Grupo 5: SOFTIX

“PÏNWÏ”

Integrantes del equipo:

Alba Sánchez Ibáñez	albasanchezibanez6@uma.es
Susana Gómez Álvarez	susana.gomez.2000@uma.es
Fernando Calvo Díaz	fernandouni@uma.es
Alejandro Caro Casado	alcaca172@uma.es
Francisco Javier Martín Luque	javimartinluque@uma.es
Jose Torres Postigo	jostorpos@uma.es
Miguel Moya Castillo	mcmiguel@uma.es
Alejandro Ramos Peña	aleeexrp02@uma.es
Javier Carmona Gálvez	javicg@uma.es

Enlace del repositorio:

https://github.com/Mikemece/g5-SOFTIX\

ÍNDICE

1. Introducción 3
2. Roles 3
3. Gestión de Riesgo 4
4. Planificación 5
5. Requisitos 6-11
6. Casos de uso 12-15
7. Modelos de dominio 16
8. Diagramas de secuencia 17-23
9. Pruebas Jest 24-30
10. Herramientas Software usadas 31

1. Introducción

Siguiendo la temática de crear un aplicación de entretenimiento no adictivo en línea**,** vamos a realizar un juego en el que encontraremos una mascota virtual, Pïnwï, a la cual podemos cuidar, evolucionar y comprarle objetos con monedas canjeables que se consiguen a través de un minijuego de responder preguntas de cultura general.

2. Roles

Miguel Moya Castillo	PRODUCT OWNER	DISEÑADOR
Jose Torres Postigo	DESARROLLADOR	SCRUM MASTER
Susana Gómez Álvarez	ANALISTA	TESTER
Javier Carmona Gálvez	DESARROLLADOR	SCRUM MASTER
Alejandro Caro Casado	TESTER	ANALISTA
Francisco Javier Martín Luque	TESTER	DESARROLLADOR
Fernando Calvo Díaz	DISEÑADOR	ANALISTA
Alba Sánchez Ibáñez	DISEÑADOR	PRODUCT OWNER
Alejandro Ramos Peña	DESARROLLADOR	SCRUM MASTER

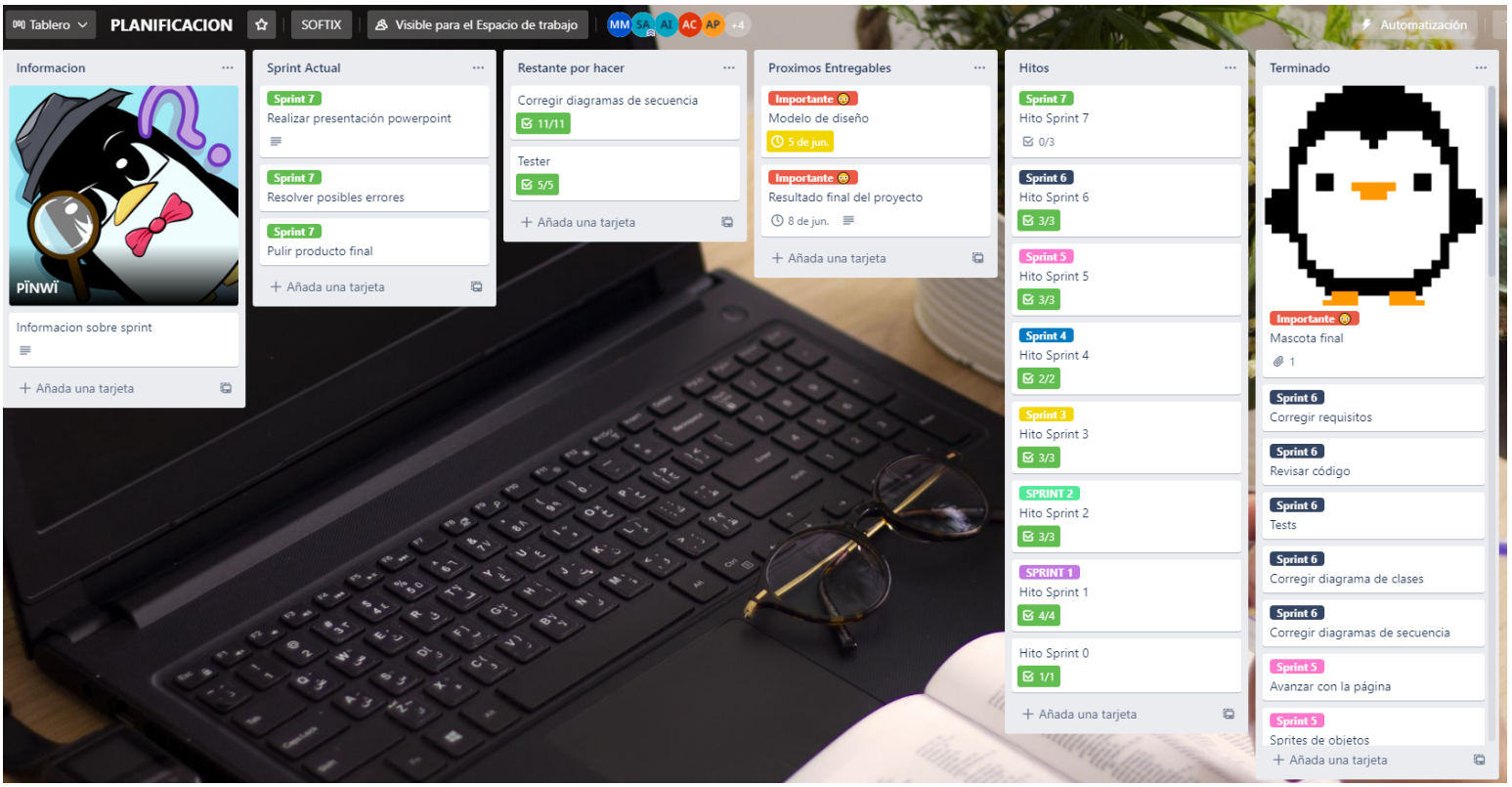
3. Gestión de Riesgo

Riesgo	Tipo	Descripción	Probabilidad	Efecto	Estrategia
Subestimación del tamaño	Proyecto y producto	El tamaño del sistema se ha subestimado	Moderada	Serio	Analizar bien el posible tiempo de realización de la tarea
Bajo desempeño	Personal	Los miembros no cumplen con las expectativas puesta en ellas	Alta	Serio	Considerar el trabajo pendiente actual y reorganizar conforme el potencial de cada uno.
Factor tiempo	Proyecto	Riesgo a que el proyecto requiera más de lo esperado	Moderado	Serio	Dedicarle más horas y replantear los estándares de calidad al igual que lo necesario para el proyecto
Inexperiencia	Proyecto	No tenemos idea de cómo realizar alguna tarea	Alta	Tolerable	Investigar en internet, acceder a los recursos de la universidad
Abandono de algún integrante	Proyecto	Un integrante abandona el proyecto	Poco probable	Catastrófico	Reasignar el trabajo y replanteamiento de las ideas necesarias.
Retraso en las especificaciones	Proyecto y producto	Retraso en las especificaciones	Moderada	Serio	Saber cuánto tiempo dedicar a cada tarea desde el inicio e ir documentando el progreso
Cambio repentino de ideas	Proyecto y producto	Empezar a dudar de el proyecto que estamos haciendo	Poco probable	Tolerable	Tener claro lo que se va a hacer antes de empezar

