ANEXO IV: Diseño del sistema

Discovering 2.0: La aplicación que enseña curiosidades sobre el mundo animal haciendo uso de realidad virtual

Trabajo de Fin de Grado

GRADO EN INGENIERÍA INFORMÁTICA

Julio de 2023

Autora

Cristina Alejandra Crespo Jiménez

Tutores

Gabriel Villarrubia González

André Filipe Sales Mendes

ANEXO IV: Diseño del sistema

Índice

Índice de figuras

1. Introducción

Esta fase se enfoca en el dominio de la solución, lo cual implica una aproximación cercana a la implementación. Los nombres de las clases, los métodos y los atributos presentados a continuación reflejarán una visión próxima al resultado final del sistema. Se procederá al refinamiento de las especificaciones realizadas en etapas anteriores.

2. Modelo de diseño

Se trata de una representación conceptual del sistema que servirá como documentación y guía para su diseño. Esto permite modelar el sistema a un nivel más bajo que está cerca de la programación. Proveniente de esta etapa, se conseguirá un desglose detallado de paquetes, clases de diseño y casos de uso del sistema a implementar [1].

2.1. Patrones arquitectónicos

Elegir los patrones correctos es crucial ya que implica considerar diversos aspectos para su elección, como es el número y la estructura de los paquetes que conforman el sistema así como las relaciones entre ellos [2].

2.1.1. Patrón MVC

El patrón Modelo-Vista-Controlador (MVC) divide el trabajo en tres partes, por una parte están los datos de la aplicación, por otra la interfaz de usuario, y por último la lógica de control:

- Modelo, encapsula los datos y la funcionalidad central de forma independiente a la entrada y salida.
- Vista, presenta la información al usuario obteniendo los datos del modelo.
- Controlador, sirve peticiones del modelo o de la vista, propagándose al sistema a través de los controladores.

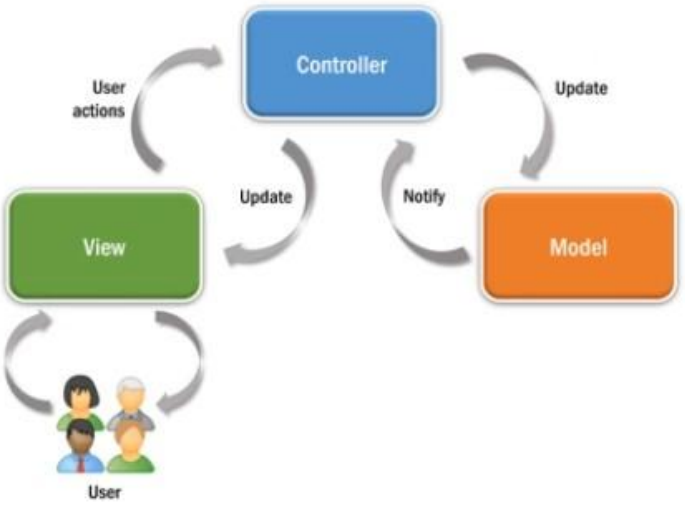

Figura 1: Esquema patrón MVC [3]

Es importante destacar que en Unity, las referencias de las clases MVC están dispersas en todo el código, por lo que es necesario tener un objeto de referencia raíz que funcione como contenedor de los datos relacionados. A mayores, se emplean componentes para acciones específicas que no notifican ni almacenan información.

2.1.2. Patrón Entidad-Componente (EC)

Se emplea principalmente en el desarrollo del videojuego. Basado en el principio de composición sobre herencia, crea una jerarquía de elementos formando entidades (GameObject), que a su vez cada una está compuesta por uno o más componentes (Rigidbody, Collider, Scripts…) que añaden la funcionalidad.

Este patrón resuelve el problema de herencia múltiple, que considera la situación de la *Figura 18*. Supongamos que HijaA e HijaB quieren modificar el mismo dato en Madre, eso podría causar un conflicto. Al segmentar las funcionalidades de las entidades y reutilizar componentes, evitamos que esto pase. Pero si empleáramos un gran número de componentes, podría dificultar la comprensión del proyecto, para abordar esto, se utiliza el patrón MVC.

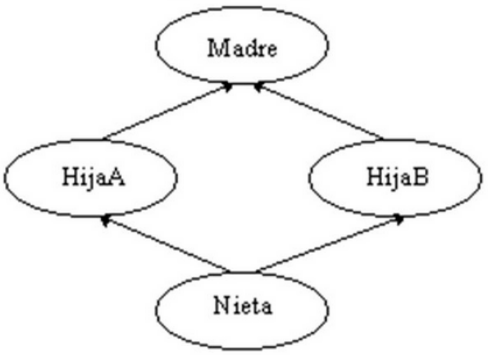

Figura 2: Problema herencia múltiple (Castro, R.) [4]

2.2. Subsistemas de diseño

A continuación se procederá a descomponer el sistema en paquetes y subpaquetes para que sean más sencillos de manejar.

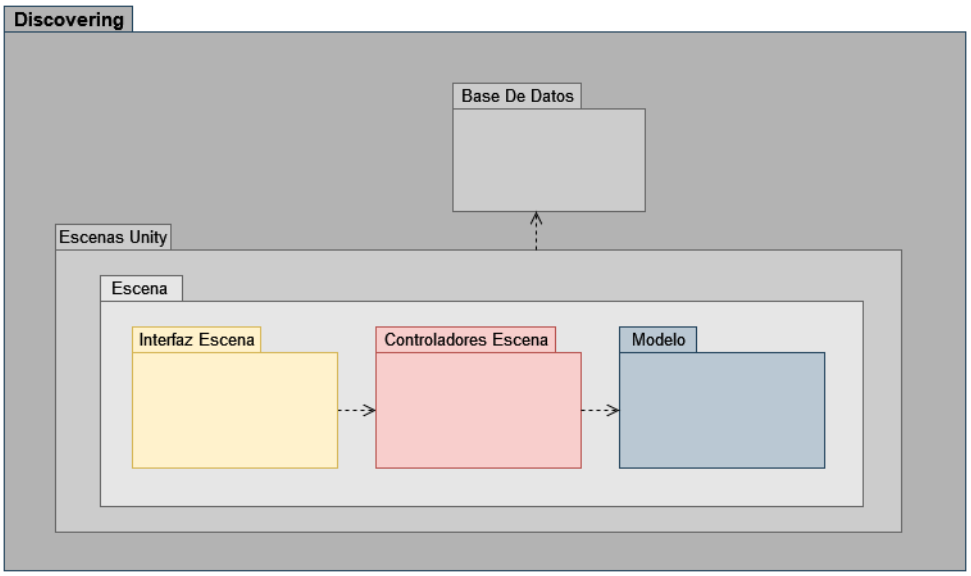

Figura 3: Subsistema de diseño

Discovering hace referencia al videojuego en sí que se genera con Unity. Dentro se encuentran dos paquetes:

- Escenas Unity, donde se efectúa la representación general del contenido de cada escena que conforma la aplicación del MVC adaptado.
- Base De Datos, se ocupa de la comunicación con la base de datos para el guardado y la extracción de datos.

2.3. Clases de diseño

Debido a que Unity sigue el patrón MVC, en este apartado se van a mostrar las clases de diseño por escenas, haciendo hincapié en la estructura de vista, controlador y modelo de cada escena [4].

