Índice

[1. ANÁLISIS: DESCRIPCIÓN DEL PROBLEMA A RESOLVER 1](#_Toc512957917)

[2. REQUERIMIENTOS 2](#_Toc512957918)

[2.1. REQUISITOS FUNCIONALES 2](#_Toc512957919)

[2.2. REQUISITOS NO FUNCIONALES 2](#_Toc512957920)

[2.3. MODELO DE DATOS 2](#_Toc512957921)

[2.4. DIAGRAMA DE CASOS DE USO 3](#_Toc512957922)

[2.5. RESTRICCIONES FUNCIONALES 5](#_Toc512957923)

[3. DISEÑO: ESQUEMA FUNCIONAL Y DE COMPONENTES 5](#_Toc512957924)

[4. DISEÑO: DIAGRAMA UML 6](#_Toc512957925)

[5. DOCUMENTACIÓN DEL CÓDIGO 9](#_Toc512957926)

[6. DEPENDENCIAS MAVEN 9](#_Toc512957927)

[7. DISEÑO DE LA BASE DE DATOS 9](#_Toc512957928)

[7.1. ENTIDAD RELACIÓN 9](#_Toc512957929)

[7.2. IMPLEMENTACIÓN FÍSICA 11](#_Toc512957930)

[8. MANUAL DE USUARIO 12](#_Toc512957931)

[9. APORTACIONES EXTRAORDINARIAS 12](#_Toc512957932)

# ANÁLISIS: DESCRIPCIÓN DEL PROBLEMA A RESOLVER

Uno de los matices más atractivos del hobby *Warhammer 40,000[[1]](#footnote-1)* (referido a partir de ahora como W40k) es su aspecto competitivo. Los jugadores pueden personalizar los ejércitos con una amplia variedad de unidades y equipamiento para enfrentarse a sus oponentes y determinar quién es el mejor comandante. Para ello, es necesario organizar campeonatos donde los jugadores puedan reunirse y jugar estas partidas.

El objetivo de este proyecto es el de desarrollar un software que permita organizar y monitorizar un campeonato a largo plazo, como una liga de fútbol. Para ello, los jugadores podrán darse de alta y gestionar los ejércitos con los que van a participar. Según el trasfondo de W40k, cada ejército pertenece a una facción, y emplean estrategias distintas: por ejemplo, un ejercito puede optar por una estrategia agresiva centrada en el combate cuerpo a cuerpo mientras que otro puede optar por otra estrategia estática y defensiva con armas a distancia.

De este modo, el sistema deberá recoger información sobre las partidas jugadas y quién fue el vencedor y el derrotado, y así poder extraer información en tiempo real del rendimiento de jugadores, ejércitos y facciones, y mostrarla de manera gráfica a los administradores del campeonato en un cuadro de mando. No se contempla la gestión de los administradores, que ya estarán dados de alta en el sistema al inicio del campeonato, y sus datos no serán modificados.

# REQUERIMIENTOS

## REQUISITOS FUNCIONALES

|  |  |  |
| --- | --- | --- |
| **Código** | **Requisito** | **Descripción** |
| **RF01** | Login | El sistema permitirá a los usuarios loguearse para acceder a su espacio personalizado, y, a los administradores, acceder a funcionalidades avanzadas. |
| **RF02** | Gestión de jugadores | El sistema permite gestionar los jugadores del campeonato. Los jugadores podrán darse de alta y gestionar sus datos. |
| **RF03** | Gestión de ejércitos | El sistema permite gestionar los ejércitos del campeonato. Los jugadores podrán registrar nuevos ejércitos con los que participar y consultar el rendimiento de sus ejércitos. |
| **RF04** | Gestión de partidas | El administrador podrá registrar el resultado de una partida, y se guardarán todos los datos referentes a ella. |
| **RF05** | Visualización de rankings | El administrador podrá visualizar diversa información sobre el rendimiento de ejércitos, jugadores y facciones, a modo de tabla y diagramas en un cuadro de mandos. |

