Universitat Oberta de Catalunya

uoc.edu

Battle Assistant

Luis Manuel Martín Guerra

Grado en Ingeniería Informática

Java EE

Consultor: Albert Grau Perisé

Profesor/a responsable de la asignatura: Santi Caballe Llobet

10 de enero de 2018

Agradecimientos

A Mamen por su infinita paciencia y apoyo en esta aventura,
A mi madre y mi hermano por estar siempre animándome,
A mis profesores, que siempre me ayudaron y guiaron,
A mi padre, que sin estar, siempre estará conmigo,
A Máximo, por esas largas sesiones de estudio,
A todos los que no dejaron que me rindiese,
Gracias.

FICHA DEL TRABAJO FINAL

Título del trabajo	*Battle Assistant*
Nombre del autor:	*Luis Manuel Martín Guerra*
Nombre del consultor/a:	*Albert Grau Perisé*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	*01/2018*
Titulación o programa:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*TFG - JavaEE*
Idioma del trabajo:	*Castellano*
Palabras clave	*JavaServer Faces, JPA, Java EE*

Resumen del trabajo:

Soy un gran aficionado a los *wargames* desde que era pequeño y siempre me ha sorprendido la cantidad de tiempo que es necesario invertir en la preparación de una partida. Mi wargame favorito es Warhammer 40.000, ambientado en un siniestro futuro distante, allá por el cuadragésimo primer milenio.

Sin embargo la gran cantidad de manuales, suplementos y libros de ejército necesarios para preparar y jugar una partida de este *wargame* hace que haya que invertir mucho tiempo del que normalmente disponemos poco, en realizar esta tarea. Es por ello que he decidido crear una solución a este problema, en forma de una asistente de batallas, que facilite y agilice el proceso de crear y supervisar una partida de Warhammer 40.000.

La metodología que he empleado es la que he aprendido a lo largo de estos años cursando los estudios de Ingeniería Informática y la experiencia que tengo de mi carrera profesional en el mundo de retail, gestionando proyectos.

El resultado de este proyecto es una aplicación completamente funcional, que si bien tiene mucho margen de mejora, para mí es un gran hito, pues supone mi primer proyecto de software desarrollado desde cero y en el que he tenido que recorrer una curva de aprendizaje en las tecnologías empleadas para implementarlo.

Abstract:

I am a big enthusiast of wargames since I was a child and I have always been surprised with the amount of time needed to set a game. My favorite wargame is Warhammer 40.000, settled in the grim darkness of the far future, in the 41st millenium.

However, the large amount number of supplements, rulebooks and army books that are needed to prepare and play a game of this famous wargame suposses a great investment of time. Time that we normally have Little. That is why I have decided to create a solution to this problem, in the form of a battle wizard, that facilitates and speeds up the process of creating and supervising a Warhammer 40,000 game

The methodology that I have used is the one that I have learned throughout these years studying Computer Engineering and the experience I have of my professional career in the Retail world, managing projects.

The result of this project is a fully functional application, which although it has a lot of room for improvement, for me it is a great milestone, since it is my first software project developed from scratch and in which I had to travel a learning curve in the technologies used to implement it. This is the main reason for me to feel proud about it, the learning process in which I have been involved myself during these months.

Índice

1. Introducción

1.1 Contexto y justificación del Trabajo

Los juegos de estrategia con miniaturas o *wargames* me han fascinado desde que los descubrí, tanto que aún a día de hoy, sigo jugando y participando en eventos relacionados con ellos, de manera activa. Desde que empecé a jugar me ha sorprendido y horrorizado a partes iguales la cantidad de detalles y parámetros que hay que tener en cuenta en cada turno, así como la gran cantidad de elecciones y decisiones que hay que tomar antes, incluso de empezar a jugar una partida.

De entre todos los títulos existentes, mi favorito sin duda es Warhammer 40.000. Este wargame está ambientado en el siniestro futuro gótico del cuadragésimo primer milenio. Este trasfondo sitúa a la humanidad amenazada por todas partes por enemigos alienígenas, antiguas razas que poblaron la galaxia y los insidiosos dioses del caos que viven en una dimensión paralela llamada la Disformidad.

