DP1 2022-2023

Documento de Diseño del Sistema

Proyecto SOLITARIO

Repositorio: https://github.com/gii-is-DP1/dp1-2022-2023-l5-3.git
Vídeo: https://youtu.be/65uOcDNLOZI

Miembros:

- Barba Trejo, Francisco Javier
- Caro Albarrán, Francisco Andrés
- Gallego Huerta, Alberto
- Navarro Rivera, Álvaro
- Sánchez Naranjo,Mario
- Sillero Manchón, Jorge

Tutor: José Antonio Parejo

GRUPO G5-03

Versión 1.0

16/11/2022

Historial de versiones

Estos son ejemplo del contenido que debería tener el historial de cambios del documento a entregar a lo largo de los sprints del proyecto

Fecha	Versión	Descripción de los cambios	Sprint
13/12/2020	V1	● Creación del documento	2
13/12/2020	V2	● Añadido diagrama de dominio/diseño ● Explicación de la aplicación del patrón caché	3

Contents

Introducción

El solitario es un juego de naipes que, como su propio nombre indica, es jugado por una sola persona. La partida comienza con 7 columnas de cartas que , de izquierda a derecha ,cada columna debe tener una carta más que la anterior , estando todas las cartas boca abajo excepto la primera de cada columna. Las cartas sobrantes se recogen en un mazo aparte del que se van sacando cartas de 1 en 1(tal y como acordamos todos los miembros del grupo,ya que es posible sacar de 2 en 2 o de 3 en 3), si se ha sacado una carta de este mazo y no se ha usado, al sacar otra carta, la anterior se colocará en la última posición del mazo.

El jugador deberá colocar las cartas de las columnas o del mazo en otras columnas de manera que formen escaleras de cartas de mayor a menor y siempre con colores intercalados,es decir,no puedes colocar dos cartas seguidas con el mismo color.

Posteriormente,el jugador deberá almacenar cartas en las pilas,clasificadas por tipos:diamantes, corazones, tréboles y picas, de menor a mayor.

El tiempo de partida puede variar según cada persona. La partida termina cuando se consigue construir 4 pilas de cartas del mismo palo o bien cuando no haya más movimientos posibles,es decir, que no se pueda colocar más cartas ni en la pilas ni en las columnas.

Con esta implementación pretendemos crear un juego de solitario totalmente funcional y que soporte más funcionalidades extra, como puede ser gestión de usuarios y otros opcionales,como por ejemplo un módulo que permita saber las estadísticas globales y por usuarios.

Existen funcionalidades en nuestro proyecto que son interesantes(ya sea desde la implementación o el diseño),como puede ser mover una carta ,que puede estar contenida en un mazo final ,intermedio o inicial(definida en el modelo de datos como mazoFinal,mazoIntermedio y mazoInicial) en una posición concreta que viene definida en cartasPartida. Dicha carta puede moverse a cualquier otro mazo(siempre que cumpla las reglas de negocio RN-2,RN-3,RN-4,RN-10,RN-11,RN-12) cambiando de posición en cartasPartida y el id del mazo al que pertenece.

Otra funcionalidad interesante puede ser la de terminar partida, que puede ser posible siempre que el jugador sea el asociado a la partida.Pueden surgir dos posibilidades:que se finalice la partida dándole al botón de finalizar partida y no se cumpla las condiciones de victoria(en ese caso la partida se da por perdida), o que la partida se finalice y si se cumplan(en específico debe saltar la regla de negocio RN-7),entonces en ese caso la partida se considerará como ganada.Ambos resultados se almacenan en las estadísticas del jugador,cuya información vendrá en el módulo de estadísticas.

Diagrama(s) UML:

Diagrama de Dominio/Diseño

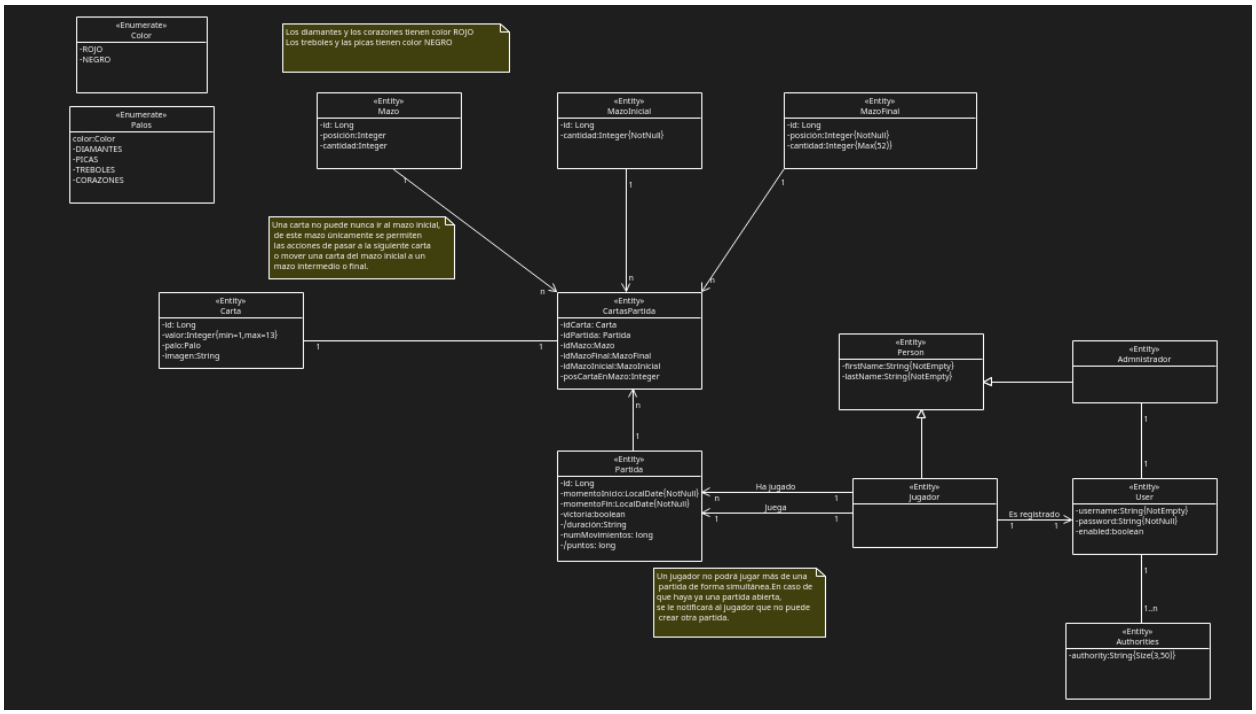
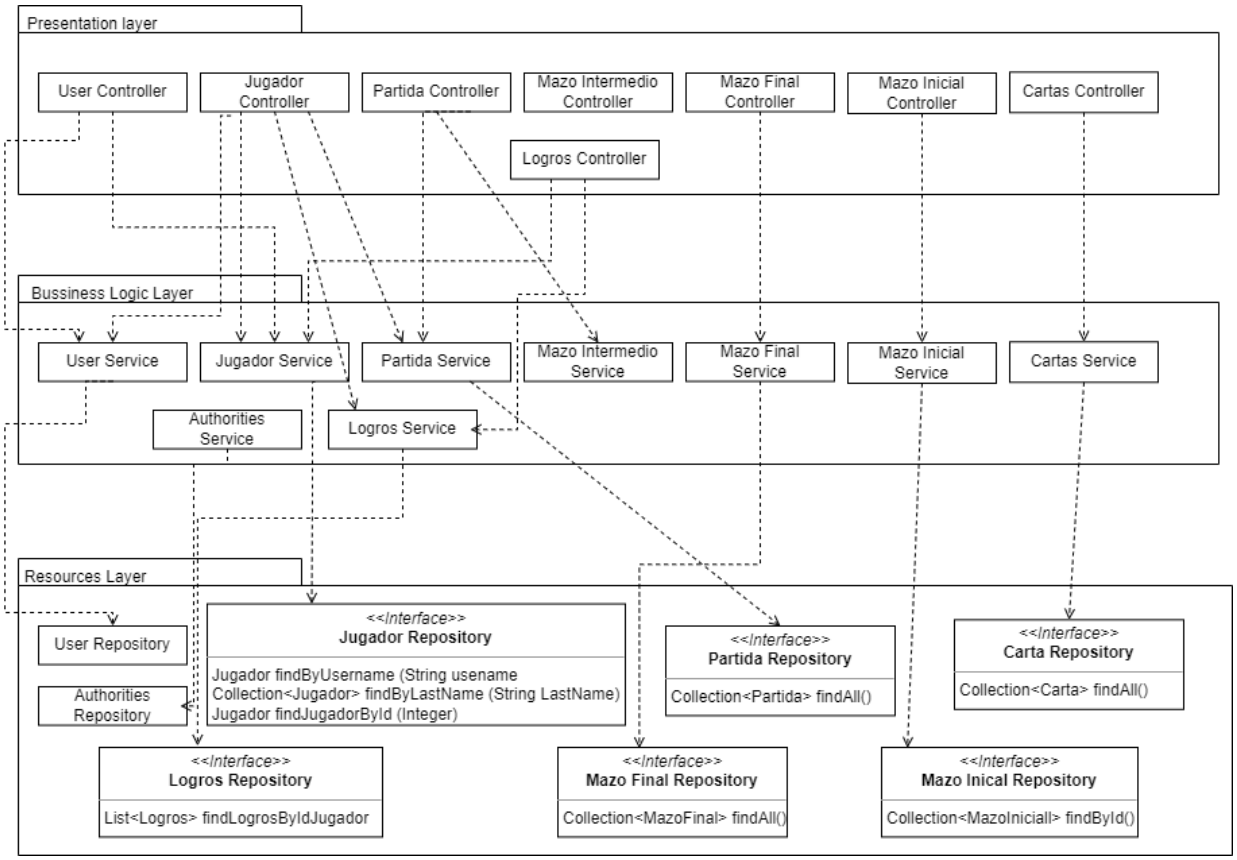


Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)

En esta sección debe proporcionar un diagrama UML de clases que describa el conjunto de controladores, servicios, y repositorios implementados, incluya la división en capas del sistema como paquetes horizontales tal y como se muestra en el siguiente ejemplo:



Patrones de diseño y arquitectónicos aplicados

En esta sección se especifica el conjunto de patrones de diseño y arquitectónicos aplicados durante el proyecto. Para especificar la aplicación de cada patrón puede usar la siguiente plantilla:

Patrón: <Prototipo>

Tipo: Diseño

Contexto de Aplicación

Hemos aplicado este patrón en los siguientes casos:

Todas las entidades heredan de BaseEntity,puesto que todas las entidades deben tener un identificador.

Jugador y administrador heredan de Person,puesto que estos dos tienen nombre y apellidos.

Clases o paquetes creados

Los prototipos que hemos implementado son:BaseEntity,que contiene un atributo id,y Person,que contiene nombre y apellidos.Todos estos prototipos se pueden ver dentro del paquete model.

Ventajas alcanzadas al aplicar el patrón

Hemos decidido hacer entidades específicas de forma que otras clases puedan heredar de él sus atributos y métodos. De esta forma evitaremos repetir el mismo código en múltiples clases.

Patrón: <Repositorio>

Tipo: Diseño

Contexto de Aplicación

Lo hemos aplicado a todas las entidades (jugador, mazoFinal, mazoIntermedio, mazoInicial, cartasPartida, carta, Partida) en sus respectivos paquetes

Clases o paquetes creados

jugadorRepository, mazoFinalRepository, mazoIntermedioRepository, mazoFinalRepository, cartaRepository, cartasPartidaRepository, PartidaRepository, LogroRepository.

Ventajas alcanzadas al aplicar el patrón

Aplicando este patrón podemos hacer consultas a la base de datos de la aplicación sin tener que acceder a ella directamente,aumentando así la mantenibilidad y reduciendo el acoplamiento de nuestra aplicación.

Patrón: <Constructor>

Tipo: Diseño

Contexto de Aplicación

Hemos utilizado este patrón para crear partidas,que se encarga de inicializar los elementos que componen una partida(mazos,cartasPartida….) y se puede encontrar en el paquete partida

Clases o paquetes creados

PartidaBuilder.java

Ventajas alcanzadas al aplicar el patrón

Al utilizar un constructor, la construcción del objeto se delega directamente a este,por lo que ganamos cohesión y rendimiento.

Patrón: <Paginación>

Tipo: Diseño

Contexto de Aplicación

Hemos utilizado la paginación para renderizar la lista de usuarios que contiene el sistema.

Clases o paquetes creados

Paginacion.java,PageRender.java,PageUserRepository.java

Ventajas alcanzadas al aplicar el patrón

Al aplicar este patrón,podremos ver a todos los usuarios registrados sin tener que plasmar toda esa información en una sola página,lo que mejora el rendimiento y navegación de la aplicación,ya que no carga toda la información de golpe y la vuelca en la página,sino que es dividido en páginas y se ve de manera más presentable.

Decisiones de diseño

En esta sección describiremos las decisiones de diseño que se han tomado a lo largo del desarrollo de la aplicación que vayan más allá de la mera aplicación de patrones de diseño o arquitectónicos.

Decisión 1: Movimiento de carta

Descripción del problema:

Como grupo nos gustaría poder mover las cartas de una manera simple y clara. El problema es que tenemos varias formas de abordar este tema.

Alternativas de solución evaluadas:

i

Alternativa 1.a:

Realizar el movimiento de cartas mediante javascript..

Ventajas

- Mejora la experiencia del usuario al hacer más intuitivo el movimiento de cartas

Desventajas

- Requiere un nivel mayor de complejidad y conocimiento de la tecnología.

Alternativa 1.b:

Utilizar formularios para el movimiento de las cartas.

Ventajas

- La implementación no es tan complicada como al utilizar javascript

Desventajas

- Puede entorpecer la experiencia del usuario.

Justificación de la solución adoptada

Como no disponemos de tiempo suficiente para dedicarle al proyecto hemos decidido que la opción más correcta sera utilizar formularios (opción 1.b)