## REQUISITOS NO FUNCIONALES

|  |  |  |
| --- | --- | --- |
| **Código** | **Requisito** | **Descripción** |
| **RNF01** | Diseño *Responsive* | La aplicación debe adaptarse a distintos tamaños de pantalla y dispositivo. Se dará prioridad a los dispositivos con relación de aspecto 16:9 y 9:16, es decir, monitores de ordenador tradicional y dispositivos móviles en vertical. |
| **RNF02** | Gráficos en tiempo real | La información que aparece en los gráficos debe actualizarse en tiempo real y refrescarse cuando se añada nueva información. |

## MODELO DE DATOS

El modelo de datos de la aplicación queda recogido en el siguiente modelo de dominio:



## DIAGRAMA DE CASOS DE USO

Los casos de uso de la aplicación se recogen en siguiente diagrama:



## RESTRICCIONES FUNCIONALES

Se aplican las siguientes restricciones:

* Borrado de información: Para no provocar borrados en cascada que causen situaciones imprevistas, como, por ejemplo, que se borre uno de los participantes en una partida, pero no el otro. No se permite el borrado de ejércitos ni de sus jugadores si éstos ya han participado en una partida.
* Modificación de información: No se permitirá modificar la facción o la estrategia de un ejército. Si un jugador desea cambiar estos valores en mitad del campeonato, se deberá registrar un ejército nuevo.
* Asignación de puntos a jugadores: Los puntos se asignarán automáticamente al registrar partidas donde participen. No se podrán modificar directamente.
* No se contempla la gestión de administradores. Estos obtendrán sus credenciales directamente de la base de datos, y su información no será modificada a lo largo del mismo. Por lo tanto, los administradores no podrán participar en partidas.

# DISEÑO: ESQUEMA FUNCIONAL Y DE COMPONENTES

Los componentes de la aplicación pueden separarse en dos grupos: componentes web y componentes de lógica de negocio:

* **Componentes web:** Se corresponden a los que interactúan con el usuario en el navegador. Estos componentes se implementan mediante UIs de Vaadin. Siguiendo la filosofía de Vaadin, todas las vistas están en una única página, y la navegación se lleva a cabo mediante una barra menú lateral, que está presente en todo momento, y que muestra todas las opciones disponibles.
* **Componentes de lógica de negocio:** Implementan la funcionalidad de la aplicación web. Los componentes controladores se encargan de la comunicación de las vistas con la capa de acceso a datos mediante el uso de clases POJO que representan los objetos de la base de datos.