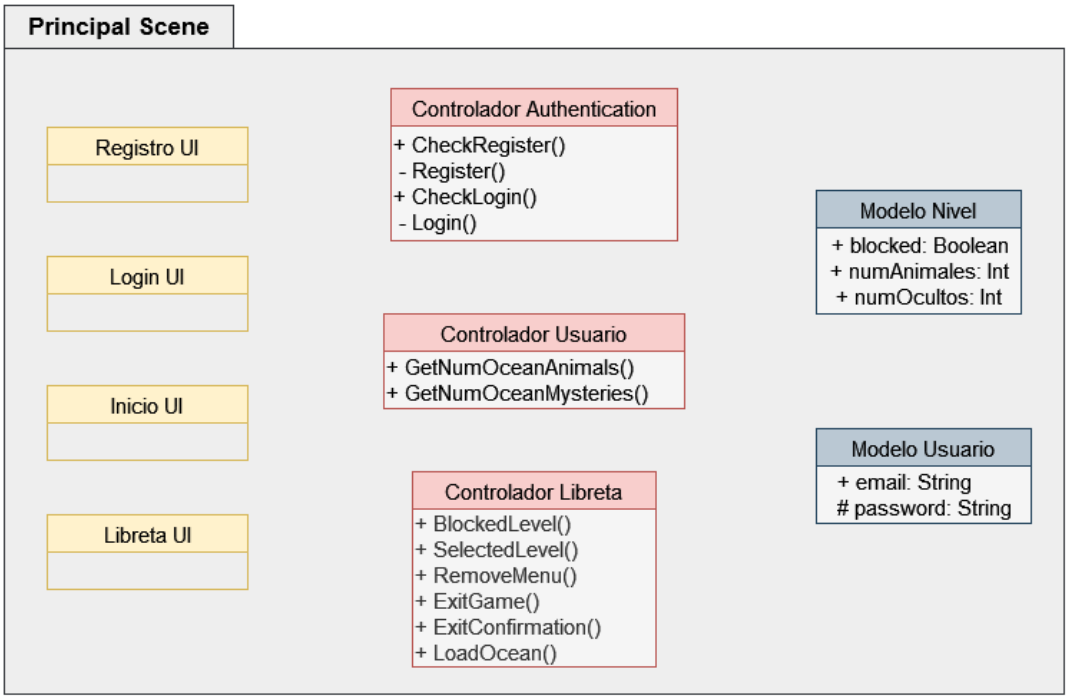

Figura 4: Clase de diseño - Principal Scene

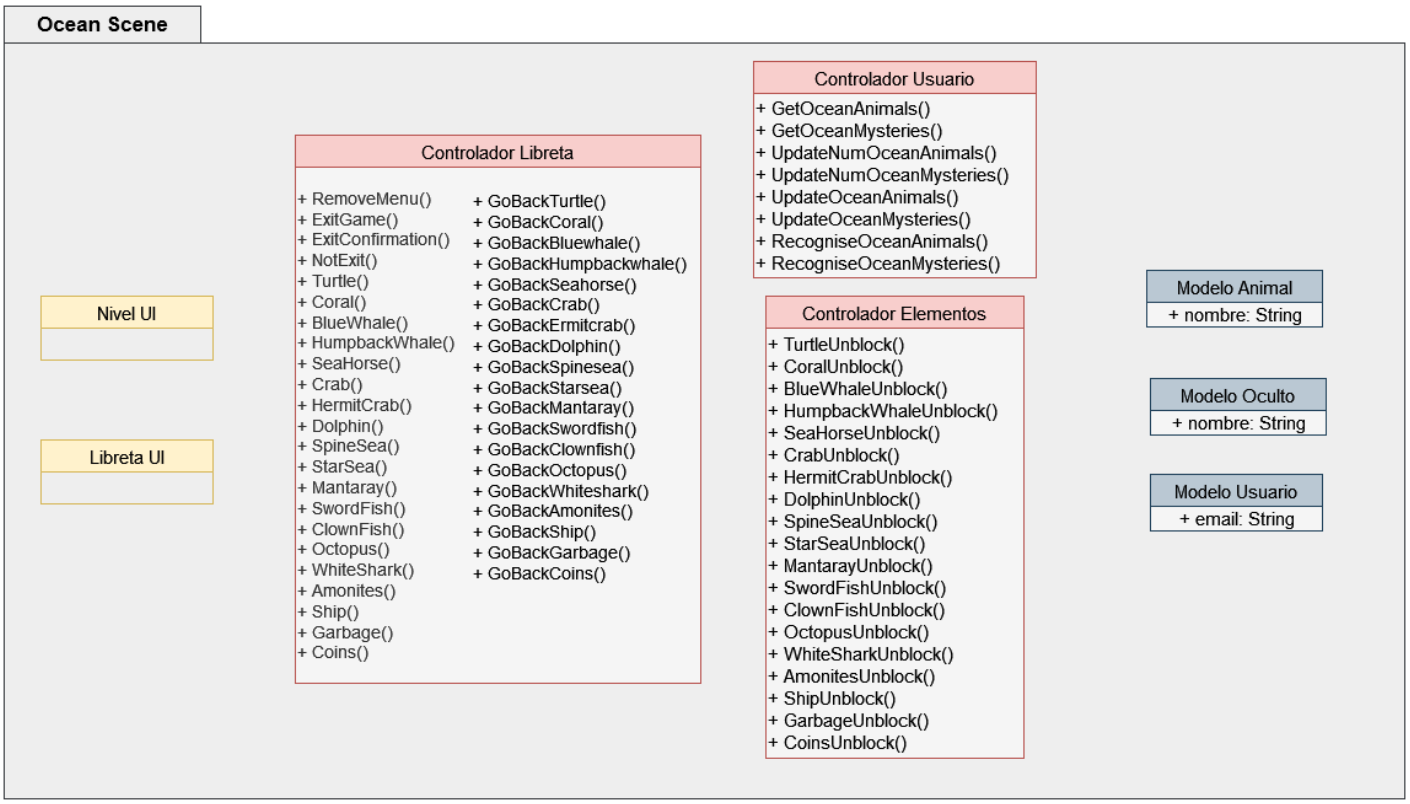

Figura 5: Clase de diseño - Ocean Scene

2.4. Vista arquitectónica

A continuación, se va a componer la vista arquitectónica del sistema basándose en el patrón MVC.

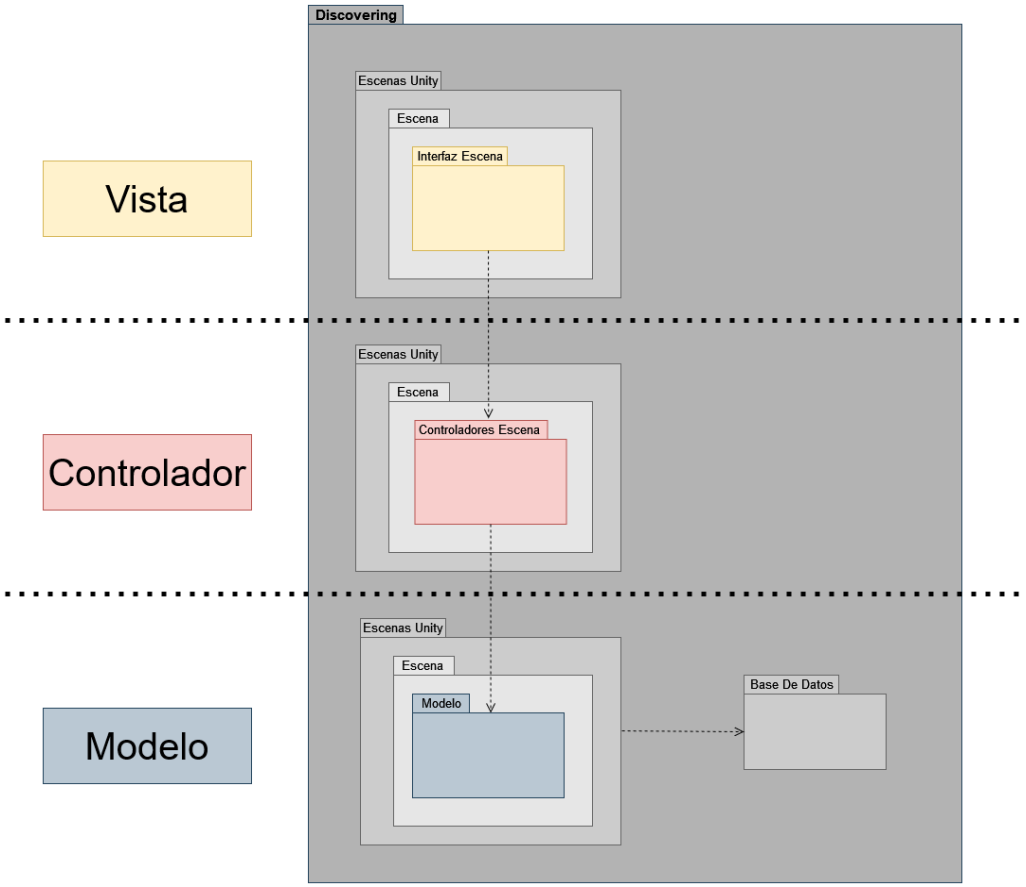

Figura 6: Vista arquitectónica

2.5. Realización de casos de uso

Los diagramas de secuencia de diseño, sacados de los casos de uso, permiten representar las interacciones entre los objetos del sistema mediante el intercambio de mensajes de envío y respuesta, ayudando así a definir las relaciones entre ellos.