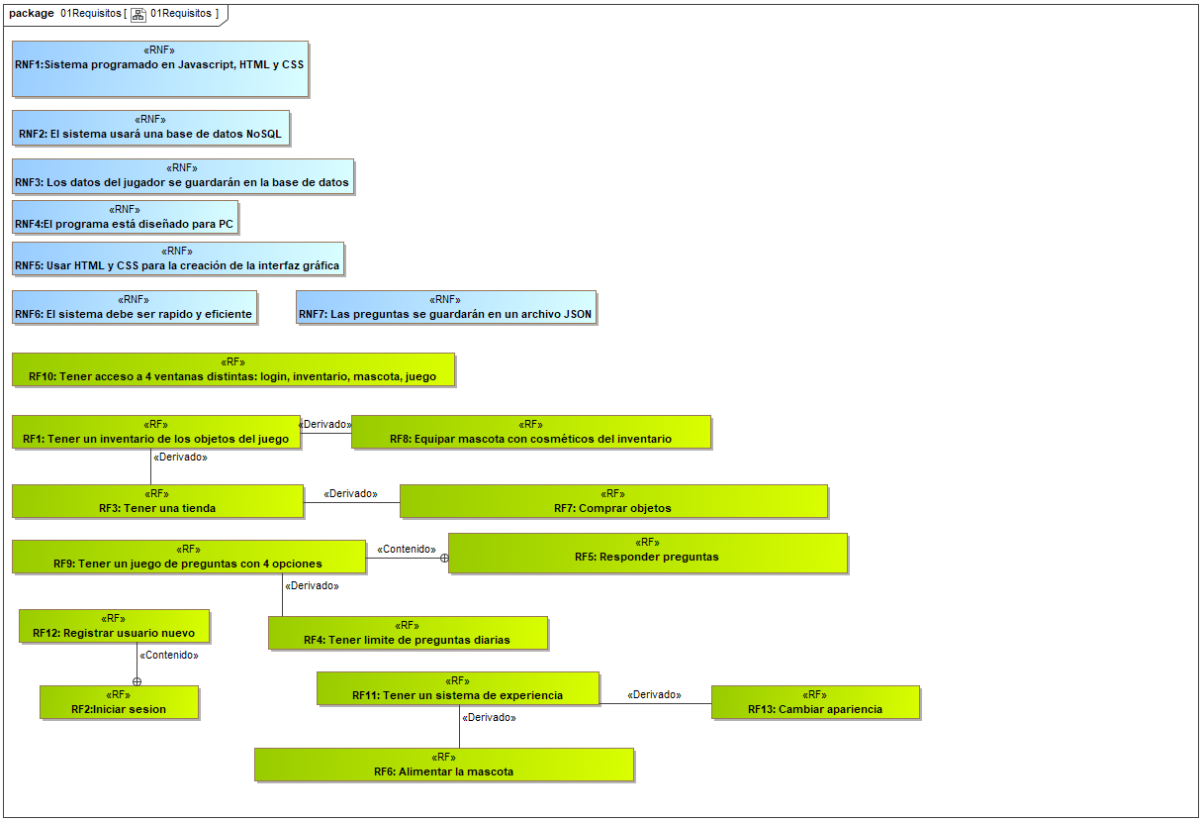
4. Planificación

Hemos elegido la metodología SCRUM, porque es la que nos permite un manejo ágil de nuestras tareas y una organización moldeable a nuestras necesidades y las del proyecto. Al ser un grupo pequeño, scrum nos permite mejorar la productividad al poder probar el producto sin pasar por todo el ciclo de producción.

- Tenemos el power up “votar”, que nos permite ver de manera rápida qué idea es más valorada.



5. Requisitos



Funcionales:

1. **Como** usuario

   Quiero acceder al inventario

   Para equipar cosméticos a mi mascota.

   Pruebas de aceptación:

   - La mascota no tiene ningún cosmético y se le equipa un solo cosmético.
   - La mascota tiene un cosmético equipado y se le equipa otro que sea compatible con el ya equipado.
   - La mascota tiene uno o más cosmético/s equipado/s y se le equipa un cosmético que es del mismo tipo que uno de los que tiene equipados, se quita el/los cosmético/s incompatibles y se equipa el cosmético.
   - El usuario intenta equipar un cosmético que no tiene comprado y le sale un aviso de que tiene que comprarlo antes de equiparlo.

2. **Como** usuario
   Quiero iniciar sesión
   Para poder acceder al juego.
   Pruebas de aceptación:
   - El usuario no tiene cuenta y tiene que registrar una nueva cuenta en la aplicación.
   - El usuario inicia sesión correctamente en su cuenta.
   - El usuario introduce unas credenciales incorrectas y le sale un aviso de credenciales incorrectas.

3. **Como** usuario
   Quiero tener una tienda
   Para poder comprar cosméticos a mi mascota.
   Pruebas de aceptación:
   - El usuario no tiene dinero e intenta comprar un cosmético y le sale un aviso de que no tiene dinero para comprar el cosmético.
   - El usuario tiene el dinero suficiente para comprar un cosmético, lo compra y se le resta el dinero que cuesta el cosmético en cuestión.

4. **Como** desarrollador
   Quiero tener un límite de preguntas diario
   Para que mi juego no sea adictivo.
   Pruebas de aceptación:
   - El usuario responde a las 4 preguntas si no ha jugado durante el día.
   - El usuario, una vez respondido a las 4 preguntas, intenta volver a jugar pero no puede por el límite establecido.

5. **Como** usuario
   Quiero responder preguntas
   Para conseguir monedas.
   Pruebas de aceptación:
   - El usuario no acierta ninguna pregunta y no consigue ninguna moneda.
   - El usuario acierta alguna pregunta y se le da unas monedas.

- El usuario acierta todas las preguntas y recibe el máximo de monedas posibles más un bonus por acertar todas las preguntas.

6. **Como** usuario
   Quiero poder alimentar a mi mascota
   Para ganar experiencia y subir de nivel a mi mascota
   Pruebas de aceptación:
   - El usuario tiene comida en su inventario y alimenta su mascota, subiendo su experiencia.
   - El usuario alimenta su mascota y sube experiencia suficiente para que pase al siguiente nivel.

7. **Como** usuario
   Quiero comprar
   Para poder equiparlos a mi mascota.
   Pruebas de aceptación:
   - El usuario intenta comprar un objeto y si tiene dinero suficiente se añade el objeto al inventario.
   - El usuario intenta comprar un objeto y si no tiene dinero suficiente no puede comprarlo.

8. **Como** usuario
   Quiero equipar cosméticos a mi mascota
   Para que mi mascota sea más personalizable.
   Pruebas de aceptación:
   - El usuario intenta equipar un objeto a la mascota y, si lo tiene comprado, se lo equipa.
   - El usuario intenta equipar un objeto a la mascota y, si no lo tiene comprado, no puede equiparlo.

9. **Como** desarrollador
   Quiero que las preguntas tengan 4 opciones de respuesta
   Para que el usuario elija la respuesta que crea correcta
   Pruebas de aceptación:

- El usuario entra en el juego de preguntas y escoge una de las 4 respuestas.

10. **Como** usuario
    Quiero que se actualicen las preguntas
    Para que el juego sea más entretenido.
    Pruebas de aceptación:
    - El usuario responde preguntas diferentes cada vez que entre
    - Las preguntas serán diferentes entre las escogidas el mismo dia

11. **Como** usuario
    Quiero que la mascota tenga un sistema de experiencia y niveles
    Para hacer tangible mi progreso.
    Pruebas de aceptación:
    - El usuario alimenta la mascota y la experiencia de esta aumenta
    - La mascota cada 10 de experiencia sube 1 nivel

12. **Como** usuario
    Quiero registrarme en la aplicación
    Para poder comenzar a jugar.
    Pruebas de aceptación:
    - Un nuevo usuario se registra en la aplicación y se guardan sus credenciales.
    - El nuevo usuario comienza una partida y puede jugar.

13. **Como** usuario
    Quiero que mi mascota cambie de apariencia
    Para tener sensación de avance en el juego
    Pruebas de aceptación:
    - El usuario gana experiencia acertando preguntas.
    - Cuando la mascota llega a cierto nivel, su apariencia cambia.

No funcionales:

1. **Como** desarrollador
   Quiero programar la lógica de la aplicación en Javascript
   Porque es la manera más sencilla y accesible de gestionar la lógica de nuestra aplicación web.

   Prueba de aceptación:

   - Que no haya fallos de compilación.

2. **Como** desarrollador
   Quiero usar una base de datos no relacional
   Porque pensamos que es una base de datos fácilmente accesible.
   Pruebas de aceptación:
   - La primera vez que el usuario accede a la aplicación tiene que registrarse e insertar un nombre de usuario y una contraseña, la cual se añade a una entidad de la base de datos.
   - Hay una entidad que lleva la cuenta del inventario de cada usuario y si tiene los cosméticos desbloqueados o no.
   - Hay una entidad que lleva las cuentas de usuario que hay en la aplicación.
   - Hay una entidad que tiene almacenadas todas las preguntas que se le irán haciendo al usuario diariamente.

3. **Como** desarrollador
   Quiero que tenga 2 bases de datos: Jugador (Inventario y Tienda) y Juego de preguntas
   Para poder almacenar información de manera eficiente

   Pruebas de aceptación:

   - Cada vez que el usuario acceda, mantendrá todo lo que tenía previamente.
   - Cada día, las preguntas tendrán que actualizarse siendo diferentes siempre.

