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

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 minijuegos. Estos juegos serán sobre cultura general en el que el usuario deberá contestar una serie de preguntas.

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

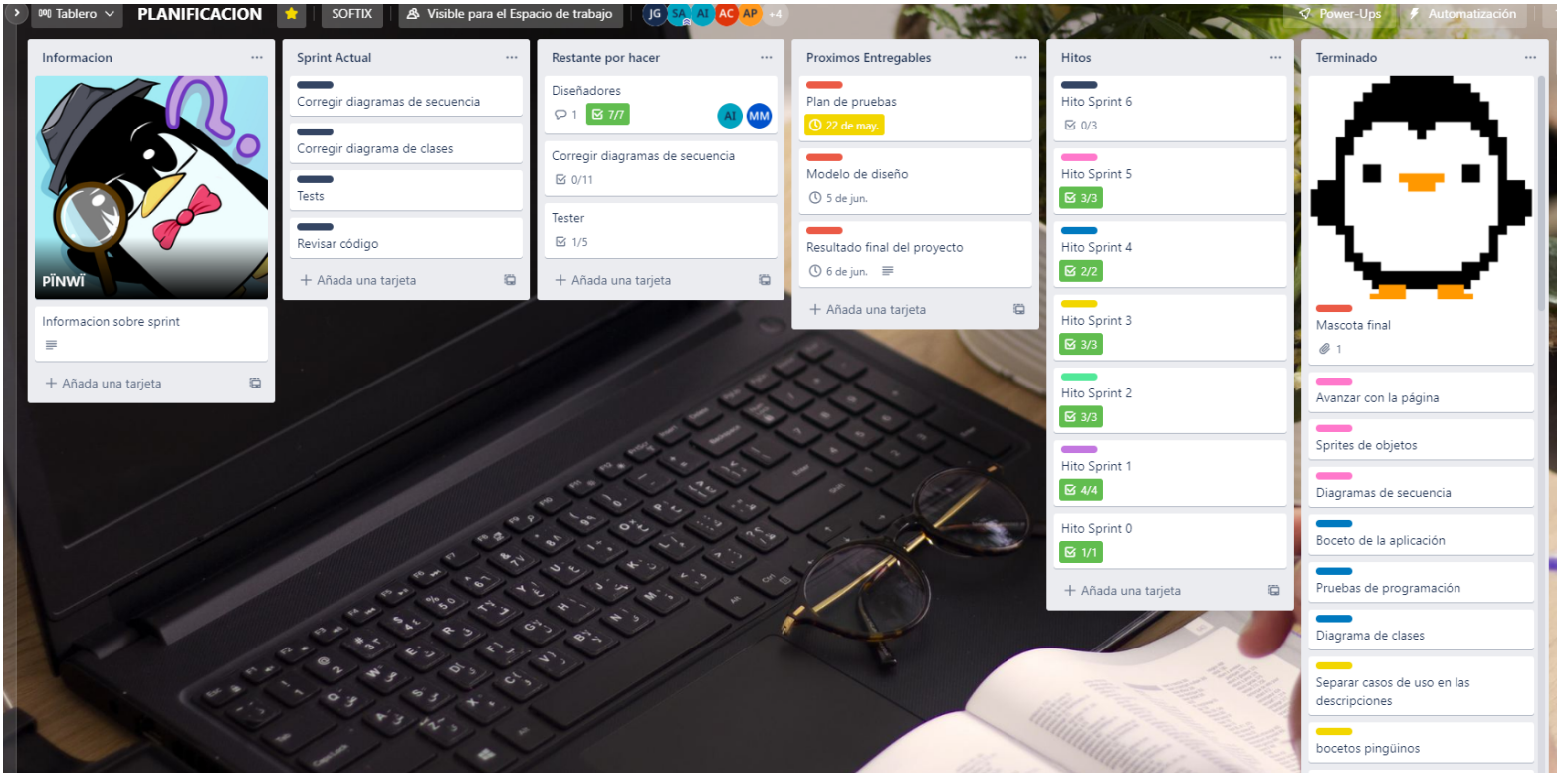
3. Gestión de Riesgo

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

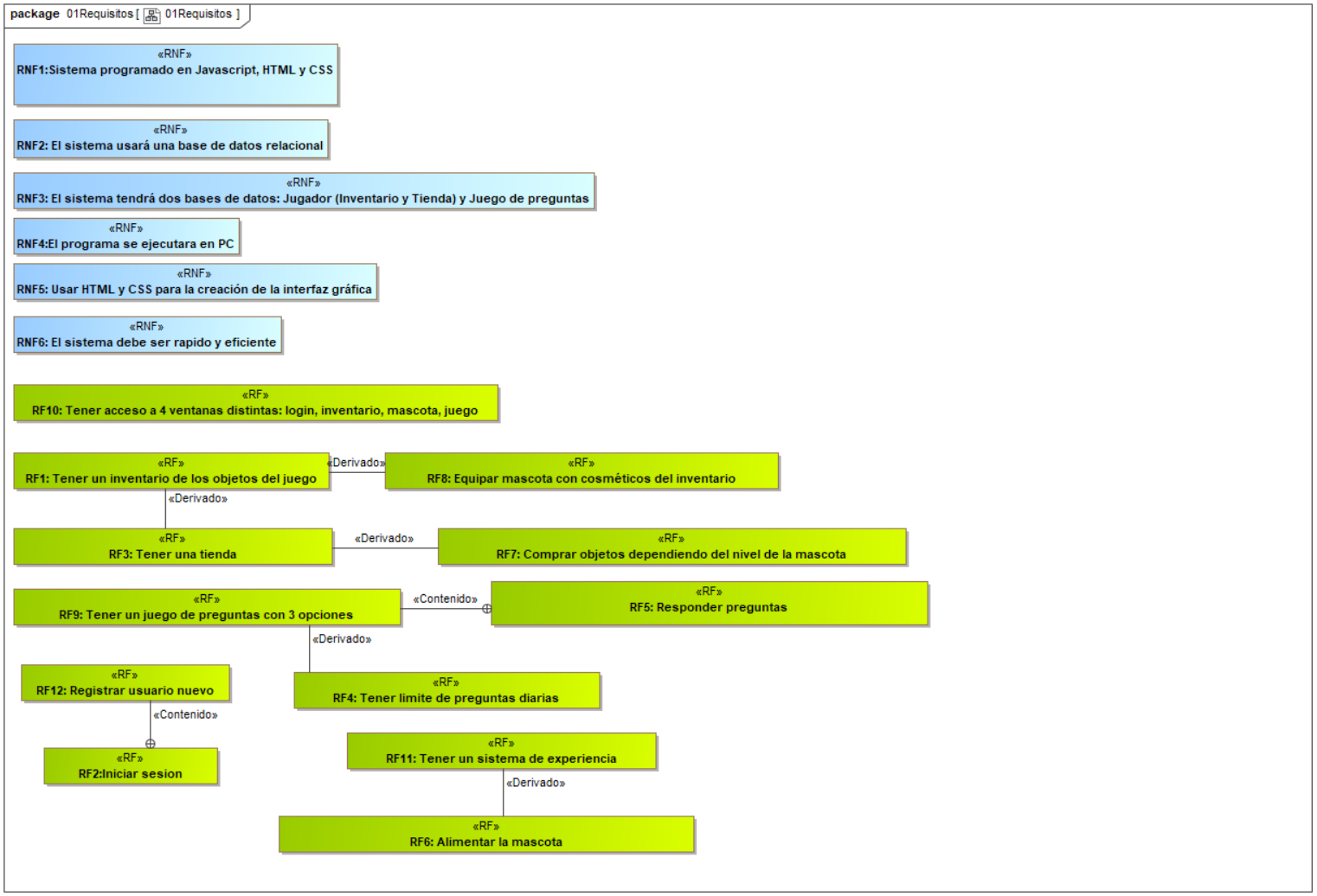
4. Planificación

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 diarias
   Para que mi juego no sea adictivo.

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 mejorar el nivel de mi mascota y usar mi dinero.

   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** desarrollador

   Quiero que los objetos de la tiendas se permiten comprar según el nivel de la mascota

   Para hacer el juego más entretenido.

   Pruebas de aceptación:

   - El usuario tiene la mascota con un bajo nivel y no se le permite comprar algunos objetos bloqueados por nivel.
   - El usuario sube de nivel y desbloquea algunos objetos.

8. **Como** usuario