2.5.1. Diagramas de secuencia de Gestión de usuarios

UC-001 Registro de usuarios

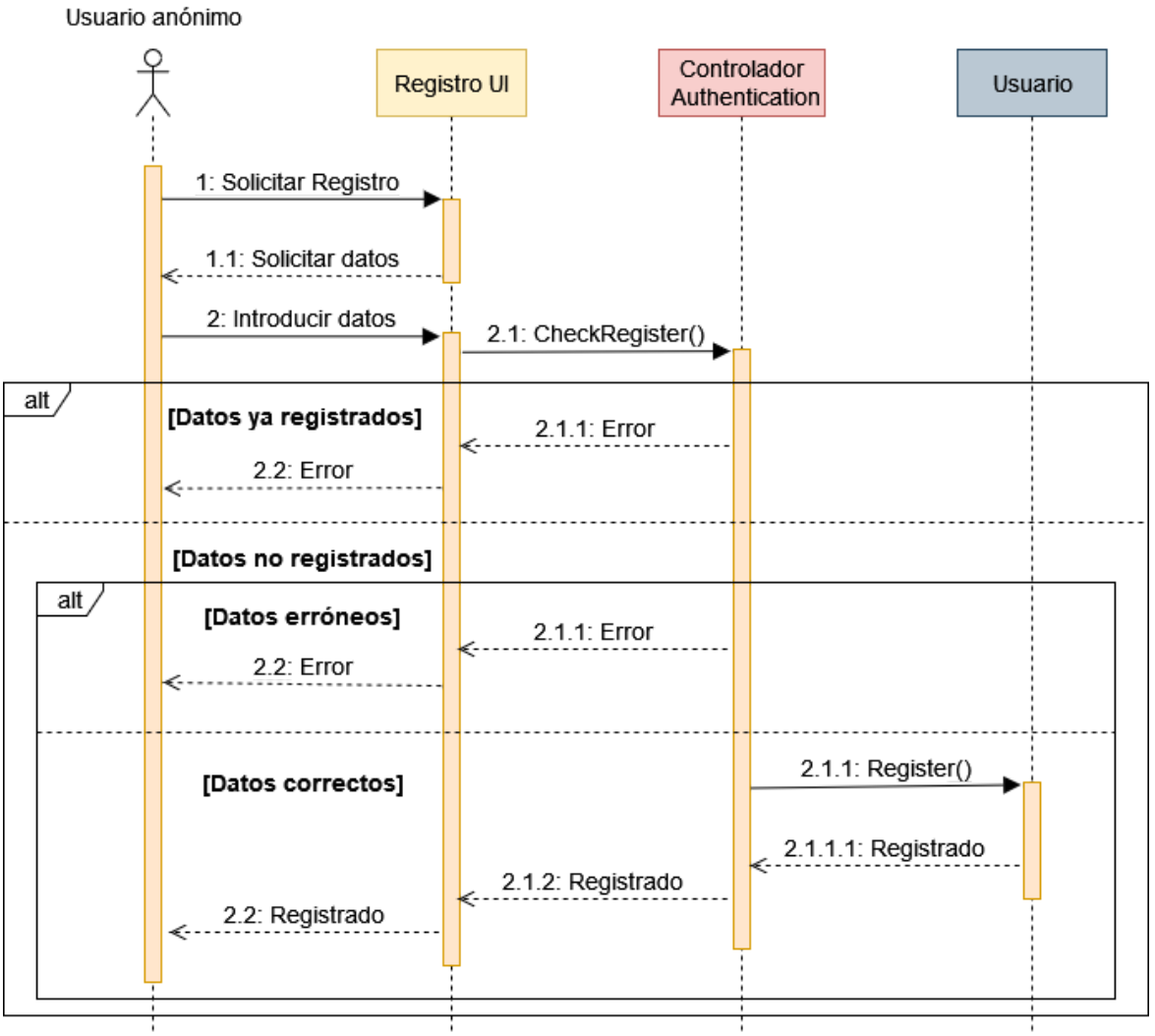

Figura 7: Diagrama de secuencia UC-001

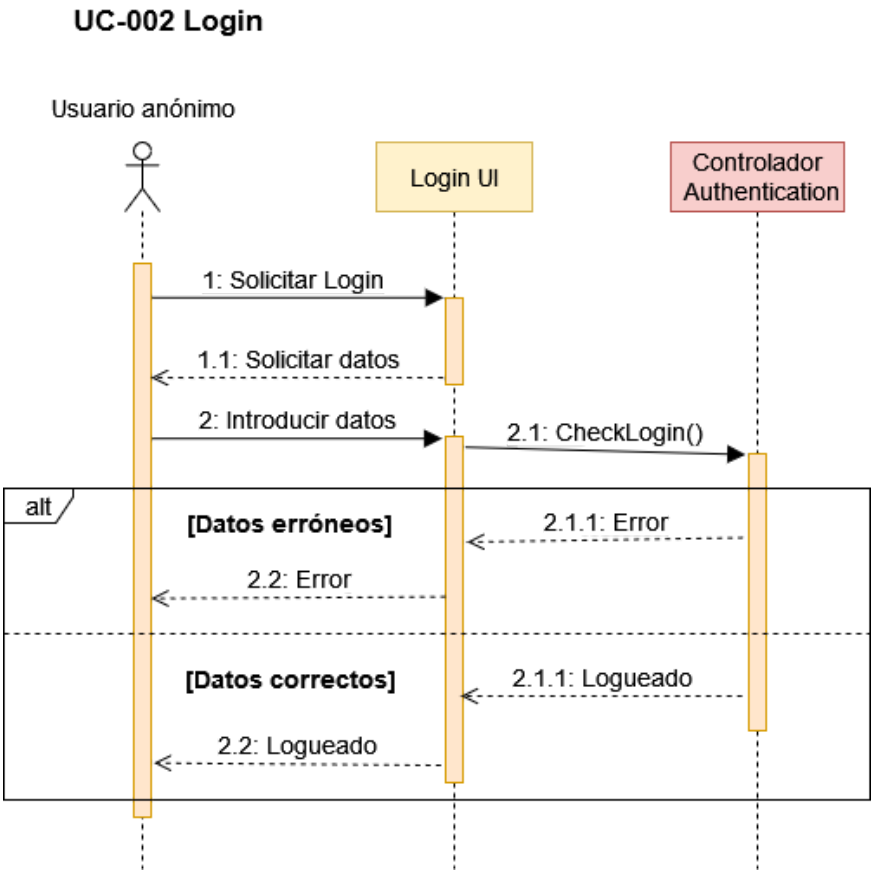

Figura 8: Diagrama de secuencia UC-002

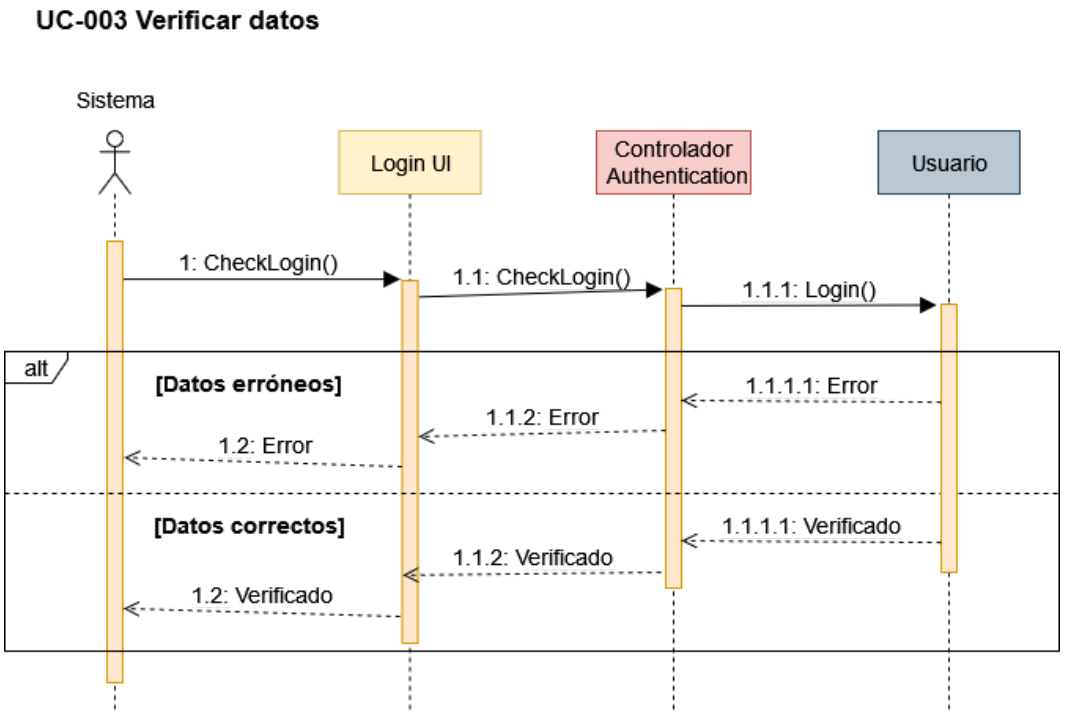

Figura 9: Diagrama de secuencia UC-003

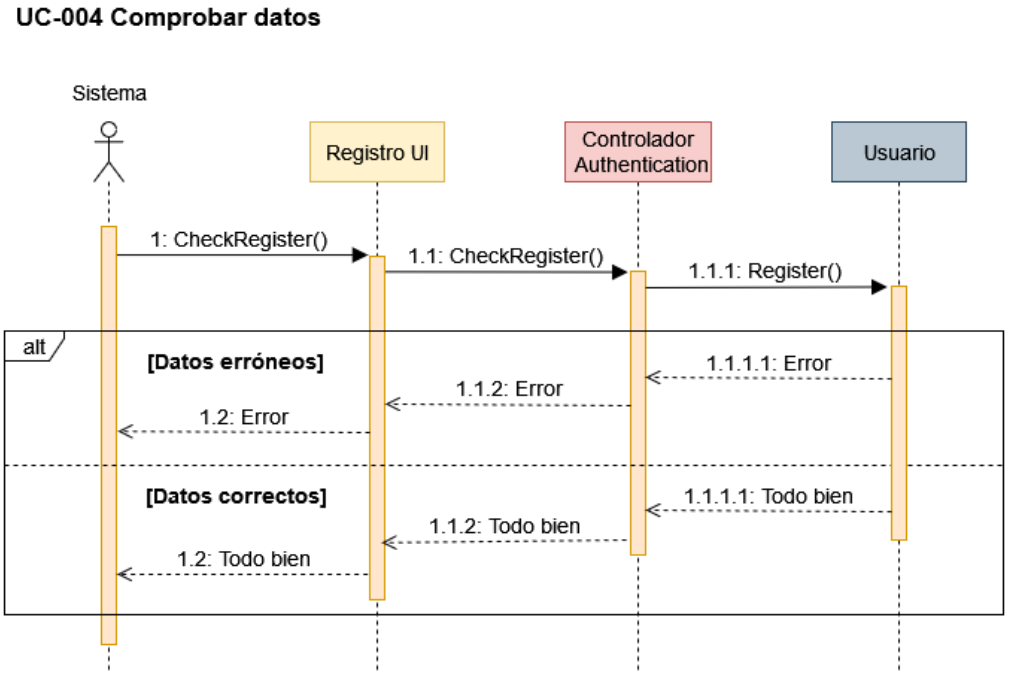

Figura 10: Diagrama de secuencia UC-004

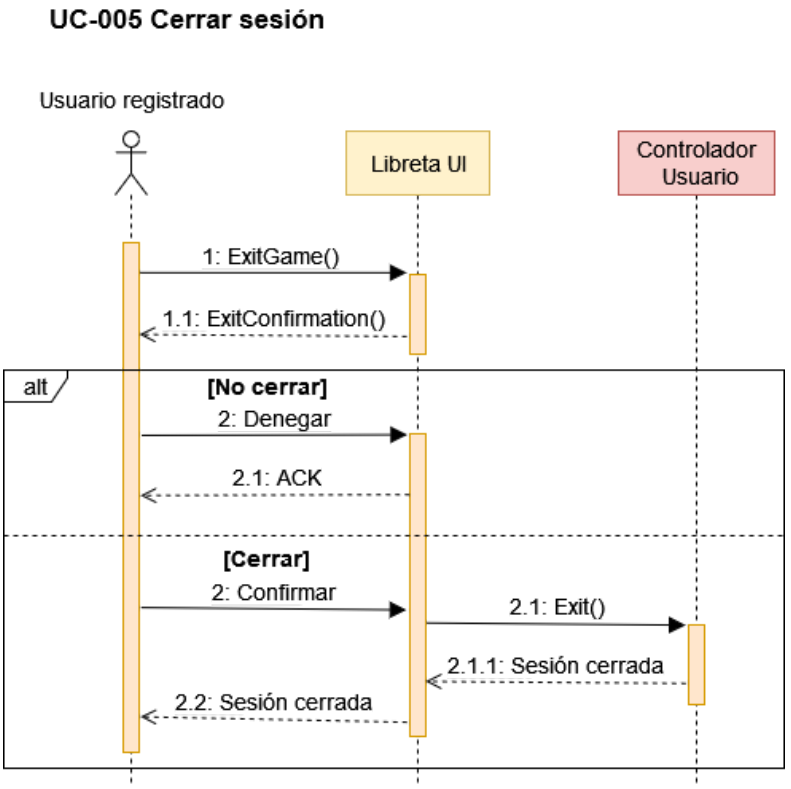

Figura 11: Diagrama de secuencia UC-005

2.5.2. Diagramas de secuencia de Gestión de la información

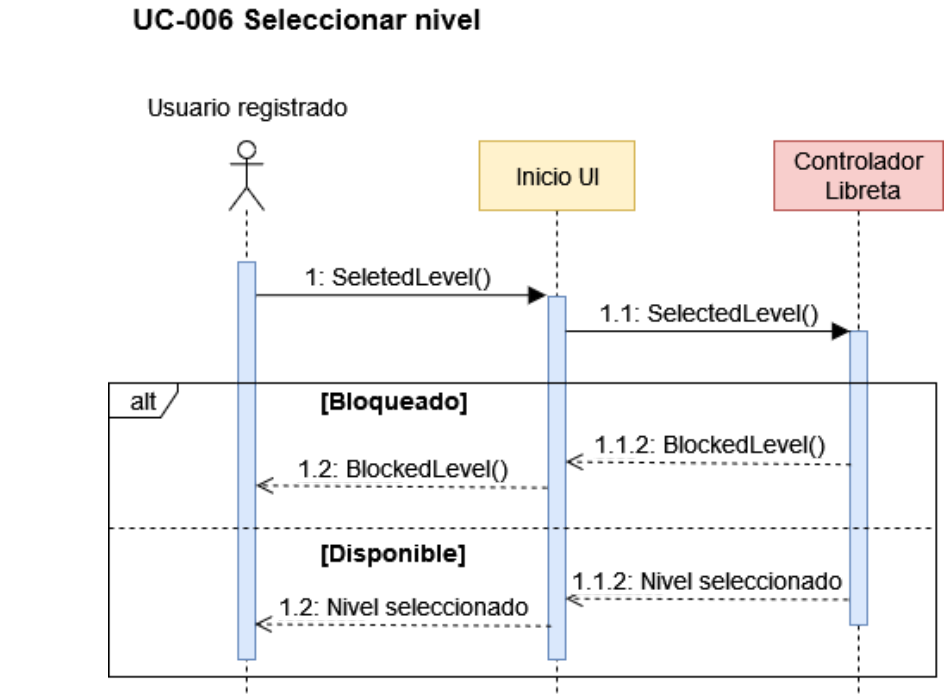

Figura 12: Diagrama de secuencia UC-006