Aunque este ambiente futurista y oscuro puede resultar muy atractivo a la hora de librar batallas con ejércitos de miniaturas, el preparar una partida de este *wargame* no lo es tanto. Hay que emplear numerosas tablas de parámetros, repartidas por diversos manuales, para poder configurar adecuadamente una partida. A todo esto hay que añadirle la necesidad de tener a mano los diferentes manuales correspondientes: reglamento, libros de ejércitos, etc.; sin los cuales, los jugadores no pueden tener una referencia de las diversas habilidades, puntos de victoria y eventos que se pueden producir durante el transcurso de una partida estándar.

Esta necesidad de buscar y consultar diferentes manuales y fuentes implica una serie de constantes interrupciones en el fluir de la partida, que obliga a los jugadores a hacer largas pausas y por lo tanto, ralentiza las partidas y corta la emoción del momento. Una solución a esta situación sería la de emplear un asistente de partidas o batallas, que facilitara a los jugadores de Warhammer 40.000 el proceso de crear y configurar una partida y, supervisar y controlar los diferentes parámetros que influyen en la misma: puntos de victoria, puntos de estrategia, etc.

1.2 Objetivos del Trabajo

Este TFG surge como una propuesta para solucionar esta necesidad de simplificar y facilitar el proceso de creación y supervisión de una partida estándar de Warhammer 40.000, y al mismo tiempo como un reto de aprendizaje y desarrollo de las tecnologías vistas en la asignatura de Ingeniería del Software de Componentes y Sistemas Distribuidos, del Grado de Ingeniería Informática.

Desde el punto de vista de los usuarios, la solución que se desarrollará será una aplicación web desde la que, previo registro, podrán configurar todos los parámetros involucrados en una partida del *wargame* Warhammer 40.000. Después de configurar y crear la partida, podrán controlar y monitorizar el desarrollo de la misma, indicándole a la aplicación los diferentes eventos que se suceden, para una vez finalizada la partida, poder consultar los resultados de la misma.

Toda la información que maneje la aplicación será gestionada por un módulo que se encargará de almacenar en una base de datos, la información necesaria para la configuración y gestión de los parámetros de las partidas jugadas, además de la referente a los usuarios que emplean el sistema.

Finalmente, y a modo de objetivo personal, me propongo con la realización de este proyecto, el aumentar mis conocimientos y habilidades sobre las tecnologías del estándar Java EE, el cual al inicio del proceso de desarrollo de este TFG son muy elementales.

1.3 Enfoque y metodología empleados

Dado que actualmente no existen soluciones a la situación descrita anteriormente, y al ser uno de los objetivos de este TFG el desarrollar y afianzar los conocimientos sobre las tecnologías Java EE, se opta por implementar una aplicación desde cero. Esta estrategia facilita, por una parte el proceso de búsqueda y asimilación de conocimientos sobre Java EE y por otra, desarrollar una aplicación que ayude a los jugadores a simplificar el proceso de creación y control de sus partidas.

El proceso de desarrollo del producto software ligado a este TFG se divide en diferentes fases, que se corresponden con las diferentes entregas de material que

había que realizar para la asignatura. Esto permitió establecer un primer marco temporal y al mismo tiempo facilitó una primera asignación de días de trabajo.

Sin embargo, aunque la planificación inicial, como ser verá en el siguiente apartado, se correspondía con un modelo de desarrollo en cascada o lineal, en la fase de implementación del proyecto, para facilitar la integración y los test de los componentes que se desarrollaban, se optó por un modelo de 5 etapas. Este modelo de 5 etapas se compone de “persistencia”, “fachada”, “acciones”, “vista” e “integración”; modelo en el que para cada parámetro del sistema, se desarrolló su correspondiente entidad de persistencia, los métodos de la fachada de su componente, las acciones vinculadas a la fachada, la vista con la que el usuario interactuará y por último, una prueba de integración de todas las etapas para comprobar que el funcionamiento es el esperado.

La adopción de esta metodología facilitó el proceso de detección y depuración de errores, además de simplificar la fase de pruebas. El ir construyendo la aplicación sobre código cuyo funcionamiento ha sido probado y verificado, me animó a perseverar en los momentos en los que creía que no iba a poder completar el producto. Gracias a la adopción de este modelo pude mantener mi nivel de implicación con el proyecto y perseverar mientras recorría la curva de aprendizaje de las tecnologías que empleadas para el desarrollo de la aplicación.

1.4 Planificación del Trabajo

A continuación se encuentran el resumen de las fechas clave de entrega de las actividades y el diagrama de Gantt con la planificación temporal.