4. **Como** desarrollador
   Quiero usar HTML y CSS
   Para la creación de la página web e interfaz.

   Pruebas de aceptación:

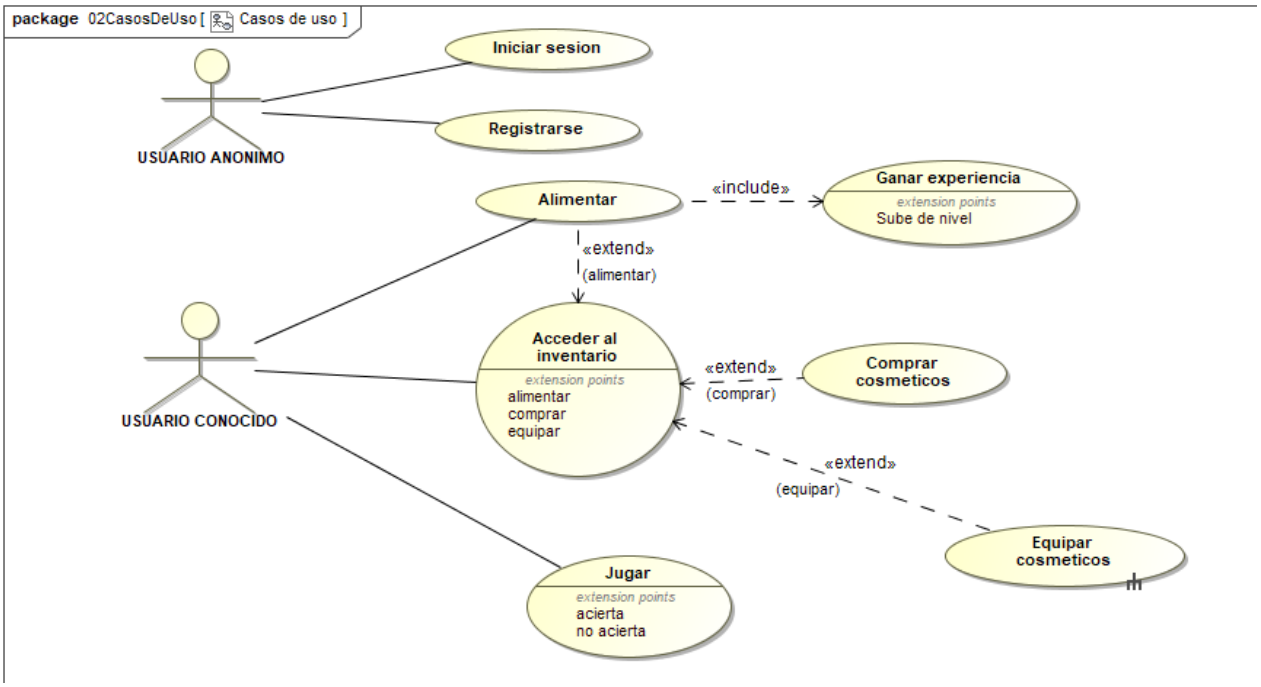
   - El código escrito en el HTML se muestra de forma correcta en la página web.
   - El código aplicado en CSS muestra el diseño deseado en la página web.

5. **Como** desarrollador
   Quiero que toda funcionalidad del sistema responda al usuario en el menor tiempo posible
   Para un mejor disfrute de la aplicación.
   Pruebas de aceptación:

   - El usuario entra en el juego y pulsa un botón cualquiera del juego, si es eficiente, responde el juego en un tiempo prudencial.
   - El usuario entra en el juego y pulsa un botón cualquiera del juego, si no es eficiente, el juego responde tarde.
   - El usuario entra en el juego y pulsa un botón cualquiera del juego, si no es nada eficiente, el juego no responde.

6. **Como** desarrollador
   Quiero diseñar la aplicación para pc
   Para ofrecersela al público objetivo
   Pruebas de aceptación:
   - El usuario utiliza la aplicación solamente y exclusivamente en un navegador web de ordenador.
7. **Como** desarrollador
   Quiero que las preguntas se guarden en un archivo JSON
   Para facilitar su guardado y acceso
   Pruebas de aceptación:
   - La aplicación intenta acceder a las preguntas mediante el archivo.

6. Casos de uso



1.- Registro

- Identificador único: CU1 Registrar un usuario.
- Contexto de uso: Cuando un usuario acceda a la página por primera vez, pueda registrarse.
- Precondiciones y activación:
    - La página ha de estar disponible.
    - El usuario tiene que pensar un nombre y contraseña.
    - El usuario ha de disponer de un aparato electrónico que tenga conexión a Internet.
- Garantías de éxito: El usuario queda registrado en la base de datos y ya puede acceder a la página de juego.
- Escenario principal:
    1. El usuario entra a la página.
    2. Se le muestran dos opciones, registrarse e iniciar sesión.
    3. Se registra con algún nombre a su elección y una contraseña. Automáticamente, se almacena en la base de datos.

- Escenarios alternativos:
    1. El usuario ya tiene cuenta y no puede registrarse, tiene que iniciar sesión.
    2. El usuario se registra pero elige un nombre o contraseña que no cumple las condiciones.

2.-Iniciar sesión

- Identificador único: CU2 Iniciar sesión.
- Contexto de uso: Cuando el usuario acceda a la aplicación y ya tenga una cuenta, acceda a ésta.
- Precondiciones y activación:
    - La página ha de estar disponible.
    - El usuario tiene que introducir su nombre y contraseña correctamente.
    - El usuario ha de disponer de un aparato electrónico que tenga conexión a Internet.
- Garantía de uso: El usuario accede a la aplicación con su cuenta que había creado con anterioridad.
- Escenario principal:
    1. El usuario entra a la página.
    2. Se le muestran dos opciones, registrarse e iniciar sesión.
    3. Inicia sesión con su cuenta.
- Escenario alternativos:
    1. El usuario se equivoca al introducir sus datos.
    2. Salta un mensaje de error, instando que lo intente de nuevo.

3.- Alimentar

- Identificador único: CU3 Alimentar la mascota.
- Contexto de uso: El usuario quiere alimentar a la mascota para obtener experiencia.
- Precondiciones y activación:
    - El usuario ha debido de iniciar sesión correctamente.
    - El usuario ha de disponer de monedas suficientes para comprarla.
- Garantía de uso: La mascota es alimentada.

- Escenario principal:
    1. El usuario inicia sesión.
    2. El usuario compra la comida.
    3. Se la da a la mascota.
    4. Gana experiencia la mascota.
- Escenarios alternativos:
    1. El usuario va a alimentar la mascota, pero no tiene comida.
    2. Comprar comida, y si tampoco tiene puntos/dinero, el usuario tiene que jugar para ganarlos.
    3. El usuario ya ha jugado al juego diario, tiene que esperar al día siguiente.

4.- Acceder al inventario

- Identificador único: CU4 Acceder al inventario.
- Contexto de uso: El usuario quiere acceder al inventario.
- Precondiciones y activación:
    - El usuario ha debido de iniciar sesión correctamente.
- Garantía de uso: El usuario accede al inventario.
- Escenario principal:
    1. El usuario inicia sesión.
    2. El usuario accede al apartado de inventario.
    3. A partir de ahí, el usuario puede comprar cosméticos para la personalización de la mascota.
- Escenarios alternativos:
    1. Error al acceder.