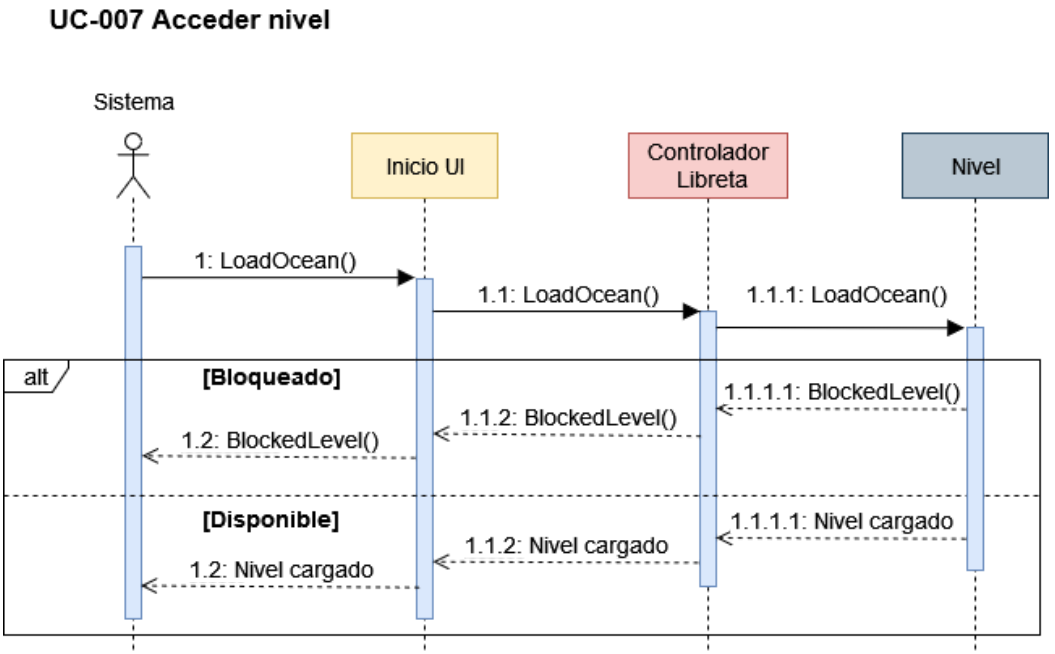

Figura 13: Diagrama de secuencia UC-007

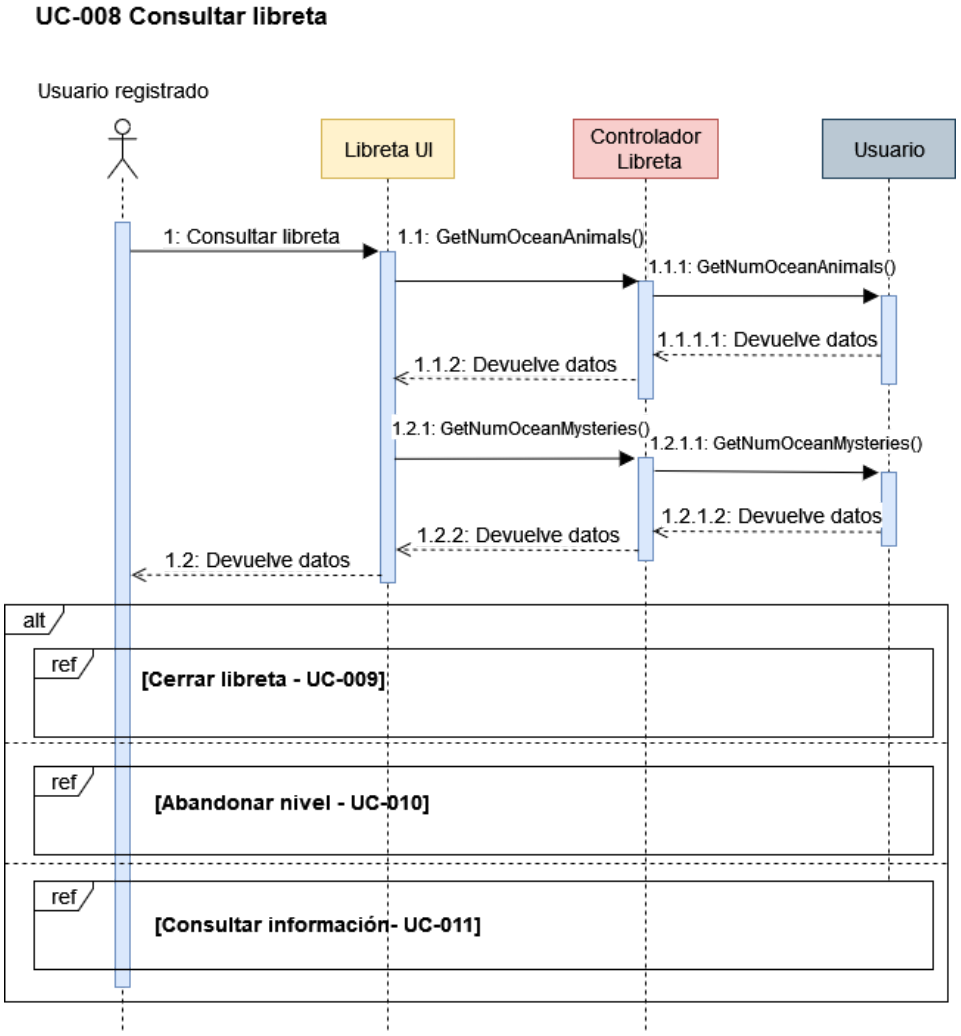

Figura 14: Diagrama de secuencia UC-008

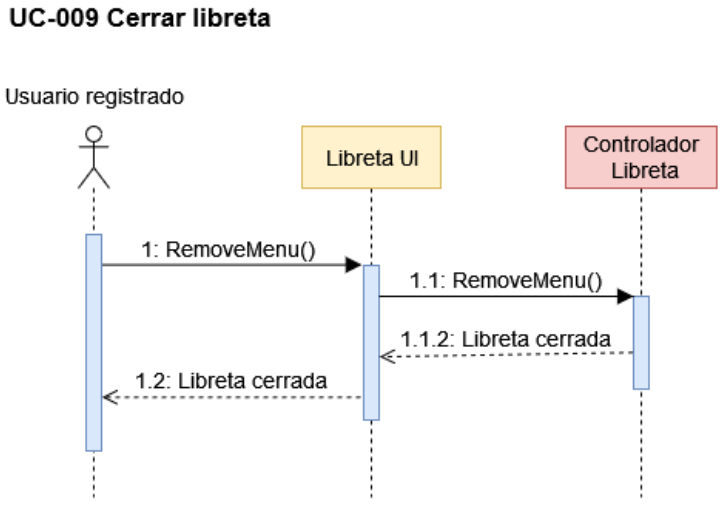

Figura 15: Diagrama de secuencia UC-009

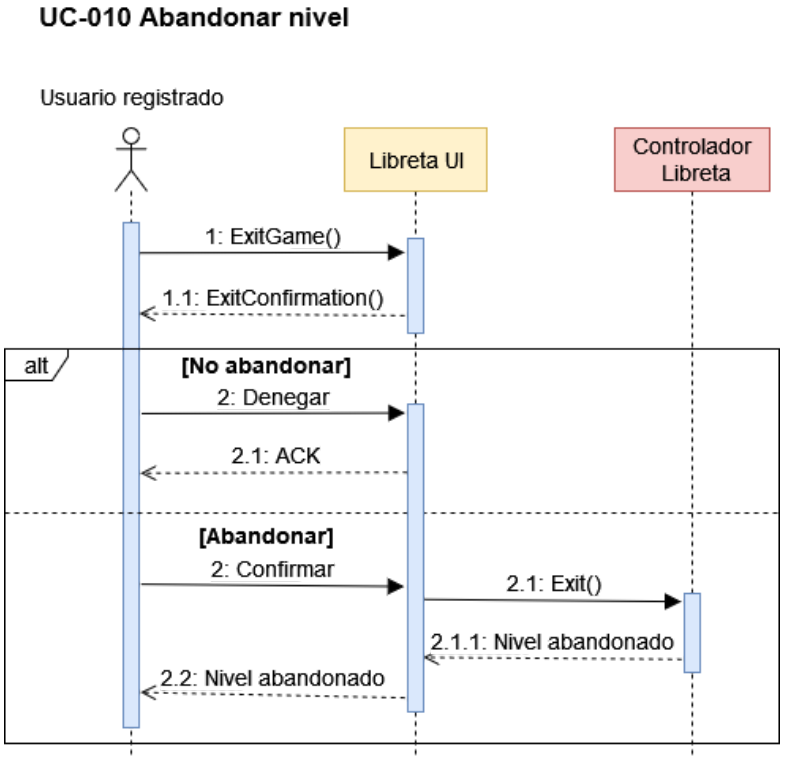

Figura 16: Diagrama de secuencia UC-010

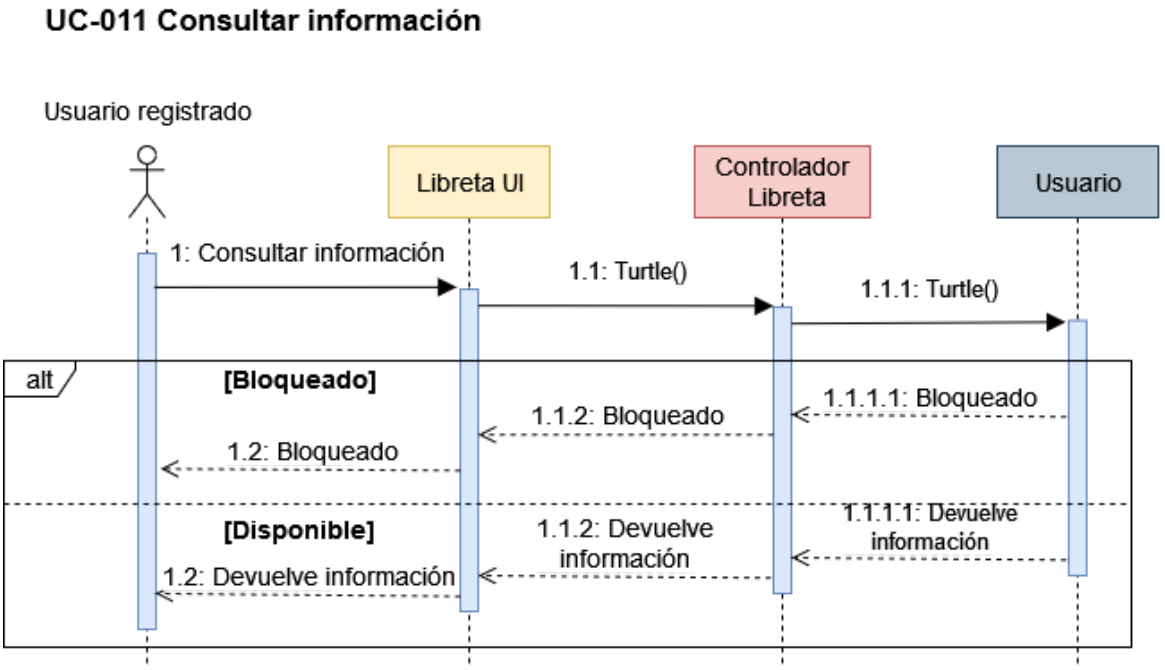

Figura 17: Diagrama de secuencia UC-011

UC-012 Cerrar información

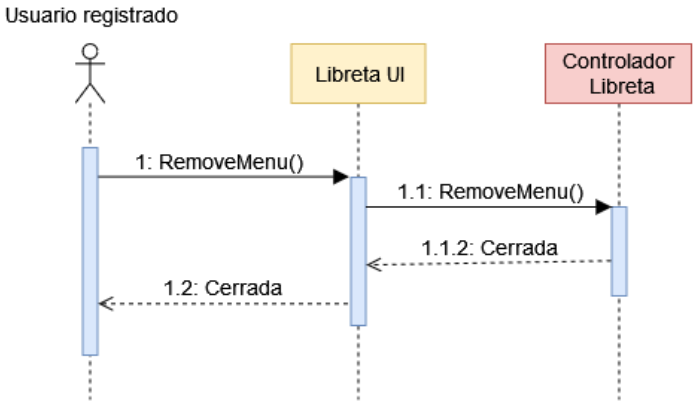

Figura 18: Diagrama de secuencia UC-012

2.5.3. Diagramas de secuencia de Gestión de elementos encontrados

UC-013 Encontrar animal