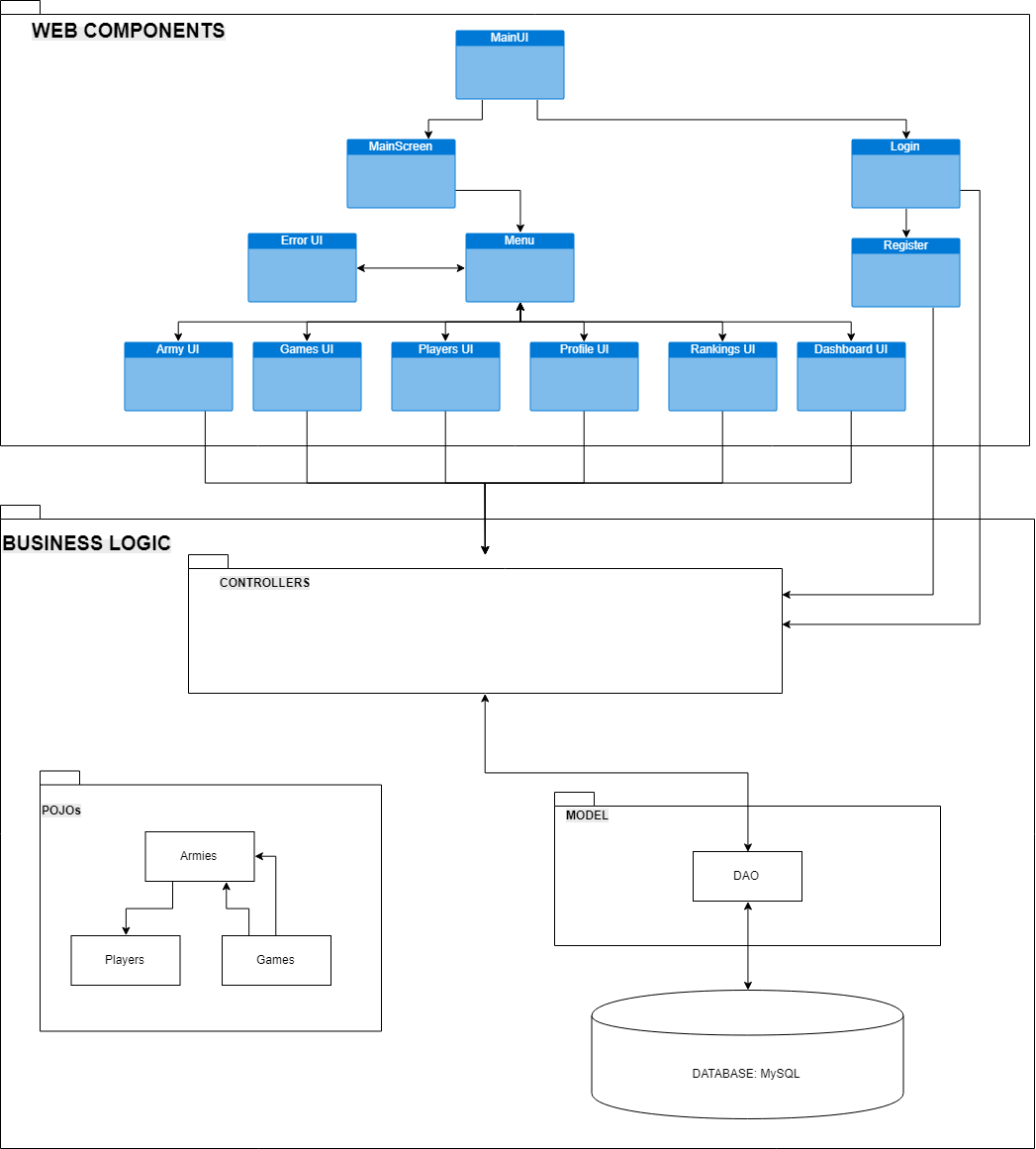
El desglose de componentes web es el siguiente:

|  |  |
| --- | --- |
| **COMPONENTE** | **FUNCIONALIDAD** |
| MainUI | Página única que contiene todas las vistas Vaadin |
| Login | Vista para que los usuarios se identifiquen |
| Register | Vista para que los usuarios se registren |
| MainScreen | Vista que contiene las funcionalidades principales de la aplicación. Cambiará las funcionalidades ofrecidas según si el usuario es jugador o administrador |
| Menu | Vista que contiene la barra de navegación lateral. Desde aquí se accede al resto de las vistas de la aplicación |
| ErrorUI | Vista que muestra un mensaje de error si fuera necesario |
| ArmyUI | Vista para gestionar los ejércitos |
| GamesUI | Vista para gestionar las partidas |
| PlayersUI | Vista para gestionar los jugadores |
| ProfileUI | Vista para gestionar el perfil de un jugador |
| RankingsUI | Vista para consultar las estadísticas del campeonato |
| DashboardUI | Vista principal de bienvenida a la aplicación para los usuarios logueados |