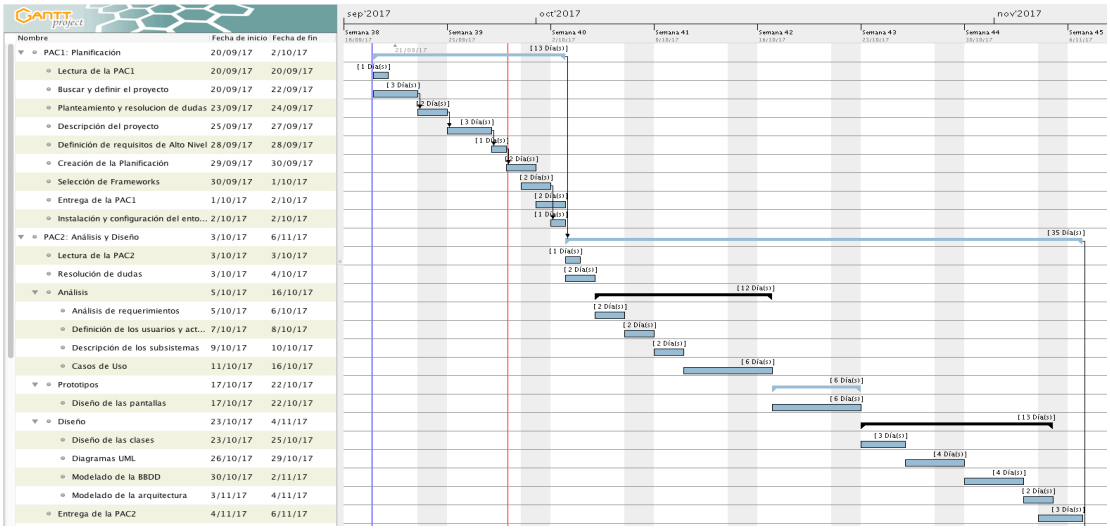
1.4.1. Planificación del Trabajo

Resumen de las fechas de entrega de las actividades relacionadas con nuestro proyecto:

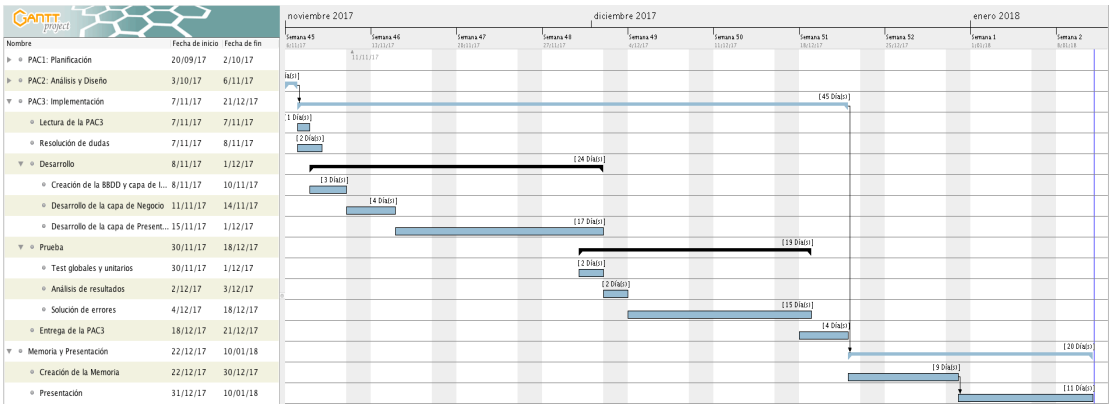
Documento	Inicio	Entrega	Actividad	Duración
PAC1	20/09/2017	02/10/2017	Plan de trabajo	13 días
PAC2	03/10/2017	06/11/2017	Análisis, prototipos y diseño	35 días
PAC3	04/11/2017	21/12/2017	Implementación	45 días
Memoria y presentación	22/12/2017	10/01/2018	Memoria y presentación	11 días

1.4.2. Diagrama de Gantt

Para poder llevar a cabo una correcta planificación del proyecto, definición de hitos y control del proyecto se elaboró el siguiente diagrama de Gantt. Para la elaboración de este, se empleó la herramienta gratuita Gantt Proyect. Por motivos de espacio, se ha dividido en dos partes, que corresponden a diferentes etapas del desarrollo.