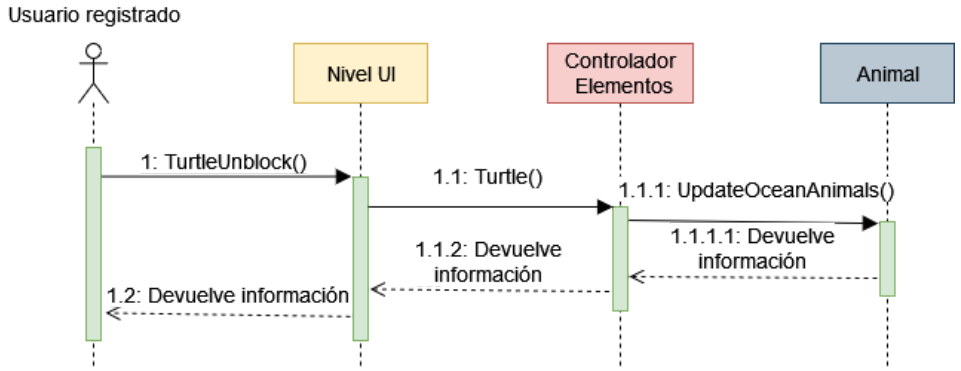

Figura 19: Diagrama de secuencia UC-013

UC-014 Mostrar información animal

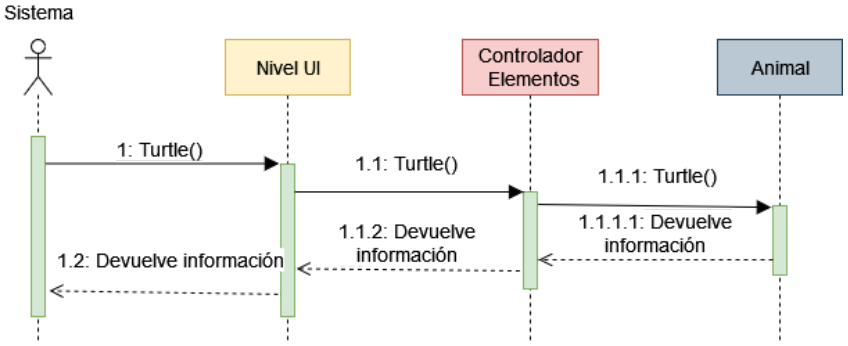

Figura 20: Diagrama de secuencia UC-014

UC-015 Encontrar objeto oculto

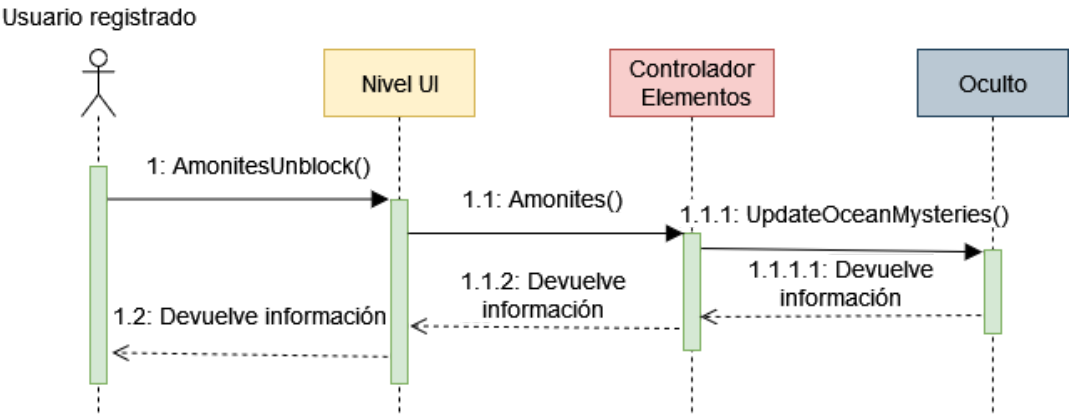

Figura 21: Diagrama de secuencia UC-015

UC-016 Mostrar información objeto oculto

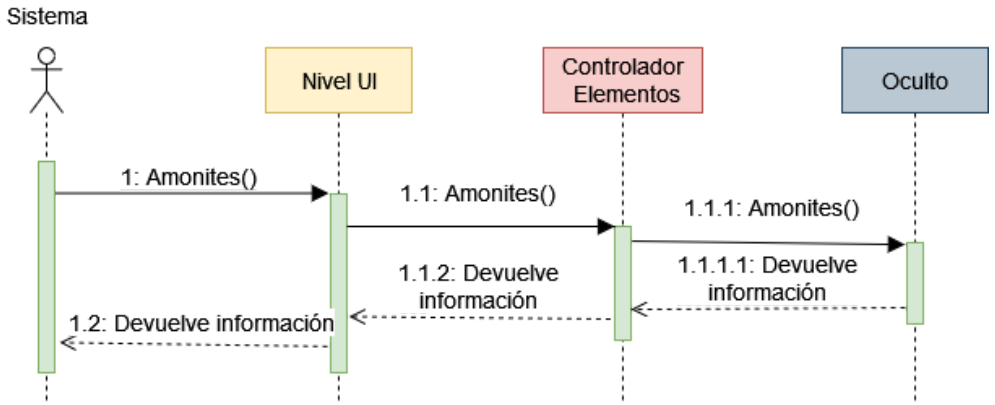

Figura 22: Diagrama de secuencia UC-016

UC-017 Actualizar la base de datos

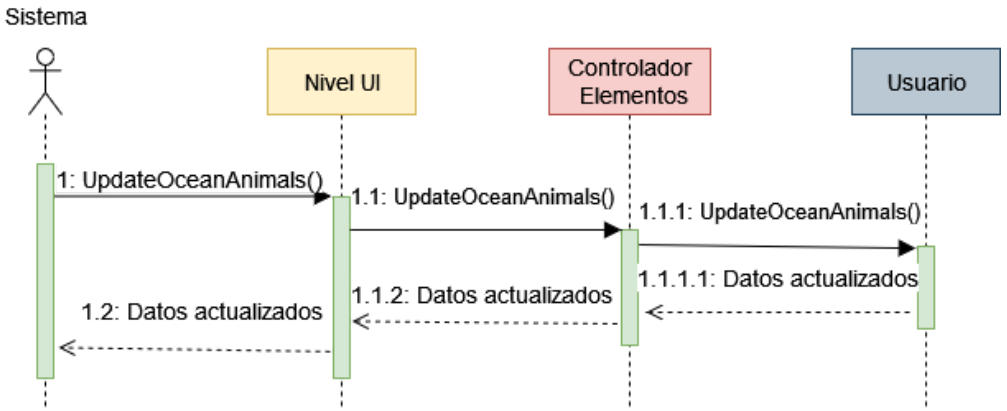

Figura 23: Diagrama de secuencia UC-017

2.5.4. Diagramas de secuencia de Gestión de las escenas