El desglose de componentes de lógica de negocio es el siguiente:

|  |  |  |
| --- | --- | --- |
| **PAQUETE** | **COMPONENTE** | **FUNCIONALIDAD** |
| Controllers | Controladores | Se encargan de comunicar la vista con el acceso a datos, así como de realizar las operaciones sobre los datos que sean necesarias. Será necesario un controlador por cada POJO |
| POJOs | Player | Clase que representa un objeto jugador de la base de datos con el que representar, en Java, la información que este contiene |
| Army | Clase que representa un objeto ejército de la base de datos con el que representar, en Java, la información que este contiene |
| Game | Clase que representa un objeto partida de la base de datos con el que representar, en Java, la información que este contiene |
| Model | DAO | Se encarga de realizar el acceso a datos, tomando parámetros desde el controlador y devolviendo instancias de los POJOs |
| Base de datos | Base de datos | Base de datos SQL que se encarga de la persistencia de la información de la aplicación |

Finalmente, la interrelación de los componentes se refleja con el siguiente diagrama:

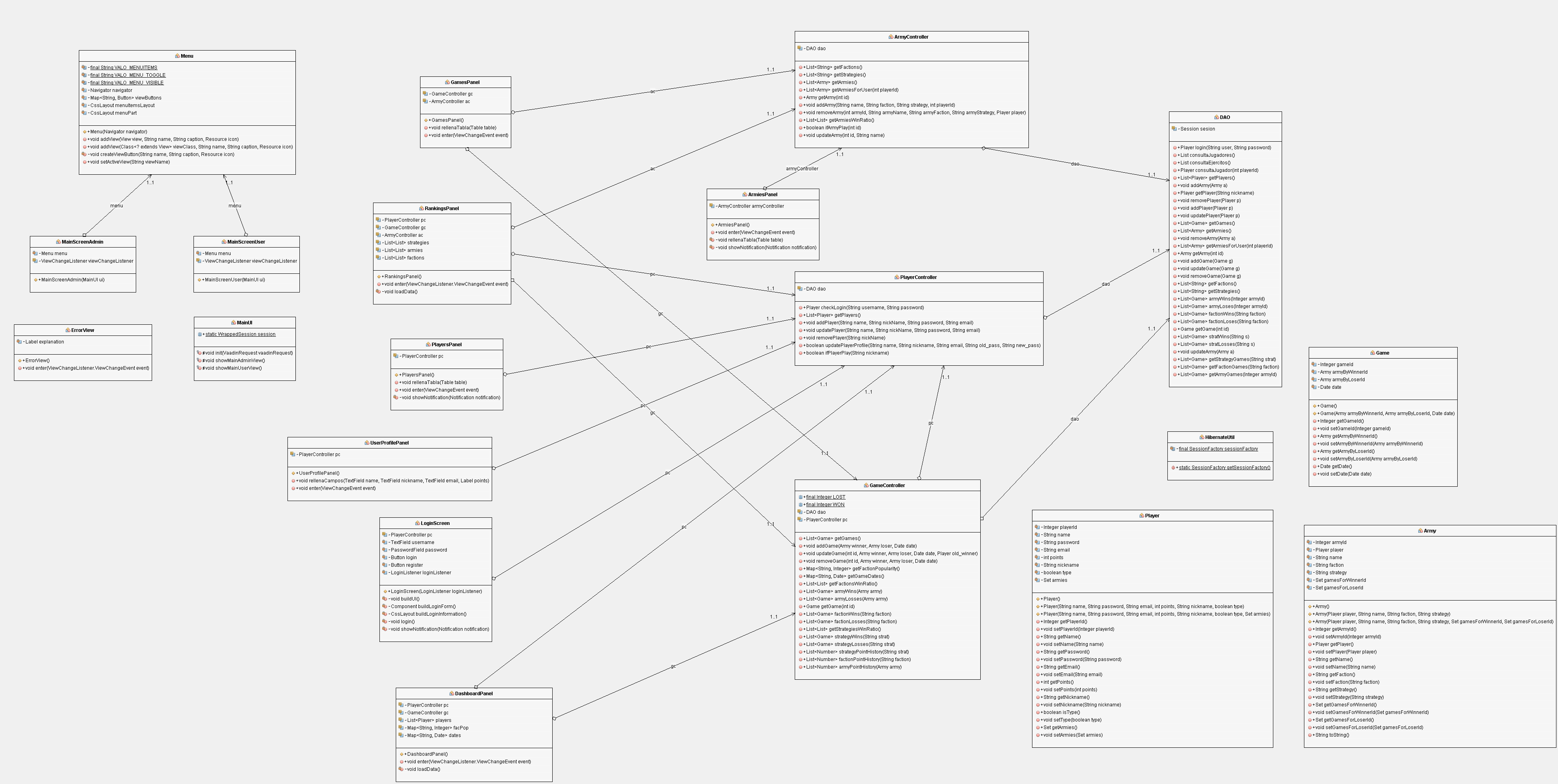


# DISEÑO: DIAGRAMA UML

Siguiendo el patrón MVC, se ha planteado el esquema anterior como una aplicación Java, y se han implementado las clases divididas por paquetes según su funcionalidad. Las clases para la aplicación son las siguientes:

|  |  |  |
| --- | --- | --- |
| **PAQUETE** | **CLASE** | **FUNCIONALIDAD** |
| POJOs | Army | Implementan los objetos extraídos de la base de datos. Todos los POJOs tienen métodos getter y setter pero no implementan ninguna lógica de negocio. Para una descripción más detallada de los objetos y entidades, ver puntos 2 y 7. |