5.- Jugar

- Identificador único: CU5 Jugar a los juegos disponibles
- Contexto de uso: El usuario quiere jugar.
- Precondiciones y activación:
    - El usuario ha debido de iniciar sesión correctamente
    - El usuario no ha jugado previamente en el mismo día.
- Garantía de uso: El usuario juega.
- Escenario principal:

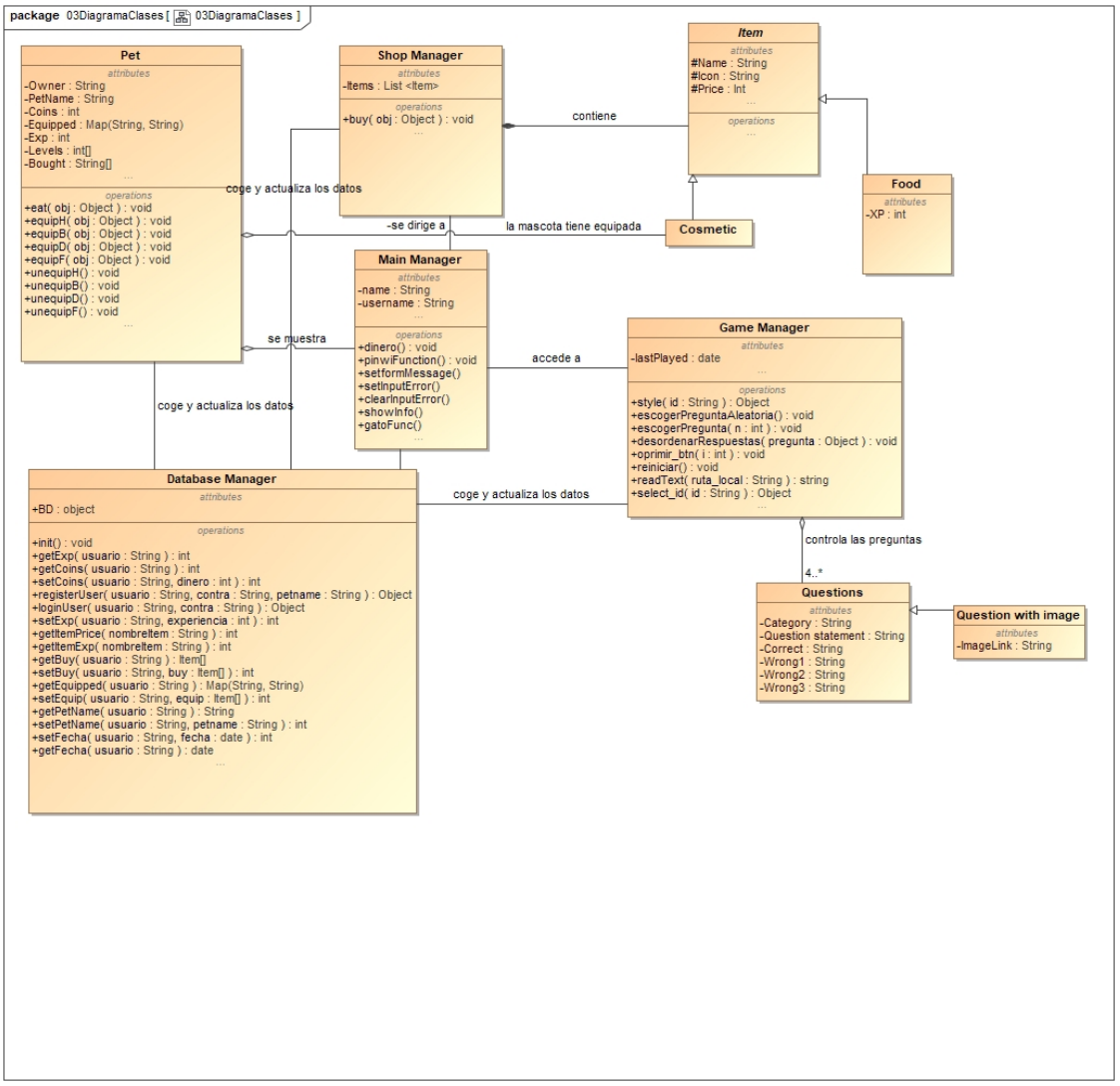
1. El usuario inicia sesión.
2. El usuario accede al apartado de juegos.
3. El usuario accede al juego que desee.
4. Acierta y obtiene monedas.

- Escenarios alternativos:
    1. El/Los juego/s no están disponibles por su previo uso.
    2. El jugador no acierta la pregunta y no recibe monedas.

6.- Comprar cosméticos

- Identificador único: CU7 El usuario compra un cosmético.
- Contexto de uso: El usuario quiere comprar un cosmético desde el inventario.
- Precondiciones y activación:
    El usuario tiene que haber iniciado sesión.
    El usuario tiene que estar en el inventario.
- Garantía de uso: El usuario consigue un cosmético nuevo.
- Escenario principal:
    1. El usuario da click en un cosmético que no tiene.
    2. Se le resta el dinero que cueste el cosmético.
    3. El cosmético se desbloquea para su uso.
- Escenarios alternativos:
    1. El usuario da click en un cosmético que no tiene.
    2. No tiene suficiente dinero para el cosmético.
    3. No se compra el cosmético.

7. Modelos de dominio



Main manager: encargado de la interfaz gráfica de la pantalla de inicio de sesión y la principal del juego.

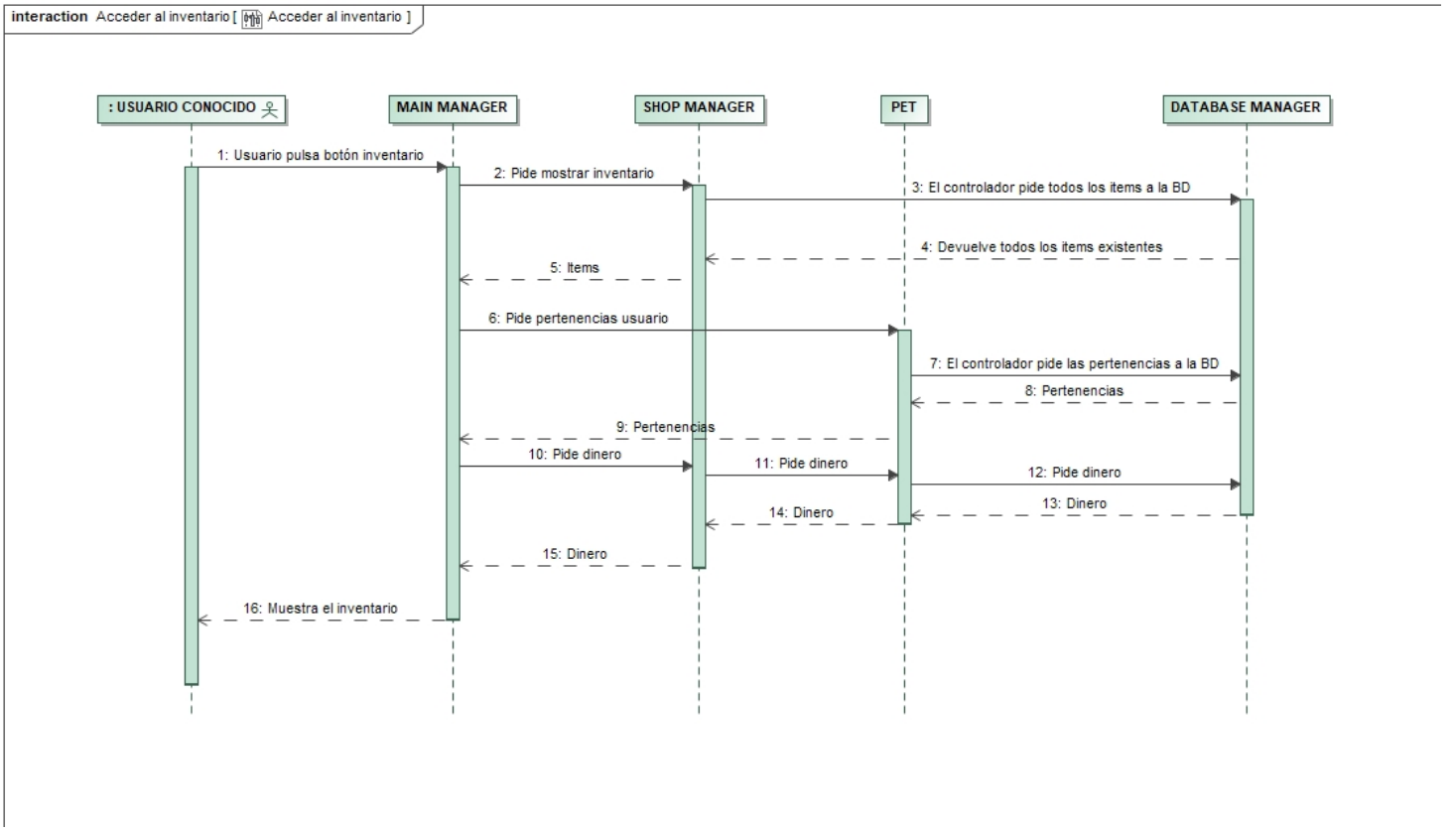
Game Manager: está a cargo de la interfaz gráfica del juego de preguntas y la lógica detrás del mismo.

Shop Manager: es el encargado de la interfaz gráfica de la pantalla de inventario/tienda, al igual que incorpora la lógica de la misma.