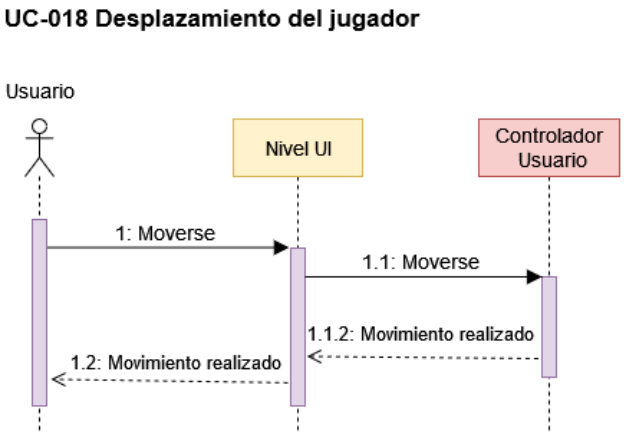

Figura 24: Diagrama de secuencia UC-018

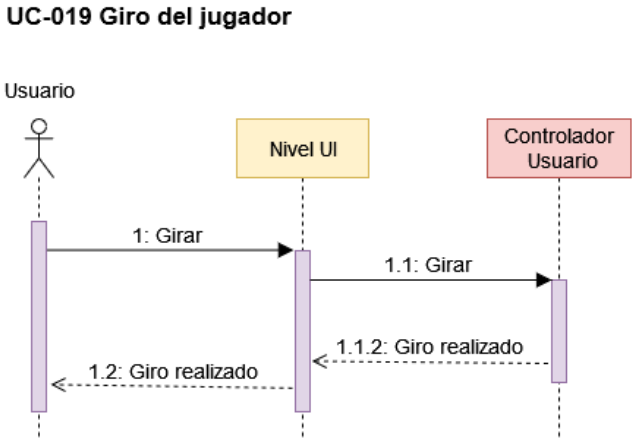

Figura 25: Diagrama de secuencia UC-019

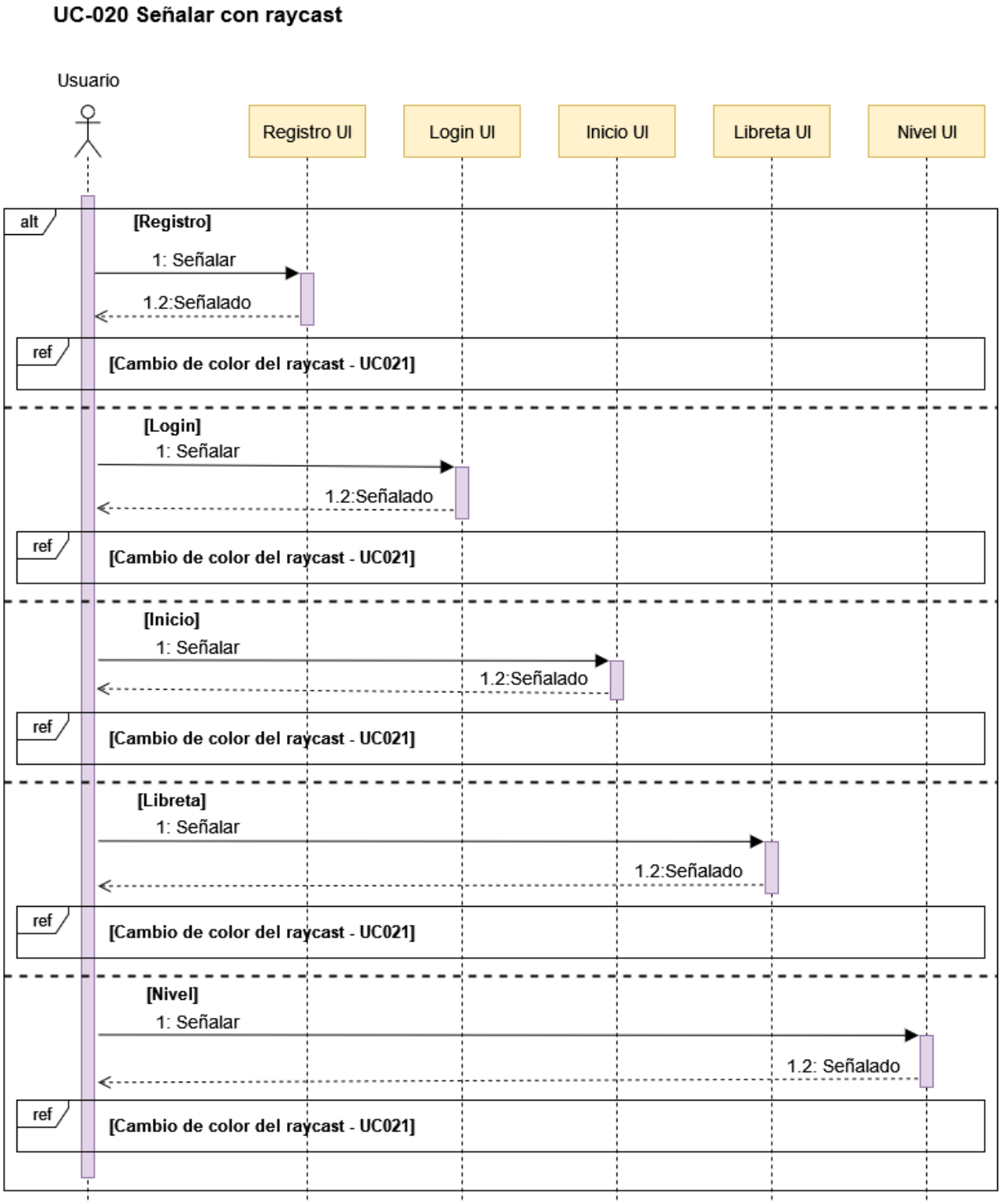

Figura 26: Diagrama de secuencia UC-020

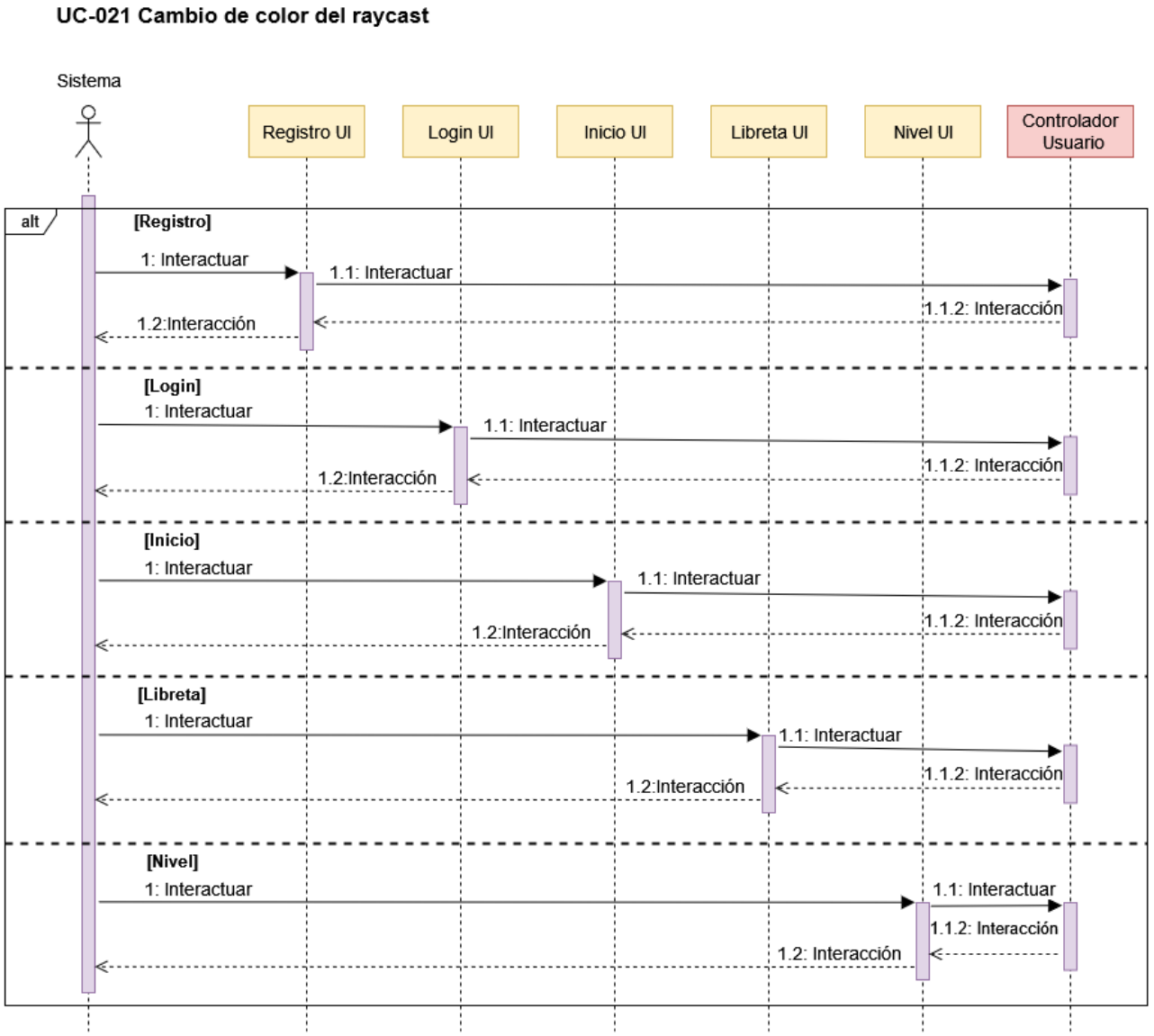

Figura 27: Diagrama de secuencia UC-021

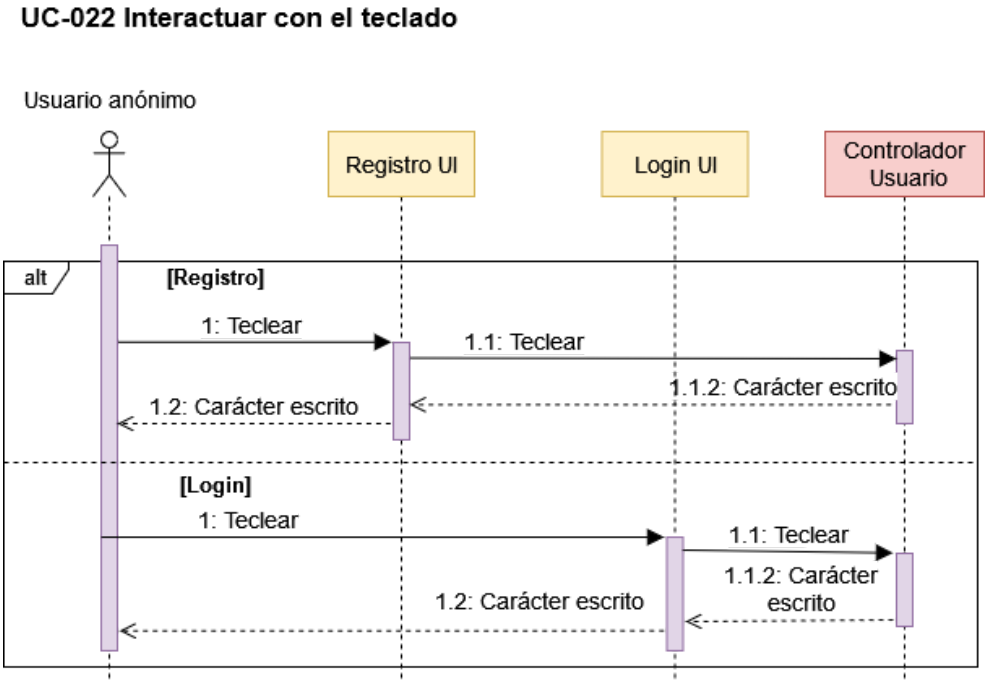

Figura 28: Diagrama de secuencia UC-022

3. Diseño base de datos

La base de datos que se ha utilizado es la proporcionada por *Firebase*, una base de datos NoSQL que se ordena por colecciones.

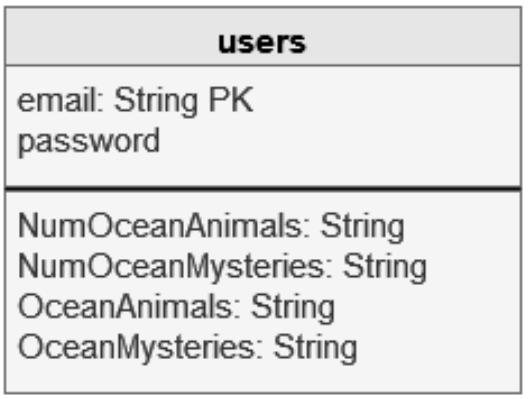

Figura 29: Diagrama de la base de datos