| Game |
| Player |
| HibernateUtil | Factoría de sesiones para las conexiones a Hibernate |
| Controller | ArmyController | Implementa la lógica de negocio relacionada con los ejércitos |
| GameController | Implementa la lógica de negocio relacionada con las partidas |
| Playercontroller | Implementa la lógica de negocio relacionada con los jugadores |
| Model.Dao | DAO | Implementa la capa de acceso a datos mediante Hibernate. Sus métodos construyen consultas HQL mediante parámetros y devuelven POJOs o listas de POJOs |
| View | ErrorView | Implementa la vista de error |
| LoginScreen | Implementa la pantalla de login |
| Menu | Implementa el menú lateral |
| MainScreenAdmin | Añade contenido al menú lateral para los administradores |
| MainScreeUser | Añade contenido al menú lateral para los jugadores |
| MainUI | Implementa la vista principal que arranca la aplicación y llama a las demás vistas |
| View.Panels | ArmiesPanel | Genera el panel donde se gestionan los ejércitos |
| GamesPanel | Genera el panel donde se gestionan las partidas |
| PlayersPanel | Genera el panel donde se gestionan los jugadores |
| UserProfilePanel | Genera el panel donde un jugador gestiona su perfil |
| RankingsPanel | General el panel de la vista de rankings |
| DashboardPanel | Genera el panel de bienvenida de los usuarios |

Los métodos y atributos de las clases quedan reflejados en el siguiente diagrama UML de clases (se adjunta a tamaño completo “Documentacion/Fuentes/Diagrama UML de clases.png”:



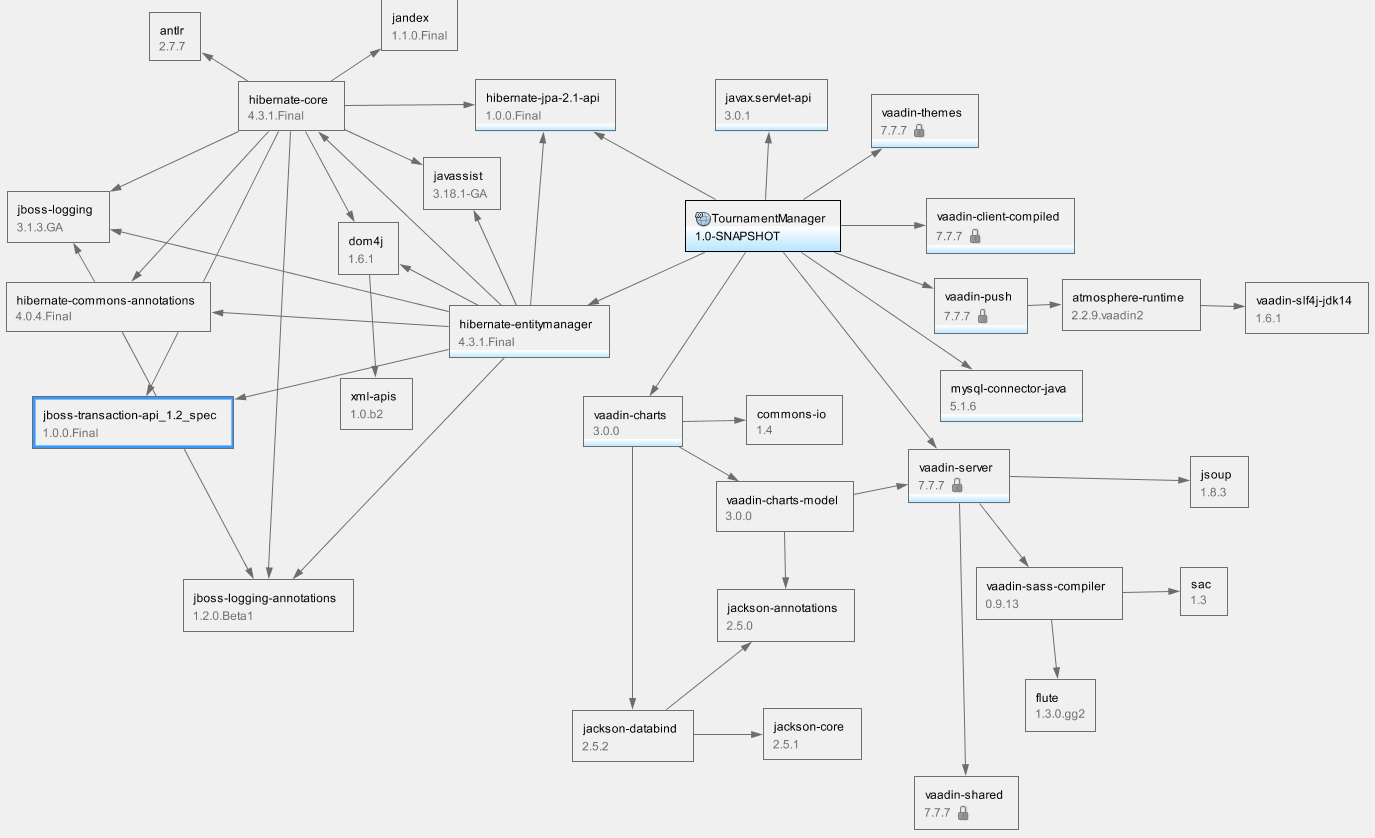
# DOCUMENTACIÓN DEL CÓDIGO

La documentación del código implementado se ha llevado a cabo usando la herramienta Javadoc. El resultado puede consultarse en la carpeta “Documentacion/Javadoc” de la entrega. Esta puede ser consultada a través de un navegador web, donde pueden verse las explicaciones detalladas de su funcionamiento e interrelación de componentes.

En concreto, se recomienda consultar las clases contenidas en los paquetes tournamentmanager.model y tournamentmanager.controller.