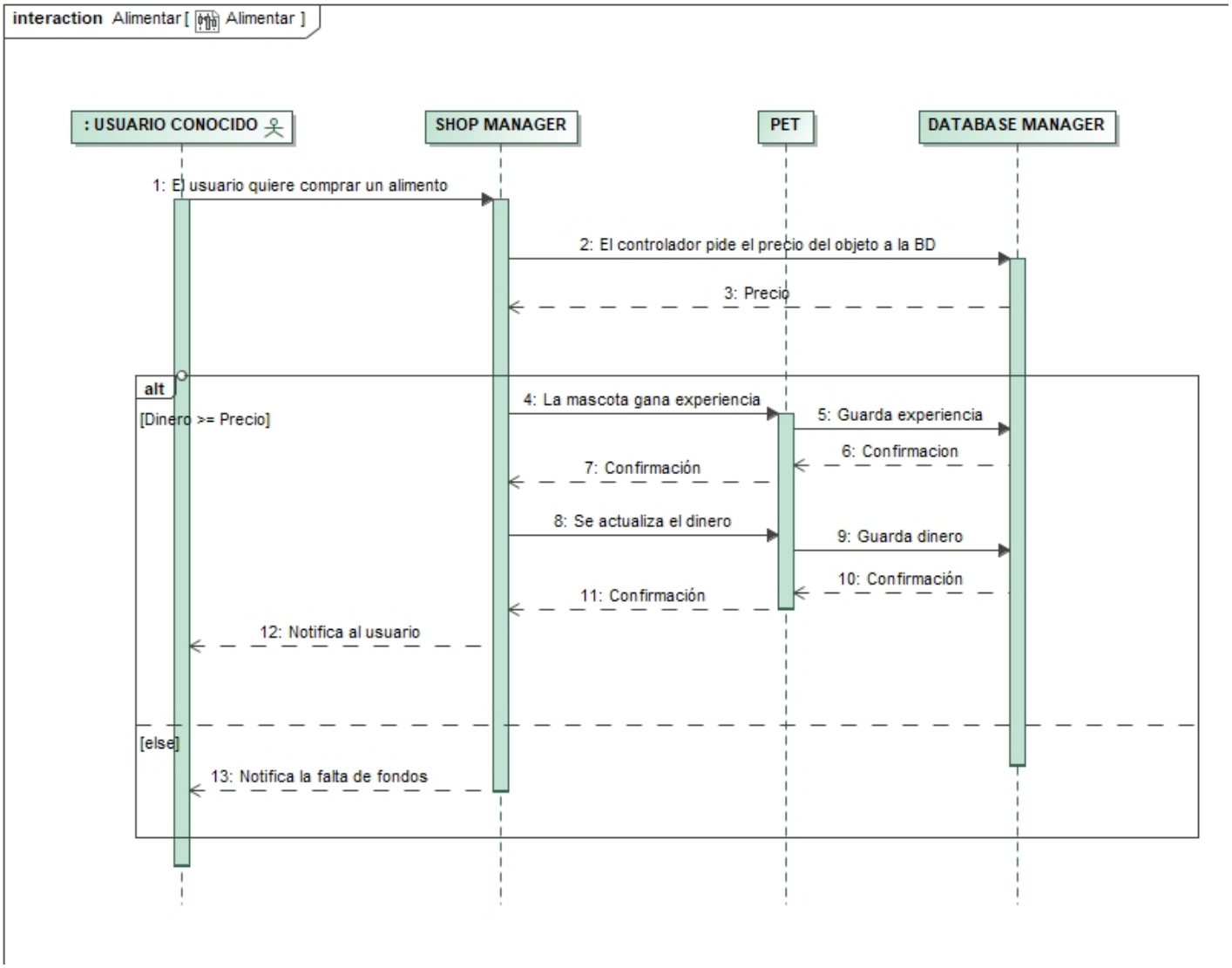
Database Manager: es el operador de toda transacción entre la aplicación y la base de datos.

8. Diagramas de secuencia

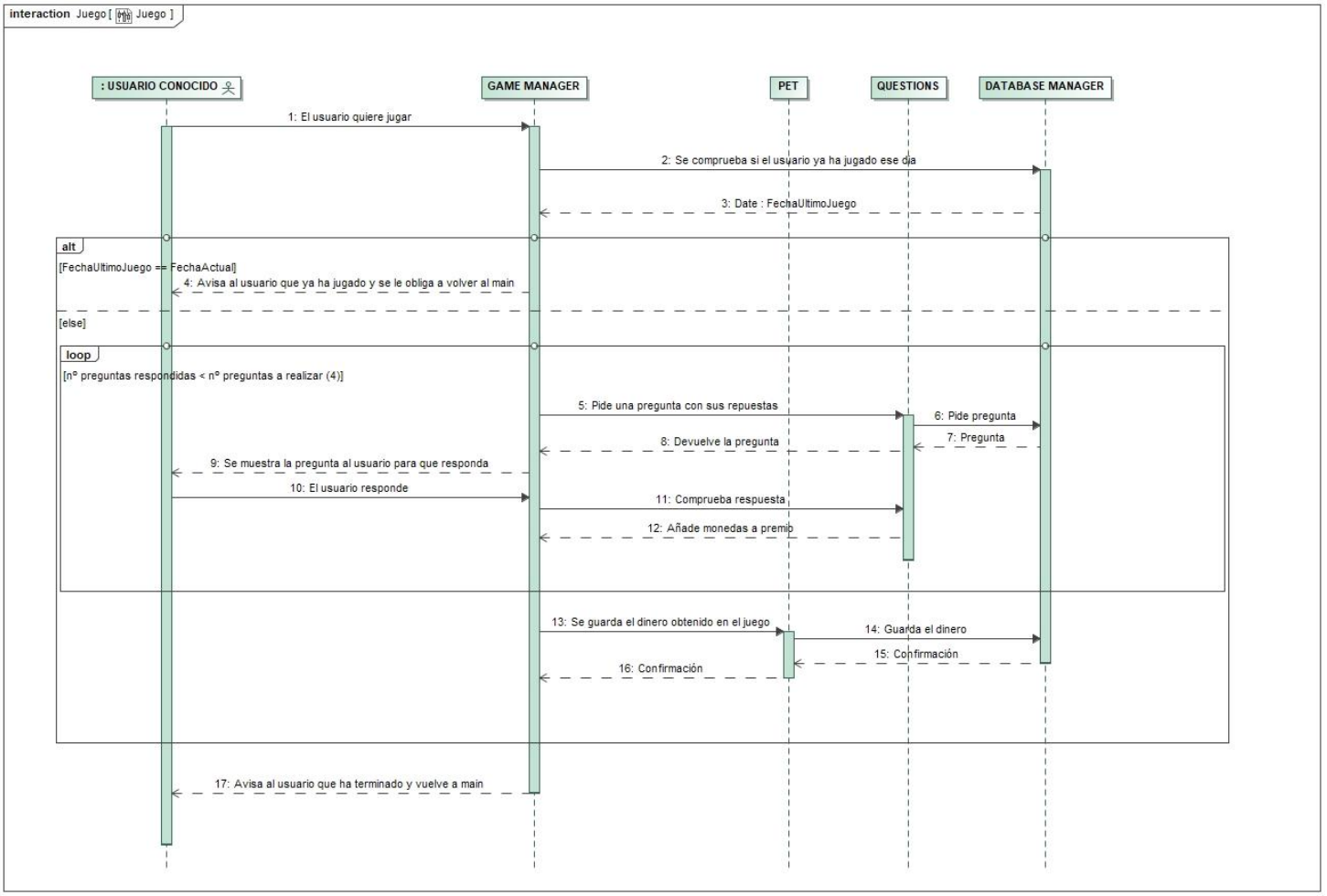
- *Acceder al inventario:* Cuando el usuario quiera acceder al inventario, el controlador pide los ítems a la base de datos. Para ello, el Database Manager obtendrá los ítems que posee el usuario, se los devolverá al Shop Manager y este se los mostrará al usuario.