   Quiero equipar a la mascota con cosméticos que he adquirido

   Para que mi mascota sea más personalizable y darle una utilidad al dinero.

   Pruebas de aceptación:

   - El usuario adquiere un objeto de la tienda.
   - El usuario equipa el objeto a la mascota.

9. **Como** desarrollador

Quiero que las preguntas tienen 3 opciones de respuesta
Para darle opciones al usuario
Pruebas de aceptación:

- El usuario va a jugar y tiene 3 preguntas que responder .

10. **Como** usuario
    Quiero que se actualicen las preguntas cada intervalo de tiempo
    Para que no sean las mismas preguntas siempre y sea más entretenido.

11. **Como** usuario
    Quiero que la mascota tenga un sistema de experiencia y puntos
    Para hacer tangible mi progreso.

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.

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.

2. **Como** desarrollador
   Quiero usar una base de datos 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

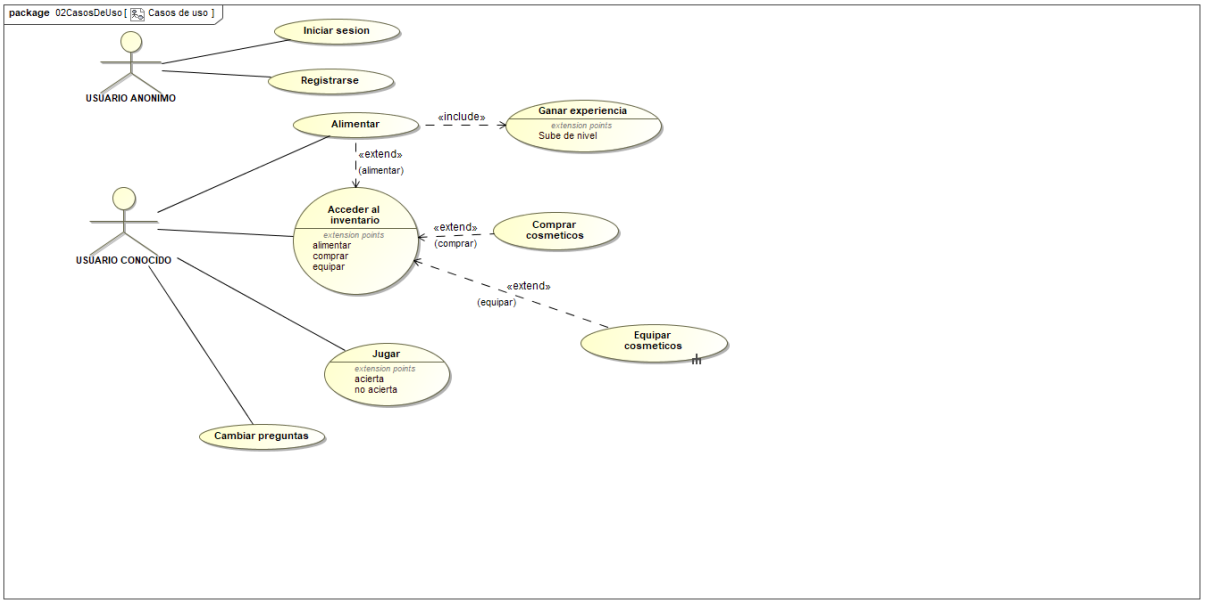
   Para poder almacenar información de manera eficiente.