1. Planificación de las Etapas PAC1 (Planificación) y PAC2 (Análisis y Diseño).



2. Planificación de las Etapas PAC3 (Implementación) y la Memoria y Presentación del TFG.

1.5 Breve sumario de productos obtenidos

El producto obtenido es una aplicación Java EE en la que se distinguen dos módulos principales: jugador y administrador. Desde el punto de vista de la computación, nos encontramos con aplicación web, que sigue el modelo cliente/servidor, en la que podemos crear y configurar de manera rápida y sencilla partidas del *wargame* Warhammer 40.000.

3. Pantalla de la aplicación con el histórico de partidas de un usuario.

La aplicación obtenida se ha desarrollado con las tecnologías Java EE: JavaServer Faces (JSF), Enterprise Java Beans (EJB) y Java Persistence API (JPA). Se eligieron estas dado que son el estándar de Java EE y que además son las tecnologías de desarrollo de aplicaciones distribuidas que mejor conozco y por tanto considero que tienen las características necesarias para implementar la aplicación.

1.6 Breve descripción de los otros capítulos de la memoria.

Del capítulo 2 al 6 se encuentra la información clave sobre el proyecto y su desarrollo. En estos capítulos se tratarán los requisitos del proyecto, su definición, las fases de análisis y diseño; después analizaremos los prototipos, esquemas y herramientas empleadas, para finalmente abordar la implementación del producto software.

Las conclusiones del proyecto se abordarán en el capítulo 7, en el que se analizará el producto final obtenido y el cumplimento de los objetivos planteados para este TFG.

Por último, encontraremos un glosario de términos en el capítulo 8, la bibliografía empleada en la elaboración de este TFG en el capítulo 9 y un anexo en el capítulo 10, en el que se incluye un pequeño manual de instalación de la aplicación.

2. Análisis de Requisitos

Este capítulo se centra en el análisis en profundidad de los requerimientos de alto nivel que se identificaron en la fase previa de planificación.

Aunque se trate de un proyecto de tipo académico, para reflejar una situación lo más real posible, se llevaron a cabo reuniones con los diferentes jugadores de Warhammer 40.000, que se han empleado como *stakeholders* del proyecto, para detallar y acotar los requisitos de la aplicación. Como resultado de las reuniones se ha tomado la decisión de que la aplicación únicamente contemple las partidas de tipo “equilibrado”, en las que ambos jugadores organizan sus ejércitos en base a un sistema de puntos y los ejércitos son de tipo “Veterano”, es decir que obligatoriamente, deben incluir como mínimo un tipo de formación en su ejército. Además, las misiones incluirán el uso de estratagemas específicas de cada misión y facción, así como el uso de diferentes objetivos tácticos.

Este conjunto de decisiones se tomaron en base a que este tipo de partidas son las más difíciles de monitorizar y a que son el tipo de partida más comúnmente jugado por la amplia comunidad de jugadores de Warhammer 40.000.

A continuación se describen las características principales, los requisitos funcionales y los requisitos no funcionales.

2.1 Características principales

La aplicación que se desarrollará consistirá en una ayuda de juego para las partidas del *wargame* futurista, *Warhammer 40.000.* En un posible escenario, uno de los jugadores, será el encargado de crear la partida en el sistema, indicándole las características de los ejércitos que ambos jugadores emplearán, y el tipo de misión que desean jugar.

Una vez creada la partida, ambos jugadores podrán ver el estado y actualizar el resultado, indicando al sistema los diferentes eventos que se pueden desarrollar durante el transcurso de la partida. Estos eventos son: emplear una estratagema, lograr un objetivo de misión, cumplir con un objetivo táctico, pasar turno, o finalizar la partida.

Una vez finalizada, los jugadores podrán consultar las partidas que hayan monitorizado con la aplicación, pudiendo ver los detalles o incluso eliminándolas del sistema.

Además de los usuarios que empleen la aplicación, el sistema dispone de la figura del administrador, que será el encargado de crear, modificar o eliminar parámetros del sistema y de eliminar usuarios registrados en la aplicación.

2.2 Requisitos funcionales

En este punto se tratarán los requisitos de la aplicación que están relacionados con las funcionalidades que deberá tener. Se utilizan como base las características indicadas por los *stakeholders* en la reunión y se toma la decisión de dividir los requisitos funcionales en diversos grupos, divididos por tipologías.

Estos grupos son los siguientes:

2.2.1. Gestión de Usuarios

- **Alta de usuarios:** deben poder registrarse nuevos usuarios en el sistema. Estos usuarios serán siempre de tipo Jugador.