Se ha decidido poner todos los datos en una misma colección y no separados por animales y objetos ocultos puesto que cuanto más grande sea la colección más lento será el envío de datos, y los costes serán más grandes a medida que aumenta el envío de datos. *Firebase* proporciona un límite de uso de hasta 3000 usuarios activos por día.

4. Modelo de despliegue

A continuación, se detalla el diagrama de despliegue del sistema:

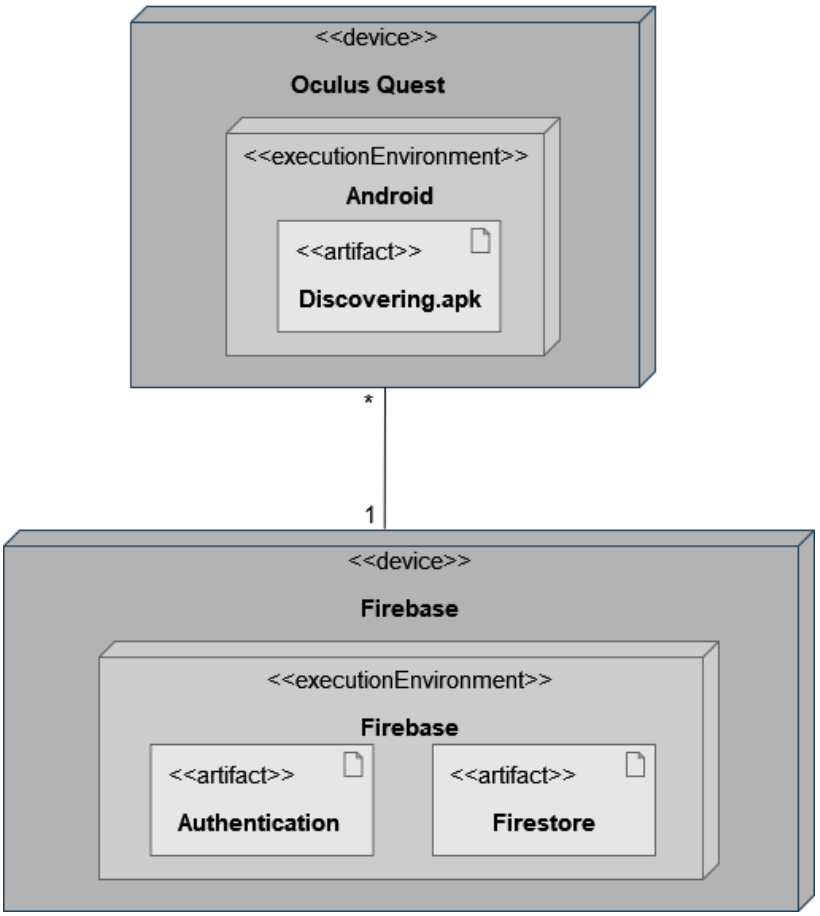

Figura 30: Diagrama de despliegue

Se observa que el sistema se compone de dos nodos:

- Oculus Quest, es el dispositivo empleado para la ejecución del *apk* en entorno Android. Será utilizado por los usuarios que ejecuten el videojuego.
- Firebase, es el servicio que nos ofrece autenticación y base de datos para el videojuego.

5. Referencias bibliográficas

[1] Moreno García, M. (2020). *UML. Unified Modeling Language*. Asignatura de Ingeniería de Software II.

[2] Moreno García, M., García Peñalvo, F. (2020). *Patrones*. Asignatura de Ingeniería de Software II.

[3] Rodríguez-Aragón, J., Zato Domínguez, C. (2020). *Diseño*. Asignatura de Ingeniería de Software II.

[4] Castro, R. Problema de herencia múltiple. *Universidad Latina*. https://slideplayer.es/slide/1724738/