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

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

6. **Como** desarrollador
   Quiero que la aplicación se ejecute en pc
   Para ofrecersela al público objetivo

6. Casos de uso



Detalles en documentación:

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.

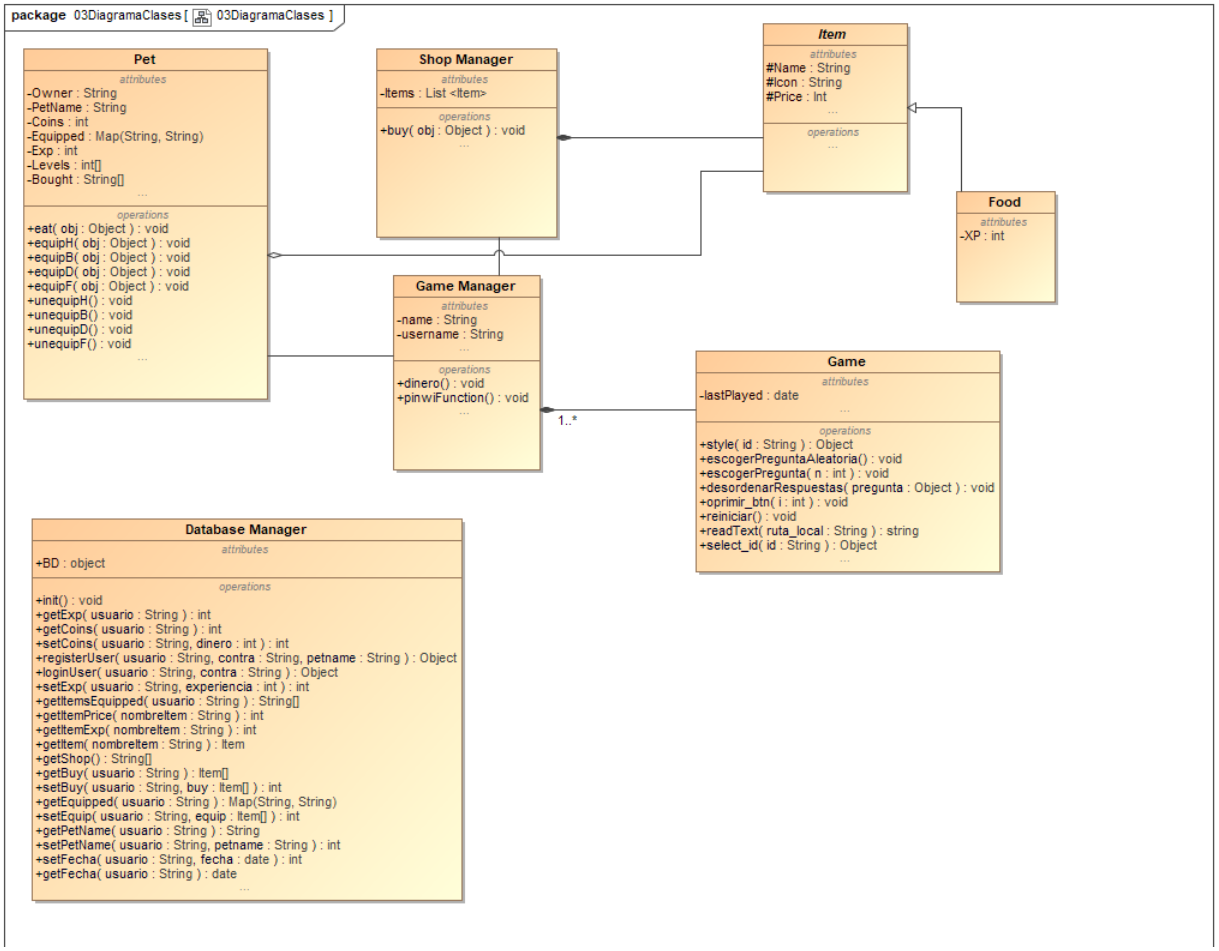
6.- Cambiar las preguntas

- Identificador único: CU6 El usuario conocido obtiene las preguntas.
- Contexto de uso: Cada 24 horas, se cambian las preguntas de los juegos.
- Precondiciones y activación:
    - Pasan 24 horas con respecto a la última vez.
- Garantía de uso: Las preguntas se cambian.
- Escenario principal:
    1. Se cambian las preguntas.
- Escenarios alternativos:
    1. Problemas en el cambio de las preguntas.