- **Baja de usuario**: eliminar usuarios del sistema. Únicamente se pueden eliminar los usuarios de tipo Jugador del sistema.

- **Modificar datos de usuario:** los usuarios, tanto Administrador como Jugador, deben poder ver y modificar sus datos personales que tienen registrados en el sistema.

2.2.2. Acceso al sistema

- **Acceso de un usuario:** el sistema debe garantizar que los usuarios únicamente puedan ver las áreas a las que tienen acceso.

- **Abandonar la aplicación:** la aplicación debe permitir a los usuarios desconectarse correctamente, cerrando su sesión.

2.2.3. Gestión de Partidas

- **Crear partidas:** los usuarios Jugadores deben poder crear nuevas partidas en el sistema, así como configurar los parámetros necesarios para jugarlas.

- **Actualizar el estado de una partida:** los jugadores deben poder indicar al sistema los cambios que se vayan produciendo a lo largo del transcurso de la batalla, para que el sistema los aplique y actualice el resultado de la partida.

- **Eliminar una partida**: los usuarios Jugadores deben poder eliminar las partidas que hayan creado.

- **Consultar partidas jugadas:** los usuarios Jugadores deben poder ver un listado con las partidas que han creado y jugado.

- **Ver detalles de una partida:** los usuarios Jugadores deben poder ver los detalles de una partida que hayan jugado y que esté finalizada.

2.2.4. Facciones

- **Alta de una nueva facción:** los usuarios Administradores deben poder dar de alta nuevas facciones en el sistema.

- **Modificar los datos de una facción:** los usuarios Administradores deben poder modificar los datos de una facción existente.

- **Eliminar una facción:** los usuarios Administradores deben poder eliminar del sistema facciones existentes.

- **Consultar facciones:** los usuarios Administradores deben poder ver un listado con todas las facciones existentes en el sistema.

- **Ver detalles de una facción:** los usuarios Administradores deben poder ver los detalles de una facción determinada.

2.2.5. Comandantes

- **Alta de nuevo comandante:** los usuarios Administradores deben poder dar de alta nuevos comandantes en el sistema.

- **Modificar los datos de un comandante:** los usuarios Administradores deben poder modificar los datos de un comandante existente.

- **Eliminar un comandante:** los usuarios Administradores deben poder eliminar comandantes existentes.

- **Consultar comandantes existentes:** los usuarios Administradores deben poder ver un listado con todos los comandantes disponibles.

- **Ver los detalles de un comandante:** los usuarios Jugadores y los Administradores deben poder ver los detalles de un comandante determinado.

2.2.6. Rasgos de Mando

- **Alta de nuevo rasgo de mando:** los usuarios Administradores deben poder dar de alta nuevos rasgos de mando en el sistema.

- **Modificar los datos de un rasgo de mando:** los usuarios Administradores deben poder modificar los datos de un rasgo de mando existente.

- **Eliminar un rasgo de mando:** los usuarios Administradores deben poder eliminar rasgos de mando existentes.

- **Listado de rasgos de mando disponibles:** los usuarios Administradores deben poder ver un listado con todos los rasgos de mando existentes en el sistema.

- **Ver los detalles de un rasgo de mando:** los usuarios Jugadores y Administradores deben poder ver los detalles de un rasgo concreto.

2.2.7. Formaciones

- **Alta de una nueva formación:** los usuarios Administradores, deben poder dar de alta nuevas formaciones en el sistema.

- **Modificar los datos de una formación:** los usuarios Administradores, deben poder modificar los datos de una formación existente.

- **Eliminar una formación:** los usuarios Administradores deben poder eliminar formaciones del sistema.

- **Listado de formaciones:** los usuarios Administradores, deben poder ver un listado con todas las formaciones.

- **Ver los detalles de una formación:** los usuarios Jugadores y Administradores deben poder ver los detalles de una formación.

2.2.8. Misiones

- **Alta de una nueva misión:** los usuarios Administradores deben poder dar de alta nuevas misiones en el sistema.

- **Modificar los datos de una misión:** los usuarios Administradores deben poder modificar los datos de una misión existente.

- **Eliminar una misión:** los usuarios Administradores deben poder eliminar misiones el sistema.

- **Listado de misiones disponibles:** los usuarios Administradores deben poder ver un listado con todas las misiones disponibles en el sistema.

- **Ver los detalles de una misión:** los usuarios, Jugadores y administradores, deben poder ver los detalles de una misión disponible en el sistema.

