DP1 2020-2021

Documento de Diseño del Sistema

Proyecto Samurai Sword

## https://github.com/gii-is-DP1/dp1-2021-2022-g7-01.git

Miembros:

* Cadenas Morales, Alfonso
* Crespo Quinta, Diego
* Fraile Rodríguez, Alejandro
* López Quirós, Juan
* Oliva Rodriguez, Pedro
* Quijano Herrera, Antonio

Tutor: Irene Bedilia Estrada Torres

GRUPO G7-1

Versión 2.0

16/12/201

# Historial de versiones

| **Fecha** | **Versión** | **Descripción de los cambios** | **Sprint** |
| --- | --- | --- | --- |
| 16/12/2020 | V1 | * Creación del documento | 2 |
| 16/12/2020 | V2 | * Añadido diagrama de dominio/diseño * Explicación de la aplicación del patrón caché | 3 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Contents

[https://github.com/gii-is-DP1/dp1-2021-2022-g7-01.git](#_heading=h.iutvvot8a8tq) 1

[Historial de versiones](#_heading=h.gjdgxs) **2**

[Introducción](#_heading=h.30j0zll) **4**

[Diagrama(s) UML:](#_heading=h.1fob9te) **4**

[Diagrama de Dominio/Diseño](#_heading=h.3znysh7) 4

[Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)](#_heading=h.2et92p0) 5

[Patrones de diseño y arquitectónicos aplicados](#_heading=h.tyjcwt) **5**

[Patrón: Command](#_heading=h.ywh0pg4wa2er) 5

[Tipo: Arquitectónico | de Diseño](#_heading=h.6ee003b7phbm) 5

[Contexto de Aplicación](#_heading=h.wxsgyuwj3ev0) 5

[Clases o paquetes creados](#_heading=h.ltj4z3yq2iaq) 5

[Ventajas alcanzadas al aplicar el patrón](#_heading=h.5n4hmip7p1ot) 5

[Tipo: Arquitectónico | de Diseño](#_heading=h.lw19x1lw83kv) 6

[Contexto de Aplicación](#_heading=h.rc3gdsudo8bu) 6

[Clases o paquetes creados](#_heading=h.flnllmdxmwyu) 6

[Ventajas alcanzadas al aplicar el patrón](#_heading=h.w0sox08n2ufd) 6

[Decisiones de diseño](#_heading=h.3dy6vkm) **6**

[Decisión 1: Completar logros](#_heading=h.mkamyrac5xyg) 6

[Descripción del problema:](#_heading=h.8wr9b14hskg1) 6

[Alternativas de solución evaluadas:](#_heading=h.l6px1p7l1xw) 6

[Justificación de la solución adoptada](#_heading=h.n4ehbf26uvwi) 7

[Decisión 2: Tabla para mano del jugador](#_heading=h.rlz7dnn1chxh) 7

[Descripción del problema:](#_heading=h.cc42fgocu58c) 7

[Alternativas de solución evaluadas:](#_heading=h.xt79f5cf1jg0) 7

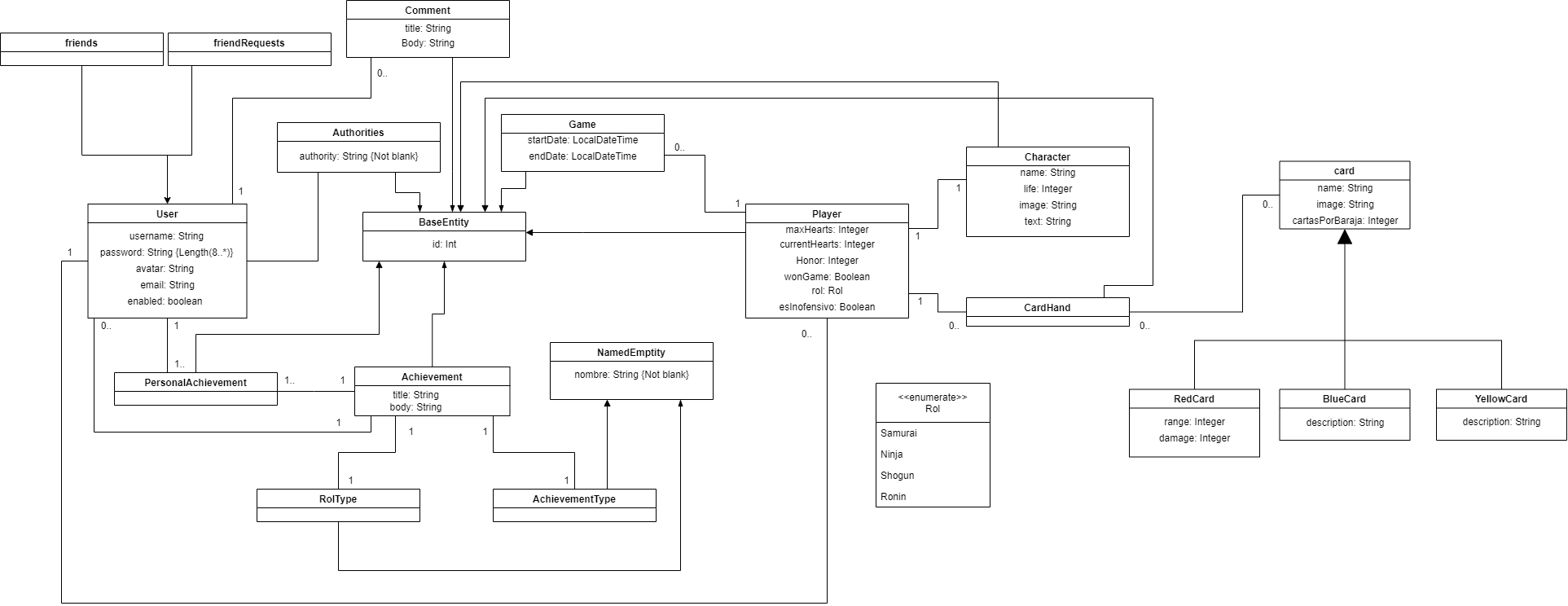
[Justificación de la solución adoptada](#_heading=h.kj582z67aq41) 7