7.- Comprar cosméticos

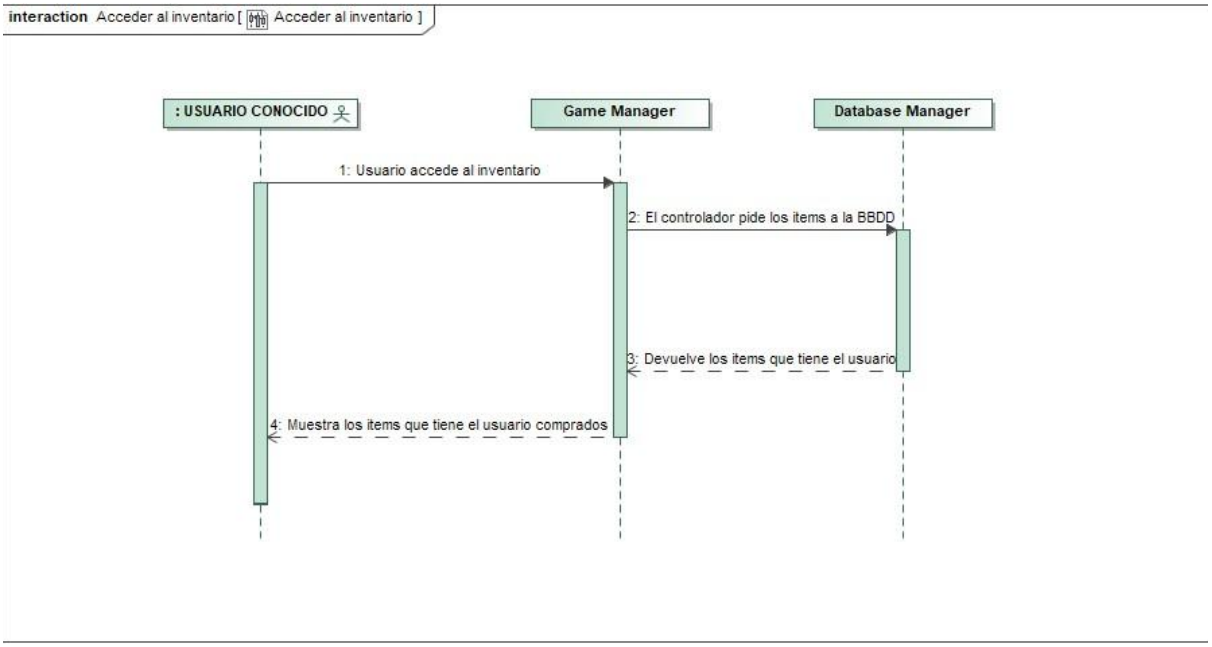
- Identificador único: CU7 El usuario compra un cosmético.
- Contexto de uso: El usuario quiere comprar un cosmético desde el inventario.
- Precondiciones y activación:
    El usuario tiene que haber iniciado sesión.
    El usuario tiene que estar en el inventario.
- Garantía de uso: El usuario consigue 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. Modelo del dominio

