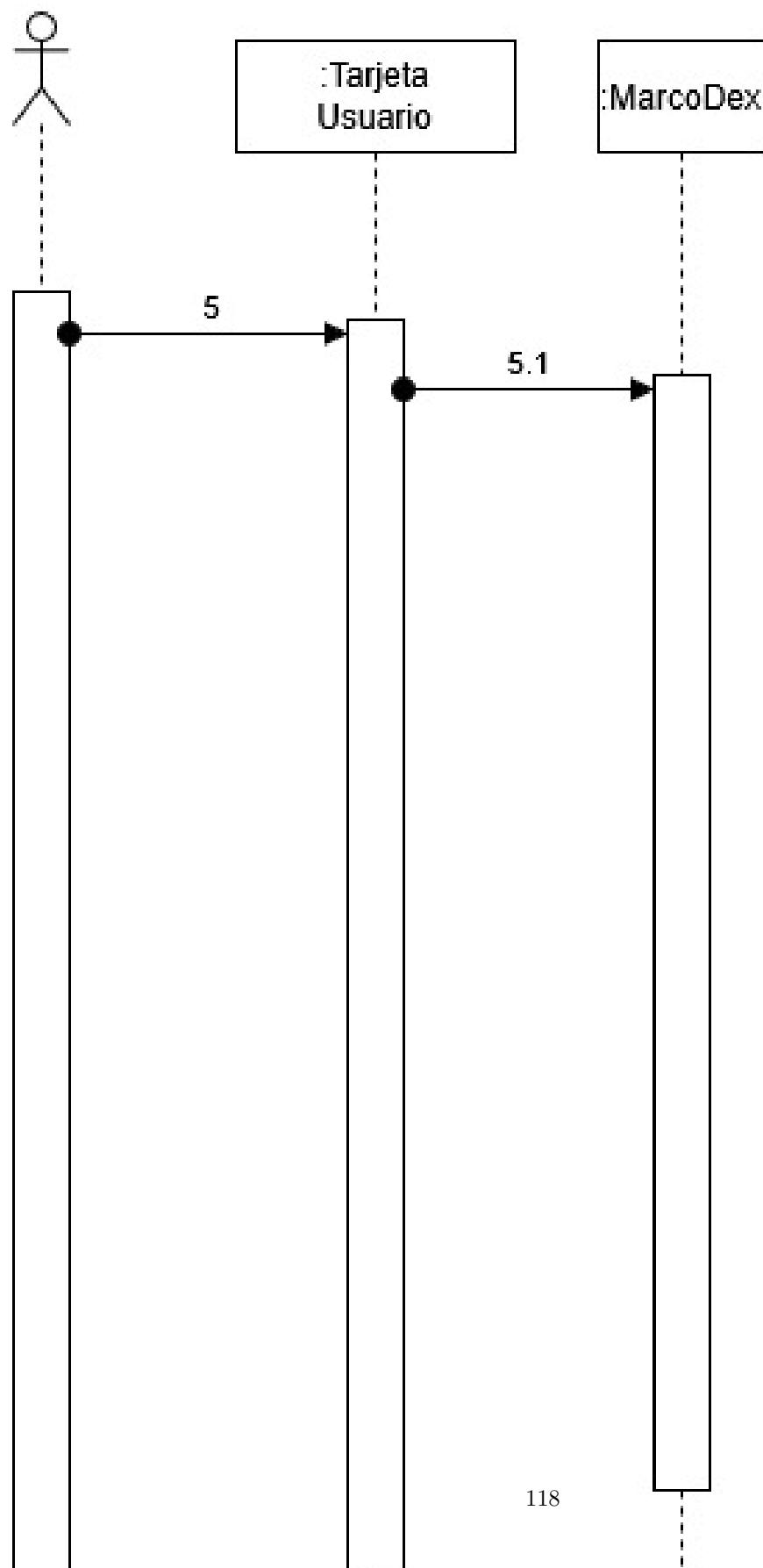


Figure 107: Ranking - getPuestoByNombre

5.- Pular botón "Solicitud amistad"