# DEPENDENCIAS MAVEN

Las dependencias del proyecto han sido organizadas con la herramienta Maven. El diagrama de dependencias es el siguiente:



# DISEÑO DE LA BASE DE DATOS

## ENTIDAD RELACIÓN

Debido a que los componentes del sistema de información están fuertemente relacionados, y a que el volumen de datos esperado es bajo, se ha optado por una base de datos relacional de tipo SQL. El diseño de la base de datos queda reflejado en el siguiente diagrama entidad-relación:



La descripción de las entidades es la siguiente:

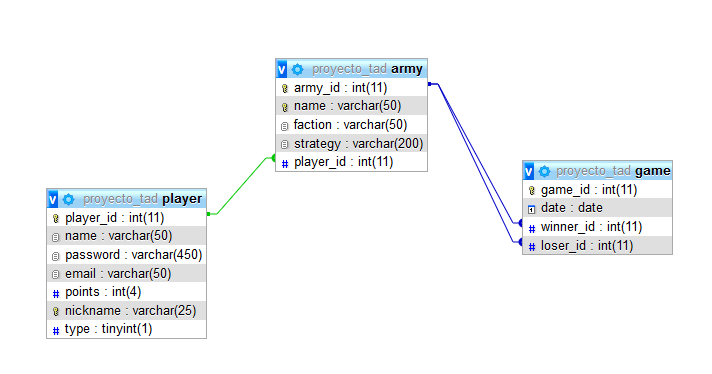
|  |  |
| --- | --- |
| **ENTIDAD** | **DESCRIPCIÓN** |
| USER | Representa a un usuario final de la aplicación. |
| ADMIN | Usuario administrador |
| PLAYER | Usuario jugador |
| ARMY | Representa un ejército que los usuarios jugadores usan para jugar partidas |
| GAME | Recoge el resultado de una partida de 40k |

La justificación de las relaciones es la siguiente:

|  |  |
| --- | --- |
| **RELACIÓN** | **DESCRIPCIÓN** |
| USER *is a* ADMIN & USER *is a* PLAYER | Representa una herencia de entidades. Todos los usuarios contienen las mismas columnas, pero administradores y jugadores tienen acceso a distintas funcionalidades. |
| ADMIN *manages* GAME | Los administradores gestionan las partidas. Cuando una partida se ha completado, éstos son los encargados de crearla, así como de modificarlas y borrarlas si fuera necesario. Los administradores también tienen acceso a la lista de partidas y a ciertas estadísticas generadas a partir de estas. |
| PLAYER *has* ARMY | Los jugadores deben tener uno o más ejércitos registrados con los que participar en el campeonato. Dispondrán de una sección para consultarlos y gestionarlos. |
| ARMY *loses* GAME & ARMY *wins* GAME | Por cada partida, se registra el ejército ganador y el perdedor. |

## IMPLEMENTACIÓN FÍSICA

La base de datos se ha implementado con MySQL, con las siguientes tablas:



Finalmente, sus atributos son:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLA** | **ATRIBUTO** | **TIPO DE DATO** | **DESCRIPCIÓN** | **CLAVE** |
| user | user\_id | int | Id autogenerado para un usuario | PK |
| name | varchar | Nombre del usuario |  |
| email | varchar | Email del usuario |  |
| points | int | Puntos acumulados en el campeonato |  |
| nickname | varchar | Apodo con el que los usuarios se identifican | Único |
| type | int | Si el usuario es jugador o administrador |  |
| amry | army\_id | int | Id autogenerado para un ejército | PK |
| name | varchar | Nombre del ejército | Único |
| faction | varchar | Facción a la que pertenece el ejército |  |
| strategy | varchar | Estrategia en la que se centra el ejército |  |
| player\_id | int | Jugador al que pertenece el ejército | FK |
| game | game\_id | int | Id autogenerado para una partida | PK |
| date | date | Fecha en la que se ha jugado la partida |  |
| winner\_id | int | Id del ejército ganador | FK |
| loser\_id | int | Id del ejército perdedor | FK |

# MANUAL DE USUARIO

# APORTACIONES EXTRAORDINARIAS

1. <https://www.games-workshop.com/es-ES/Warhammer-40-000> [↑](#footnote-ref-1)