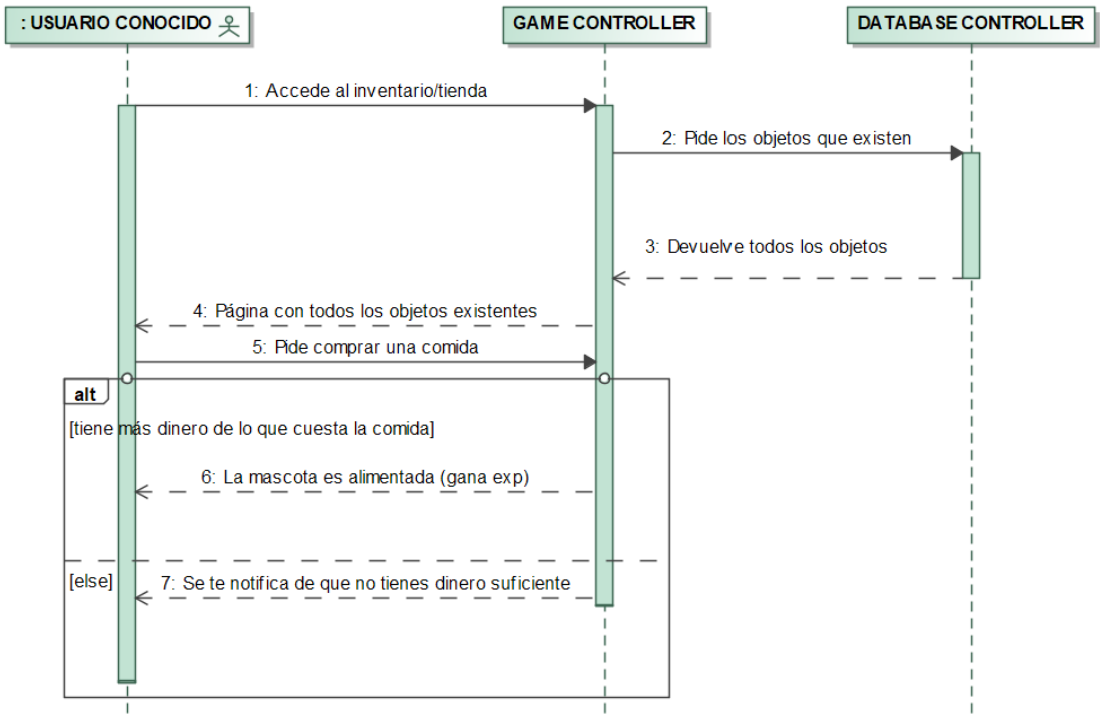


8. Diagramas de secuencia

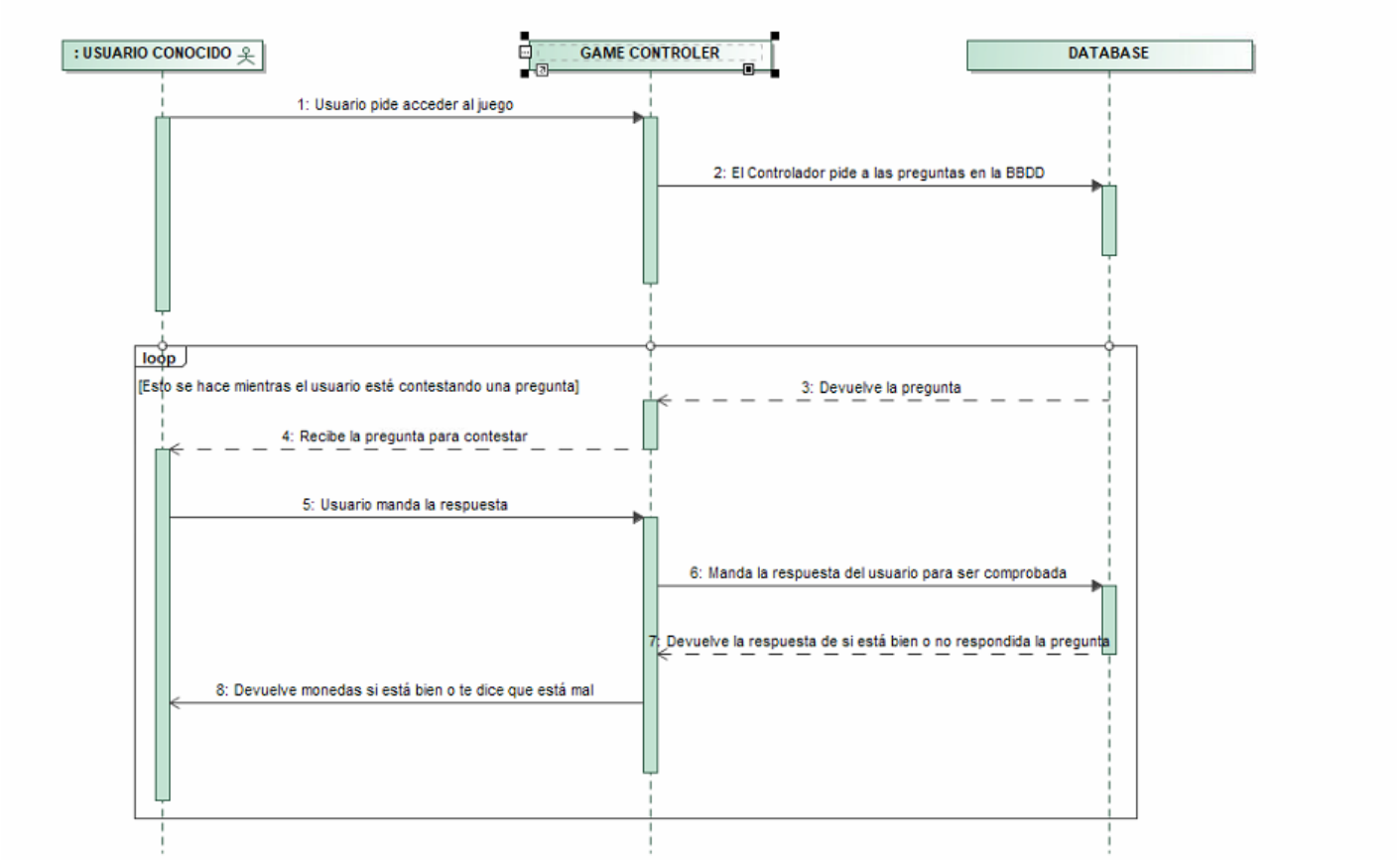
- Acceder al inventario: Cuando el usuario quiera acceder al inventario, el controlador pide los ítems a la base de datos. Para ello, el Database Manager obtendrá los ítems que posee el usuario, se los devolverá al Game Controller y este se los mostrará al usuario.