5.1.- anadirAmigoRanking(pNombreUsuario: String): bool



10 Problemas Encontrados

Durante el desarrollo del sistema **MARCODEX**, el equipo se ha enfrentado a diversos desafíos técnicos y organizativos. Al ser un grupo de seis integrantes, la coordinación y la unificación de criterios han sido puntos críticos, además de las dificultades inherentes al aprendizaje de nuevas tecnologías como Flask. A continuación, se detallan los principales problemas surgidos:

1. Dificultades de Coordinación y Control de Versiones

Al contar con seis desarrolladores trabajando en paralelo, la gestión del repositorio de código fue compleja. A pesar de dividir el trabajo por casos de uso (Gestión de Usuarios, Ranking, Chatbot, etc.), surgieron *conflictos de fusión (merge conflicts)* recurrentes en archivos comunes, especialmente en:

- **El Gestor de Base de Datos (GestorBD):** Al centralizar muchas consultas SQL, varios integrantes necesitaban modificar este archivo simultáneamente para sus respectivas funcionalidades.
- **El Menú Principal:** Al ser el punto de acceso a todas las funcionalidades, requería actualizaciones constantes por parte de todos los miembros.

Esto obligó al equipo a mejorar la comunicación y establecer turnos o ramas específicas para evitar sobrescribir el trabajo de los compañeros.

2. Duplicidad de Código y Refactorización

En las etapas iniciales, detectamos que funcionalidades similares se estaban implementando de forma redundante. El caso más evidente ocurrió entre los módulos de **Crear Equipo** y **Modificar Equipo**. Inicialmente, se trataron como procesos totalmente distintos, lo que generó duplicidad en la lógica de añadir y eliminar Pokémon de una lista.

- *Solución:* Como se observa en el diseño final, hubo que replantear la modificación de equipos. Se introdujo el concepto de **GestorCopiasEquipo** para manejar una "copia profunda" del equipo durante la edición. Esto permitió reutilizar lógica pero complicó la gestión de estados (saber qué cambios guardar o descartar al pulsar "Cancelar").

3. Curva de Aprendizaje de Flask y SQL

La integración de **Flask** fue un reto importante, ya que era una herramienta nueva para el equipo. Adaptar el diseño orientado a objetos (más propio de lenguajes como Java) a la estructura de un framework web en Python generó dudas sobre dónde colocar la lógica de negocio.

Además, el uso de consultas **SQL directas** en lugar de un ORM provocó errores de sintaxis difíciles de depurar y problemas con los tipos de datos al mapear los resultados de las consultas a los objetos de la aplicación (**Usuario**, **Pokemon**, ...).

4. Gestión de Datos Externos (API) e Imágenes

La funcionalidad de **Lista Pokémon**, que depende de una API externa, presentó problemas de rendimiento y estabilidad. Si la conexión fallaba o la API tardaba en responder, la experiencia de usuario se veía afectada. Asimismo, el manejo de las imágenes de los Pokémon para mostrarlas en el **Ranking** y los **Detalles del Equipo** fue problemático, requiriendo decidir si almacenarlas localmente o cargar las URLs dinámicamente, afectando a la velocidad de carga de la página.

5. Inconsistencias en el Diseño Inicial

Al comenzar la implementación, nos dimos cuenta de que algunos diagramas de secuencia iniciales no contemplaban todos los flujos de error posibles (por ejemplo, intentar ver el **ChangeLog** sin tener amigos o filtrar por un usuario inexistente). Esto obligó a "improvisar" soluciones en el código que no estaban documentadas originalmente, requiriendo una actualización posterior de la documentación para que reflejara la realidad del sistema.

11 Conclusiones

Este proyecto nos ha servido para asentar conocimientos previos y, sobre todo, para enfrentarnos a herramientas que no habíamos tocado hasta ahora. El uso de Flask y la integración directa de la programación con bases de datos han sido el núcleo del trabajo lo cual ha supuesto un reto importante en esta última etapa del proyecto.

Más allá de la programación, aplicar la planificación y los diagramas vistos en clase ha sido clave para coordinar el trabajo en grupo y estructurar el código de forma ordenada. Nos llevamos la experiencia de haber desarrollado una plataforma completa y funcional partiendo de tecnologías que no dominábamos al principio del curso.