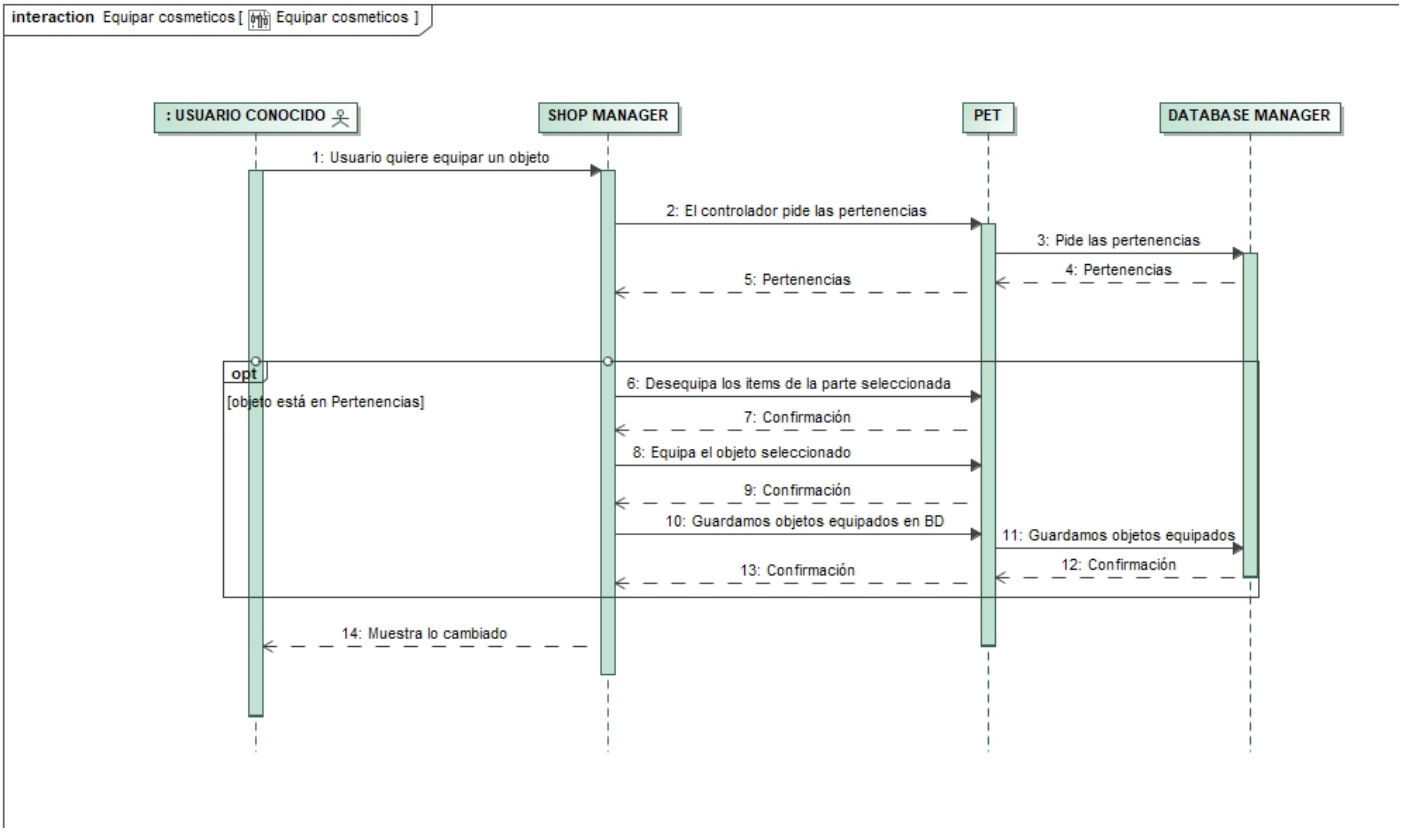
- *Alimentar:* Cuando el usuario quiera alimentar a la mascota, primero accede al inventario para luego intentar comprar la comida: si tiene dinero suficiente la mascota será alimentada y ganará experiencia, y si no tiene dinero suficiente se notificará de esto.



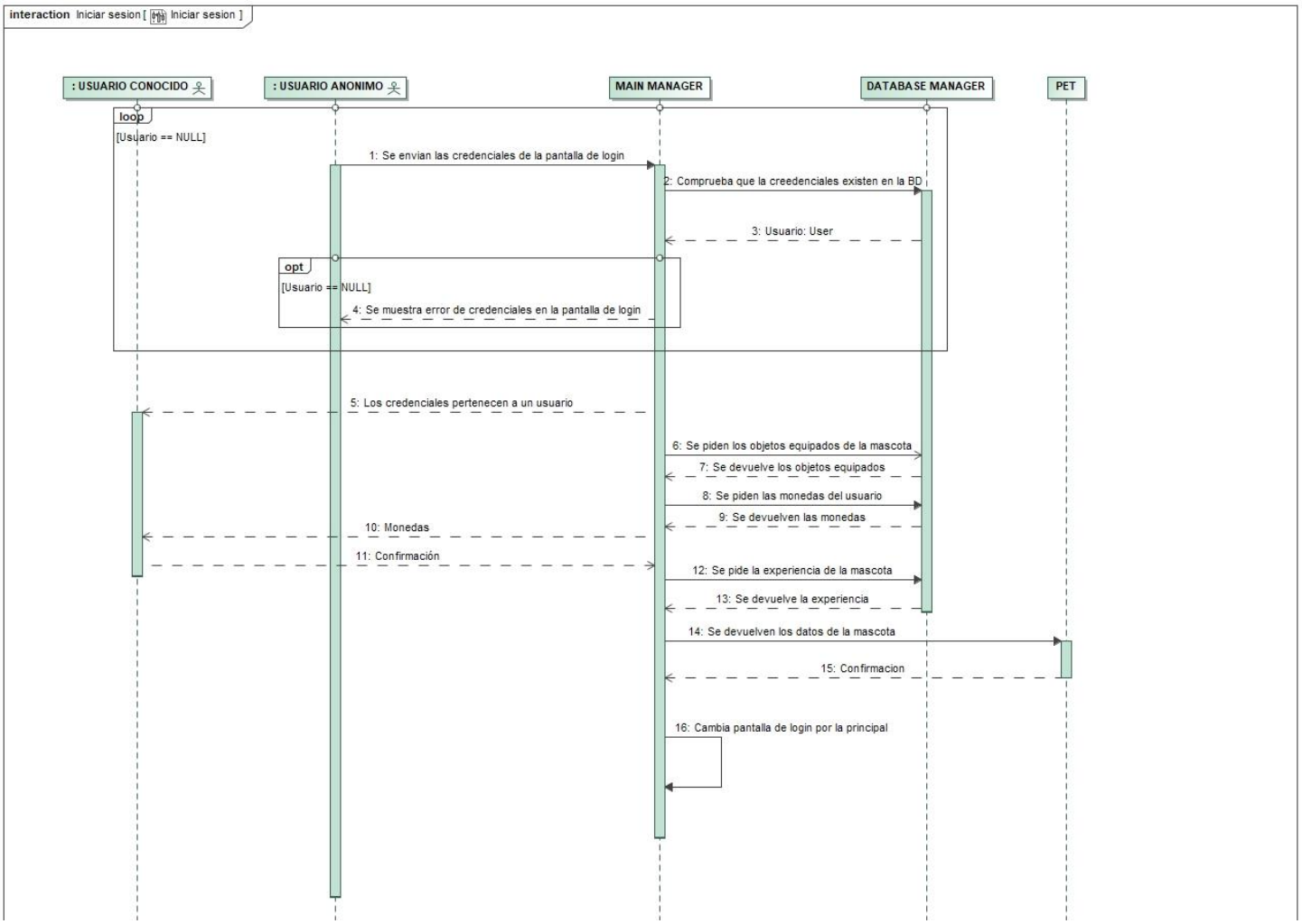
- *Jugar*: Cuando el usuario vaya a jugar, el controlador pedirá las preguntas a la base de datos para que el usuario pueda verla y contestar. Mientras el usuario está con la pregunta, envía una respuesta y recibe monedas si está bien contestada o recibe un mensaje de que está mal contestada y que siga intentándolo.