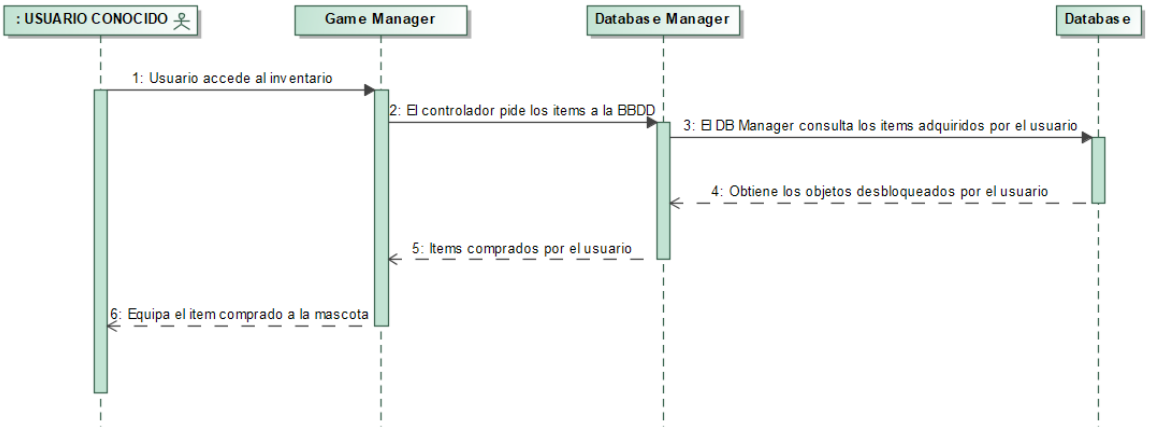
- Alimentar: Cuando el usuario quiera alimentar a la mascota, primero accederá al inventario para luego intentar comprar la comida: si tiene dinero suficiente la mascota será alimentada y ganará experiencia, y si no tiene dinero suficiente se notificará de esto.



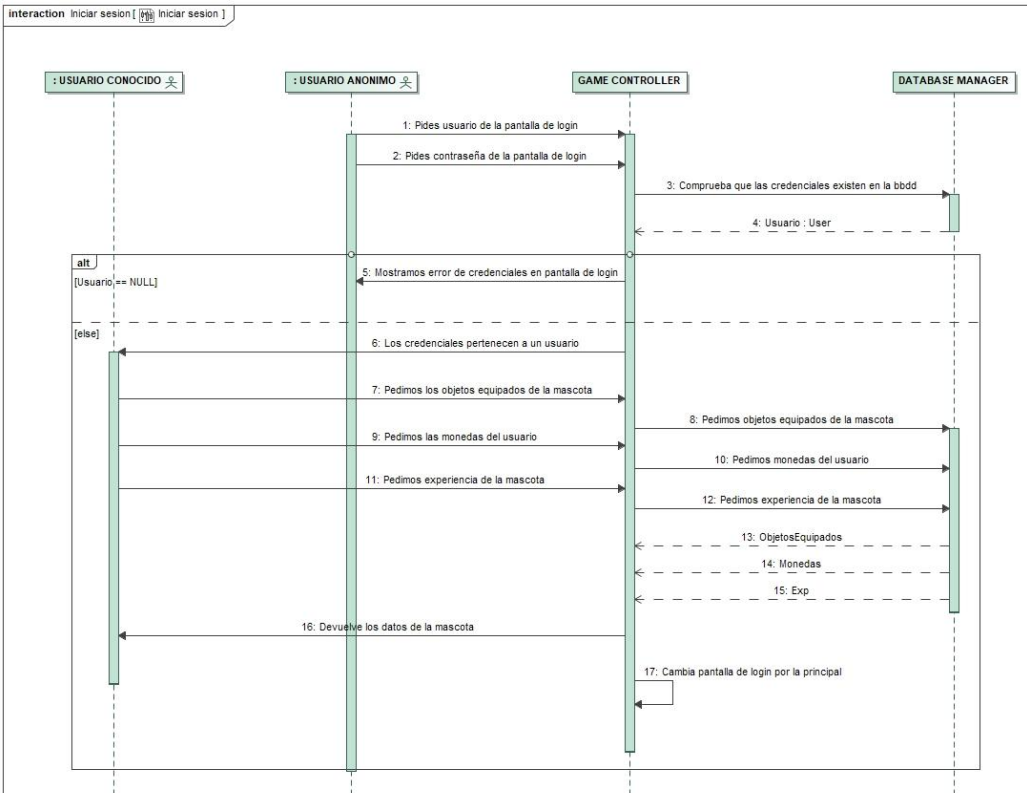
- Jugar: Cuando el usuario vaya a jugar, el controlador pedirá las preguntas a la base de datos para que el usuario pueda verla y contestarla. Mientras el usuario está con la pregunta, envía una respuesta y recibe monedas si está bien contestada o recibe un mensaje de que está mal contestada y que siga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