# Introducción

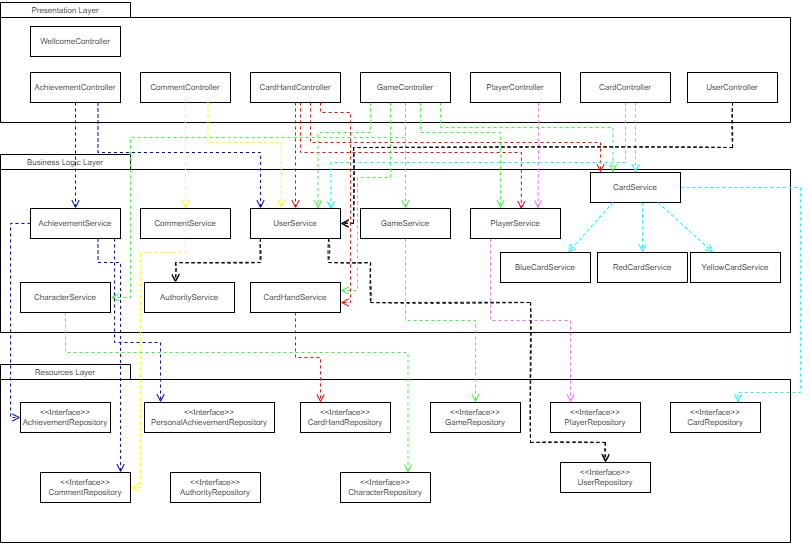
El objetivo del proyecto es implementar un juego de cartas físico como el Samurai Sword en una aplicación para poder jugar con tus amigos desde un ordenador y sin necesidad de llevar el juego en físico. Además se le podría más tarde añadir un módulo multijugador en línea.

# Diagrama(s) UML:

## Diagrama de Dominio/Diseño



## Diagrama de Capas (incluyendo Controladores, Servicios y Repositorios)



# 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: Command

### Tipo: Arquitectónico | de Diseño

### Contexto de Aplicación

Se ha utilizado para implementar las cartas. Cada carta tiene sus propios efectos, por lo que no hay forma de generalizar en un mismo método. Para solucionarlo hemos implementado el patrón command, tendremos un método para cada de carta pero todos implementará la interfaz Command

### Clases o paquetes creados

CardCommand

### Ventajas alcanzadas al aplicar el patrón

Siempre que querramos usar una cartar llamaremos a su método usar() que será independiente para cada carta

Patrón: Clone

### Tipo: Arquitectónico | de Diseño

### Contexto de Aplicación

Se ha utilizado para implementar las manos de los jugadores. Una baraja de samurai sword se compone de varias copias de la misma carta.

### Clases o paquetes creados

CardHand

### Ventajas alcanzadas al aplicar el patrón

En lugar de tener una entrada en la base de datos por cada copia, tenemos una columna que indica el número de copias

# 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: Completar logros

### Descripción del problema:

Como grupo nos gustaría poder hacer pruebas con un conjunto de datos reales suficientes, porque resulta más motivador. El problema es que al incluir todos esos datos como parte del script de inicialización de la base de datos, el arranque del sistema para desarrollo y pruebas resulta muy tedioso.

### Alternativas de solución evaluadas:

*Alternativa 1.a:* Redactar la forma de completar los logros a mano como una cadena en el formulario y almacenarla en una columna de la tabla en la base de datos que se convertiría en un predicate por código.

**Ventajas**:

* Simple, no requiere nada más que escribir el SQL que genera los datos.

**Inconvenientes**:

* Ralentiza todo el trabajo porque hay que escribir un predicate por cada logro

*Alternativa 1.b:* Parametrizar el texto que ofrece el logro a forma de descripción para que el usuario sepa la forma de completarlo y convertirlo en condiciones que se tienen que cumplir para ello.

**Ventajas:**

* Solo se escribe para cada tipo diferente de logro(escribir comentarios, ganar partidas)
* Todo estaría hecho en el código, lo que resta complejidad a la aplicación

**Inconvenientes:**

* El método que resuelve si un logro es obtenido o no será bastante complejo
* Si queremos añadir un logro nuevo habrá que sacar una nueva versión de la aplicación

### Justificación de la solución adoptada

Como consideramos que lo primordial es la comodidad y la simplicidad de la aplicación hemos elegido la alternativa 1.b

## Decisión 2: Tabla para mano del jugador

### Descripción del problema:

Al ser un juego de cartas necesitamos una manera de almacenar la mano de cada jugador

### Alternativas de solución evaluadas:

*Alternativa 2.a:* Tener una tabla que almacene las cartas que tiene un jugador en cada momento

**Ventajas**:

* Simple, no requiere nada más que escribir el SQL que genera los datos.

**Inconvenientes**:

* Tenemos una tabla vacía mientras no haya una partida en curso

*Alternativa 2.b:* Tener una lista por cada jugador y pasarle las todas las listas al modelo que se crean al principio de la partida y se destruyen al terminar

**Ventajas:**

* No almacenamos datos temporales

**Inconvenientes:**

* Es trabajoso tener los datos en varias listas distintas

### Justificación de la solución adoptada

Como consideramos que lo primordial es no tener una tabla para datos que solo utilizamos en un momento específico hemos elegido la alternativa 2.b