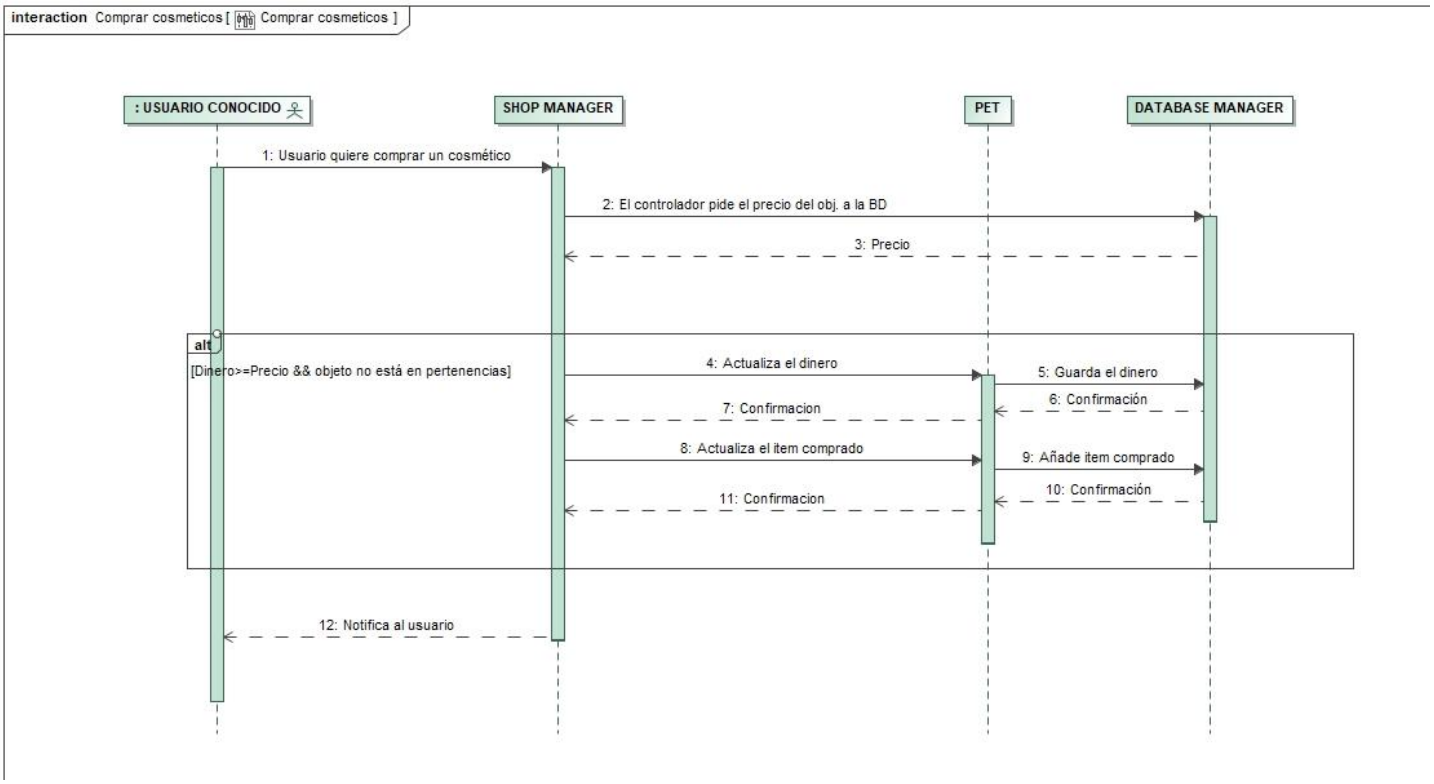
- *Equipar cosméticos:* Cuando el usuario quiera equipar cosméticos a la mascota, el controlador pide los ítems a la base de datos. Para ello, el Database Manager obtendrá los ítems que hayan sido comprados por el usuario, se los devolverá al Shop Manager y este se los equipará a la mascota.



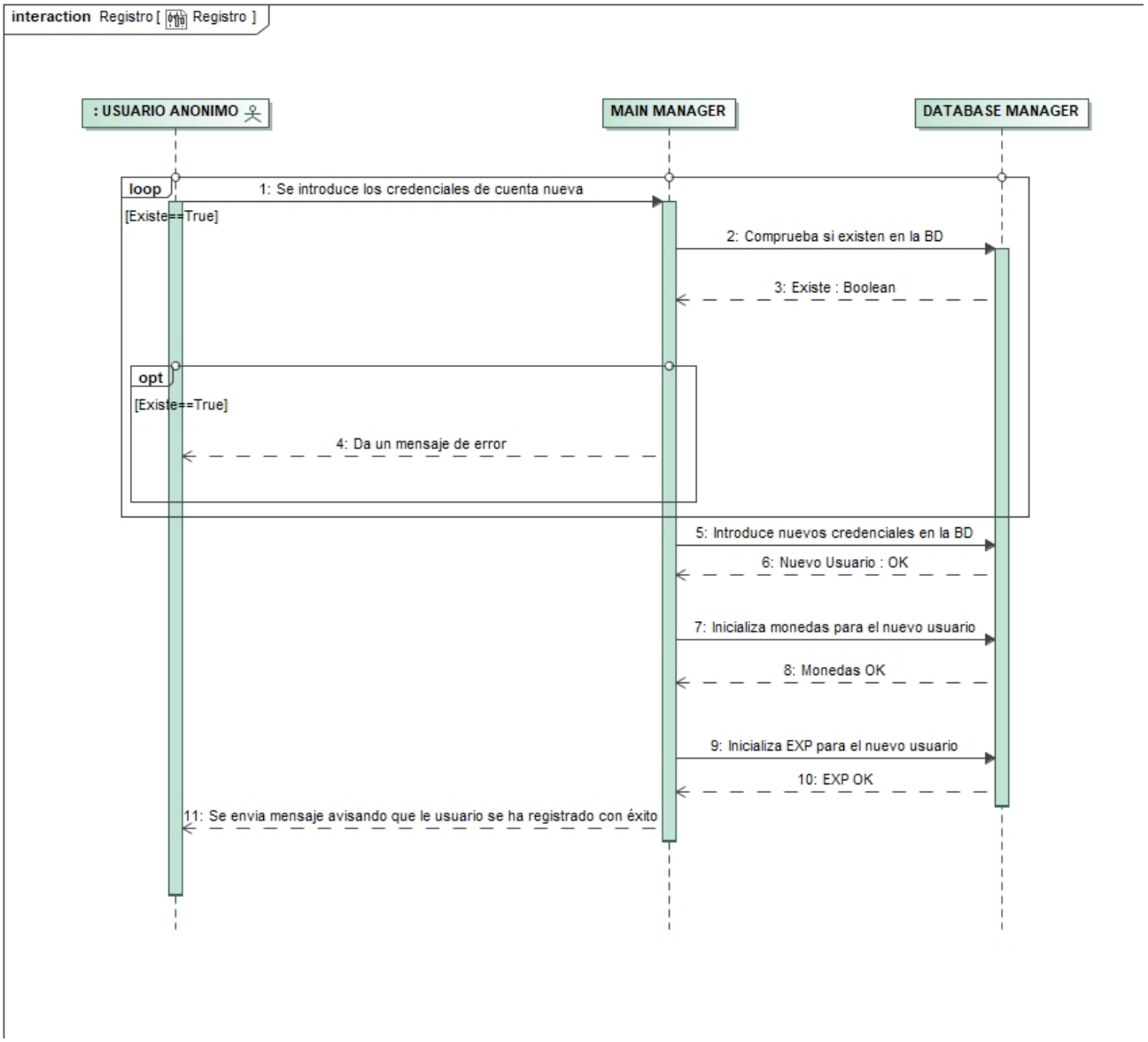
- *Iniciar sesión:* Cuando un usuario anónimo introduce sus credenciales en la pantalla de login,el controlador comprueba que sus datos existan en la base de datos, y en ese caso, se crea el objeto usuario con todo lo que tenga guardado a su nombre en la base de datos. Si no, da un mensaje de error.



- *Comprar cosméticos:* Cuando el usuario quiere comprar cosméticos para la mascota, el controlador pide el item seleccionado previamente por el usuario. El Database Manager obtendrá el item que haya sido seleccionado, que éste le enviará una respuesta con el item, a cambio de la pérdida de dinero de dicho item.



- *Registro:* Cuando un usuario anónimo quiere registrarse, la Base de Datos comprueba si ya está registrado o no. En el primer caso guarda la información, en el segundo dará un mensaje de error. Cuando acabe el registro el usuario recibirá un mensaje de confirmación.



9. Pruebas Jest

Las pruebas que hemos realizado están hechas en Jest debido a que nuestro proyecto está gran parte programado en javascript. Además, cada código de test nos permite explicar de qué trata cada uno de ellos.

A continuación les mostramos los códigos de dichas pruebas.