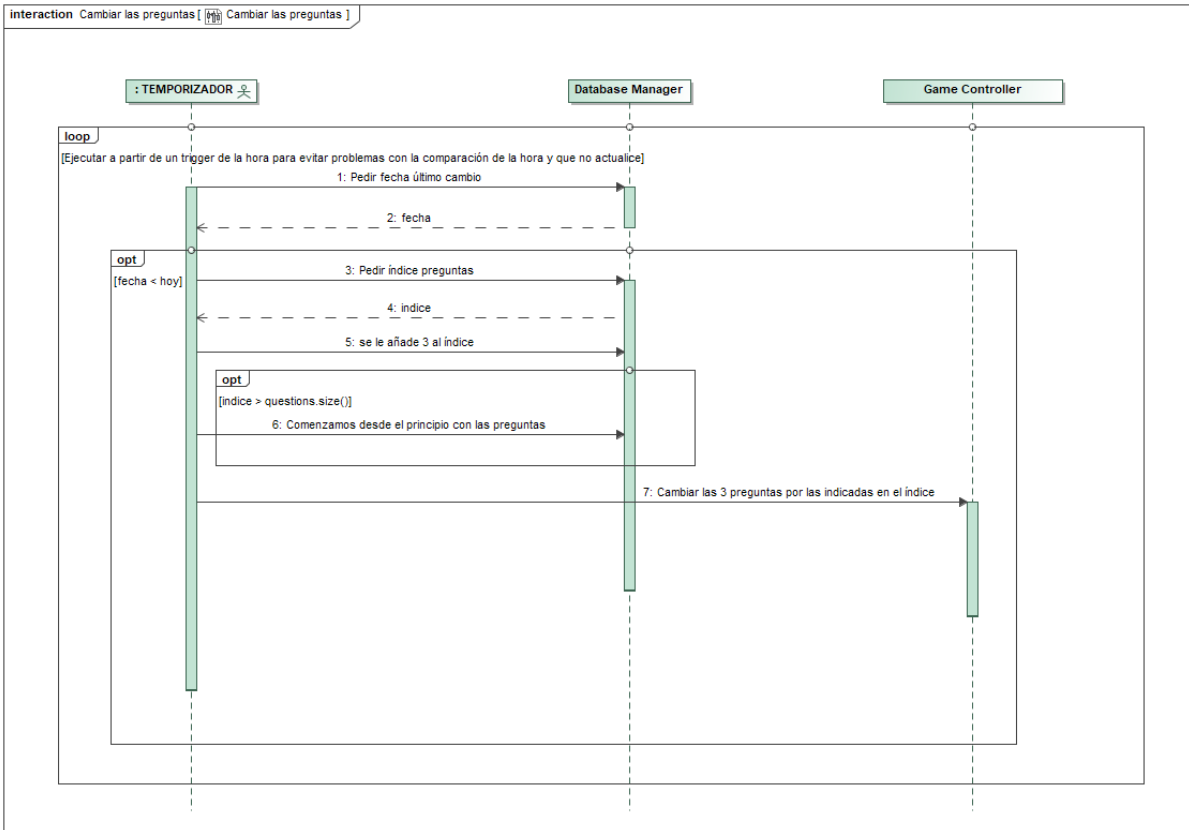
- Equipar cosméticos: Cuando el usuario quiera equipar cosméticos a la mascota, el controlador pide los ítems a la base de datos. Para ello, el Database Manager obtendrá los ítems que hayan sido comprados por el usuario, se los devolverá al Game Manager y este se los equipará a la mascota.



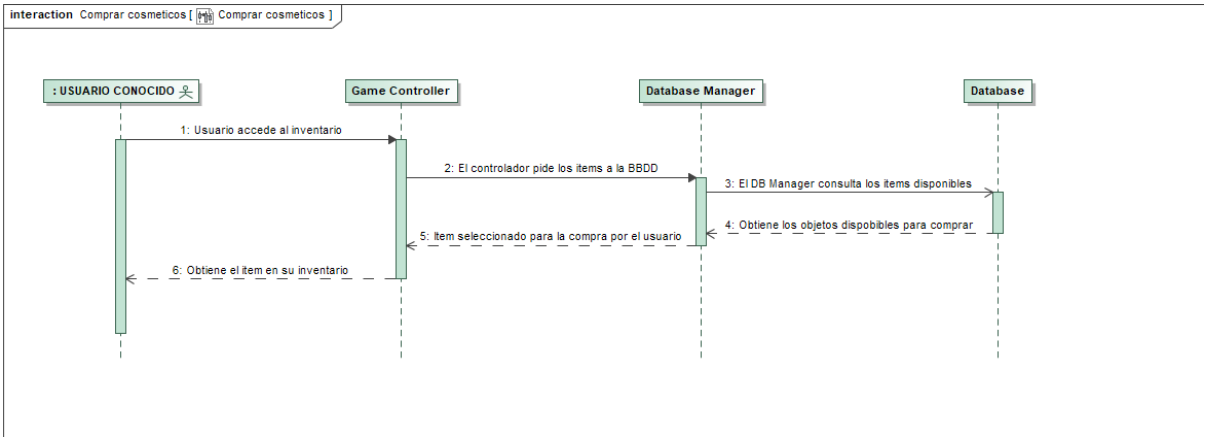
- Iniciar sesión: Cuando un usuario anónimo introduce sus credenciales en la pantalla de login,el controlador comprueba que sus datos existan en la base de datos, y en ese caso, se crea el objeto usuario con todo lo que tenga guardado a su nombre en la base de datos. Si no, da un mensaje de error.



- Cambiar pregunta: El usuario temporizador, tiene un trigger activado por fecha y hora, que comprueba en la bbdd el índice actual de las preguntas usadas en el juego, y lo aumenta 3 para las siguientes 3 preguntas. En el caso de que se haya excedido de la cantidad de preguntas disponibles comienza desde 0 en la lista.



- 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.



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.js

JS juego.test.js > ...

```
1   /*import DBManager from './DBManager.js';
2   DBManager = require('DBManager');*/
3   import juego from './juego.js';
4   juego = require('juego');
5
6   test('Escoger una pregunta con un indice no existente', async () => {
7       expect(await juego.escogerPregunta(8).toBe(-1))
8   })
9
10  test('Oprimir un boton no existente', async () => {
11      expect(await juego.oprimir_btn(8).toBe(-1))
12  })
13
14  test('Seleccionar una id no existente', async () => {
15      expect(await juego.select_id("pp").toBe(-1))
16  })
17
18  test('Seleccionar un style no existente', async () => {
19      expect(await juego.style("abduscan").toBe(-1))
20  })
21
22  test('Leer una ruta para escoger las preguntas no existente', async () => {
23      expect(await juego.readText('./preguntas.json').toBeNull())
24  })
25
```