- Pruebas del juego de las preguntas: juego.jsconst {
  JS_EXT_TO_TREAT_AS_ESM } = require("ts-jest")
  const {escogerPreguntaAleatoria, db, escogerPregunta,
  desordenarRespuestas, readText, style, select_id, reiniciar,
  oprimir_btn} = require("./juego")

```
/*
test('Escoger una pregunta con un índice no existente', async () => {

    expect(await juego.escogerPregunta(8).toThrow());
})

test('Oprimir un boton no existente', async () => {
    expect(await oprimir_btn(8).toBe(-1))
})
*/

test('Seleccionar una id no existente', async () => {

    expect(await select_id("abduscan")).toBeNull
})


/*
test('Seleccionar un style no existente', async () => {
    expect(await style("abduscan").toBe(-1))
})
*/


/*test('Leer una ruta para escoger las preguntas no existente', async
() => {
    expect(await readText('./preguntas.json').toBeNull())
})*/
```

- Pruebas para el inventario de la mascota: inventory.js

```
//const inventory = require("./inventory.js");
```

```
// import{DBManager} from "./DBManager.js";
// import {buy,eat} from "./inventory.js";

const { JS_EXT_TO_TREAT_AS_ESM } = require("ts-jest")
const {buy, eat, db, equipH, equipB, equipD, equipF, unequipH, unequipB,
unequipF,unequipD} = require("./inventory")
let database = db

beforeAll(async ()=>{
 database.init()
 window.sessionStorage.setItem('user', 'Prueba')
 database.setCoins('Prueba', 999)
})

test("buy añade objeto", async () => {

 let moneyOld = await database.getCoins('Prueba')
   await buy({
     id:'gorro',
     className:'aaaaaaaN',
     classList: {
       arr: [],
       add: function(variable)
       {
         this.arr.push(variable);
       },
       remove: function(variable)
       {
         this.arr.pop(variable);
       }
     }
   })
   let itemPrice = await database.getItemPrice('gorro')
   expect(await database.getBuy('Prueba')).toContain('gorro')
   expect(await database.getCoins('Prueba')).toBe(moneyOld-itemPrice)
   console.log("buy ejecutado correctamente")

 })

 test("eat compra comida y alimenta a la mascota",async () => {
   let moneyOld = await database.getCoins('Prueba');
   let expOld = await database.getExp('Prueba');
   await eat({
     id:'burger',
     className:'aaaaaaaN',
     classList: {
       arr: [],
```

```
        add: function(variable)
        {
          this.arr.push(variable);
        },
        remove: function(variable)
        {
          this.arr.pop(variable);
        }
      }
    })
    let moneyNew = await database.getCoins('Prueba');
    let expNew = await database.getExp('Prueba');
    expect(moneyOld-moneyNew).toEqual(await database.getItemPrice('burger'));
    let varia = expNew-expOld;
    expect(varia).toEqual(await database.getItemExp('burger'));

    console.log("eat ejecutado correctamente");
  })
```

- Pruebas para el Registro y Logeo de un usuario: index.js

```
const {setFormMessage, clearInputError, setInputError} = require("./index")

global.confirm = () => true

test('QuerySelector devuelve null cuando se le proporciona objeto vacío',
async()=>{
    //expect(index.loginUsername.value).toBe(index.signupUsername.value);
    expect(setFormMessage({
        querySelector: function(varia)
        {return {
            textContent: "",
            classList: {
                arr: [],
                add: function(variable)
                {
                  this.arr.push(variable);
                },
                remove: function(variable)
                {
                  this.arr.pop(variable);
                }
              }
        }}
    })).toBe(undefined)
});
```

```
test('ClearInputError devuelve null cuando se le proporciona objeto vacío',
async()=>{
    expect(clearInputError({
        classList: {
            arr: [],
            add: function(variable)
            {
              this.arr.push(variable);
            },
            remove: function(variable)
            {
              this.arr.pop(variable);
            }
          },
          parentElement:
          {
              querySelector: function(varia)
              {
                  return 0
              }
          }
    })).toBe(undefined);
});

test('SetInputError devuelve null cuando se le proporciona objeto vacío',
async()=>{
    expect(setInputError({
        classList: {
            arr: [],
            add: function(variable)
            {
              this.arr.push(variable);
            },
            remove: function(variable)
            {
              this.arr.pop(variable);
            }
          },
          parentElement:
          {
              querySelector: function(varia)
              {
                  return 0
              }
          }
    })).toBe(undefined);
});
```

- Pruebas para la Base de Datos:

```
/**
* @jest-environment node
*/
//Hemos creado un usuario llamado 'Prueba' para poder probar todos los
métodos de la base de datos

import  DBManager  from './DBManager.js';
//const DBManager = require("./DBManager");
let db ;

beforeEach(async ()=>{
   db = new DBManager() ;
   db.init()
})


test('prueba pedir coins de usuario que no existe', async () => {
   expect(await db.getCoins("Usuario no existente")).toBe(-1)
})

test('Pedir monedas de el usuario Prueba', async () =>{
   expect(await db.getCoins("Prueba")).toBe(20)
})

test('Establecer monedas de un usuario no existente', async () => {
   expect(await db.setCoins("Usuario no existente")).toBe(-1)
})

test('Registrar un usuario con valores incorrectos en la base de datos',
async () => {
   expect(await db.registerUser("", "12345", "Pinwi")).toBe(-1)
})

test('Logear un usuario no existente en la base de datos', async () => {
   expect(await db.loginUser("Usuario no existente", "12345")).toBe(-1)
})

test('Logear un usuario ya existente, con contraseña incorrecta', async ()
=> {
   expect(await db.loginUser("alex", "contraseñaincorrecta")).toBe(0)
})

test('Recibir experiencia del usuario Prueba', async () => {
   expect(await db.getExp("Prueba")).toBe(8)
```

```
})

test('Recibir experiencia de un usuario no existente', async () => {
   expect(await db.getExp("Usuario no existente")).toBe(-1)
})

test('Establecer experiencia de un usuario no existente', async () => {
   expect(await db.setExp("Usuario no existente")).toBe(-1)
})

test('Obtener objetos equipados de un usuario no existente', async () => {
   expect(await db.getItemsEquipped("Usuario no existente")).toBe(-1) // No
se usa
})

test('Obtener precio del objeto chancla', async () => {
   expect(await db.getItemPrice("chancla")).toBe("1")
})

test('Obtener objetos comprados del usuario Prueba', async () => {
   expect(await db.getBuy("Prueba")).toStrictEqual(["chancla","betis"])
})

test('Obtener objetos comprados de un usuario no existente', async () => {
   expect(await db.getBuy("Usuario no existente")).toBe(-1)
})

test('Establecer objetos comprados a un usuario no existente', async () => {
   expect(await db.setBuy("Usuario no existente", ["chancla",
"cadena"])).toBe(-1)
})

test('Establecer objetos equipados a un usuario no existente', async () => {
   expect(await db.setEquip("Usuario no existente", {
       Body: "pajarita",
       Down: "b3",
       Face: "gafas",
       Head: "b1"
   })).toBe(-1)
})

test('Obtener objetos equipados de un usuario no existente', async () => {
   expect(await db.getEquipped("Usuario no existente")).toBe(-1)
})

test('Obtener objetos equipados del usuario Prueba', async () => {
   expect(await db.getEquipped("Prueba")).toStrictEqual({
       Body: "b2",
```

```
        Down: "b3",
        Face: "b4",
        Head: "b1"
    })
})

test('Obtener el nombre de la mascota del usuario Prueba', async () => {
    expect ( await db.getPetName("Prueba")).toBe("PruebaPet")
})

test('Obtener el nombre de la mascota de un usuario no existente', async ()
=> {
    expect ( await db.getPetName("Usuario no existente")).toBe("")
})

test('Establecer el nombre de la mascota de un usuario no existente', async
() => {
    expect ( await db.setPetName("Usuario no existente")).toBe(-1)
})

// test('Establecer la fecha de un usuario no existente', async () => {
//     expect ( await db.setFecha("Usuario no existente")).toBe(-1)
// })

test('Obtener la fecha de un usuario no existente', async () => {
    expect ( await db.getFecha("Usuario no existente")).toBe("")
})

test('Obtener la fecha del usuario Prueba', async () => {
    expect ( await db.getFecha("Prueba")).toBe("12/02/2002")
})
```

10. Herramientas Software usadas

Comunicación:

- WhatsApp
- Discord

Trabajo colaborativo:

- Github
- Trello

Elaboración de documentos:

- Google Documents

Elaboración de requisitos:

- MagicDraw

Elaboración de imágenes:

- Paint Tool Sai 2
- Pixilart: https://www.pixilart.com/

Elaboración de código:

- Visual Studio Code