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

```
import{DBManager} from "./DBManager.js";
import {buy,eat} from "./inventory.js";
beforeEach(async ()=>{
  window.sessionStorage.clear();
  window.sessionStorage.setItem("name", "alex");
  //Inicializamos al usuario alex con los siguientes parametros
  await DBManager.BD.setCoins("alex",55);
  await DBManager.BD.setBuy("alex",[]);
  await DBManager.BD.setExp("alex",0);
  await DBManager.BD.setEquip("alex", []);
})

test("buy añade objeto", async () => {
    let obj = {id: "gato"};//identificamos el objeto que deseamos comprar de la base de datos
    let moneyOld =await DBManager.BD.getCoins("alex");
    await buy(obj);
    //Esperamos que buy() haya añadido el objeto al inventario de usuario
    expect(await DBManager.BD.getBuy("alex")).toContain("gato");
    //Esperamos que buy() haya modificado el dinero de usuario(decrementandolo de acuerdo al precio del producto)
    expect(await DBManager.BD.getCoins("alex")).toEquals(moneyOld-(await DBManager.BD.getItemPrice("gato")));

  })

  test("eat compra comida y alimenta a la mascota",async () => {
    let moneyOld = await DBManager.BD.getCoins("alex");
    let expOld = await DBManager.BD.getExp("alex");
    let food = {id: "sandia"};
    await eat(food);
     //Esperamos que eat() haya modificado el dinero de usuario(decrementandolo de acuerdo al precio del producto)
    expect(await DBManager.BD.getCoins("alex")).toEquals(moneyOld-(await DBManager.BD.getItemPrice("sandia")));
    //Esperamos que eat() haya modificado la experiencia del usuario (incrementandola de acuerdo a la experiencia asociada al producto)
    expect(await DBManager.BD.getExp("alex")).toEquals(expOld + (await DBManager.BD.getItemExp("sandia")));
  })

  test("comprobamos que los metodos equip() equipen los objetos", async () => {
    let cosmeticH = {id: "gorro"};
    let cosmeticB = {id: "pajarita"};
    let cosmeticD = {id: "chancla"};
    let cosmeticF = {id: "gafas"};
    await DBManager.BD.setBuy("alex",["gorro","pajarita","chancla", "gafas"]); //Señalamos que estos objetos pertenecen al usuario
    await DBManager.BD.equipH("gorro");
    await DBManager.BD.equipB("pajarita");
    await DBManager.BD.equipD("chancla");
    await DBManager.BD.equipF("gafas");
    expect(await DBManager.BD.getEquipped("alex")).toContain("gorro"); //Esperamos que equipH() haya equipado el objeto
    expect(await DBManager.BD.getEquipped("alex")).toContain("pajarita"); //Esperamos que equipB() haya equipado el objeto
    expect(await DBManager.BD.getEquipped("alex")).toContain("chancla"); //Esperamos que equipD() haya equipado el objeto
    expect(await DBManager.BD.getEquipped("alex")).toContain("gafas"); //Esperamos que equipF() haya equipado el objeto
})

test("comprobamos que los metodos unequip() quiten los objetos anteriormente equipados a la mascota", async () => {
  let cosmeticH = {id: "gorro"};
  let cosmeticB = {id: "pajarita"};
  let cosmeticD = {id: "chancla"};
  let cosmeticF = {id: "gafas"};

  await DBManager.BD.setBuy("alex",["gorro","pajarita","chancla", "gafas"]); //Señalamos que estos objetos pertenecen al usuario
  await DBManager.BD.setEquip("alex",["gorro","pajarita","chancla", "gafas"]); //Señalamos que los objetos estan equipados a la mascota

  //Quitamos el equipamiento a la mascota
  await DBManager.BD.unequipH();
  await DBManager.BD.unequipB();
  await DBManager.BD.unequipD();
  await DBManager.BD.unequipF();

  expect(await DBManager.BD.getEquipped("alex")).not.toContain("gorro"); //Esperamos que unequipH() haya quitado los objetos de la cabeza
  expect(await DBManager.BD.getEquipped("alex")).not.toContain("pajarita"); //Esperamos que unequipB() haya quitado los objetos del cuerpo
  expect(await DBManager.BD.getEquipped("alex")).not.toContain("chancla"); //Esperamos que unequipD() haya quitado los objetos de los pies
  expect(await DBManager.BD.getEquipped("alex")).not.toContain("gafas"); //Esperamos que unequipF() haya quitado los objetos de la cara
})

```

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

```
import{DBManager} from "./DBManager.js";
import{index} from "./index.js";
test('El nombre de login tiene que ser igual que el nombre de register', async()=>{
    expect(loginUsername.value).toBe(signupUsername.value);
});

test('La contraseña de login tiene que ser igual que la contraseña de register', async()=>{
    expect(loginPassword.value).toBe(signupPassword.value);
});

//El nombre de la mascota es igual al
//nombre del usuario si deja el campo del nombre de la mascota
//vacio o igual al nombre que elija el usuario
test('El nombre de la mascota no puede estar vacio', async()=>{
    expect(petName).not.toBeUndefined();
});

```

- Pruebas para la Base de Datos:

```
//Hemos creado un usuario llamado 'Prueba' para poder probar todos los métodos de la base de datos

import  DBManager  from './DBManager.js';
const DB = 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", "alex")).toBe(0)
    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