2.2.9. Objetivos

- **Alta de un nuevo objetivo:** los usuarios Administradores deben poder dar de alta nuevos objetivos en el sistema.

- **Modificar los datos de un objetivo:** los usuarios Administradores deben poder modificar los datos de un objetivo existente.

- **Eliminar una objetivo:** los usuarios Administradores deben poder eliminar objetivos del sistema.

- **Listado de objetivos:** los usuarios Administradores, deben poder ver un listado con todos los objetivos disponibles en el sistema.

- **Ver los detalles de un objetivo:** los usuarios, Jugadores y Administradores, deben poder ver los detalles de un objetivo disponible en el sistema.

- **Cumplir Objetivo:** los Jugadores deben poder indicar al sistema han conseguido un objetivo, en cualquier momento de la partida.

2.2.10. Estratagemas

- **Alta de una nueva estratagema:** los usuarios Administradores deben poder dar de alta nuevas estratagemas en el sistema.

- **Modificar los datos de una estratagema:** los usuarios Administradores deben poder modificar los datos de una estratagema existente.

- **Eliminar una estratagema:** los usuarios Administradores deben poder eliminar estratagemas existentes en el sistema.

- **Ver los detalles de una estratagema:** los usuarios Jugadores y Administradores, deben poder ver los detalles de una estratagema determinada.

- **Listado de estratagemas disponibles:** los usuarios Administradores, deben poder ver un listado con las estratagemas disponibles durante la partida.

- **Emplear una estratagema:** los Jugadores, deben poder indicar al sistema que una determinada estratagema se emplea en un determinado momento de la partida.

2.3 Requisitos no funcionales

En este apartado, se describen los requisitos que el proyecto debe cumplir, para garantizar aquellos aspectos que no están relacionados con la funcionalidad del mismo.

2.3.1. Requisitos de Presentación: la aplicación debe tener una interfaz clara y sencilla, que evite que el usuario se distraiga con facilidad y que proporcione la información de una manera ordenada y fácil de monitorizar.

2.3.2. Requisitos de Usabilidad y Humanidad: es muy importante que la aplicación sea fácil e intuitiva de emplear y que no requiera de ningún tipo de formación previa para poder usarse.

2.3.3. Requisitos Operacionales y de Entorno: la aplicación debe ser usable en cualquier tipo de navegador web disponible.

2.3.4. Requisitos de Mantenimiento y soporte: la aplicación debe ser fácilmente actualizable, para incorporar nuevos modos de juego, misiones, comandantes, etc., sin necesidad de realizar grandes modificaciones.

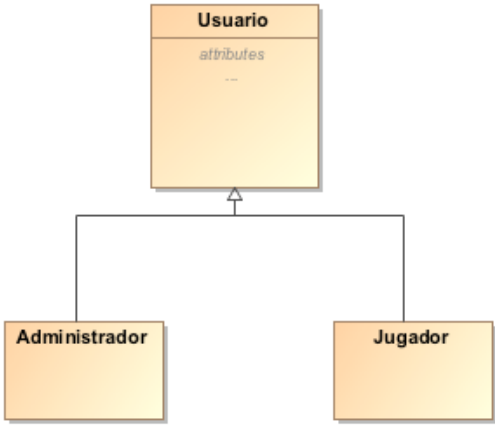
2.3.5. Requisitos de Seguridad: un usuario únicamente deberá tener acceso a sus datos personales y a los referentes a los de sus partidas y sus ejércitos. Además, un usuario que no se encuentre registrado en el sistema, no podrá acceder al mismo.

3. Especificación

En este capítulo se analiza la definición del comportamiento que tiene el sistema implementado. Para ello se hará la definición de los actores que emplearán el sistema, para, teniendo en cuenta los requisitos expuestos en el capítulo anterior, diseñar los diferentes casos de uso y escenarios.

3.1 Definición de actores del sistema

La aplicación tendrá un sistema de seguridad de identificación de usuarios, que separe a los usuarios registrados de los que no. Esto además, servirá para diferenciar usuarios que son Administradores de los que son Jugadores. El siguiente diagrama muestra la definición de los actores del sistema



4. Diagrama de actores del sistema

3.1.1 Usuario Jugador (J)

Es el tipo de actor estándar, no tiene acceso a las funciones administrativas y su ámbito de acción se limita a:

- **Gestión de usuarios:** puede ver y modificar sus datos personales.
- **Gestión de partidas:** puede crear, configurar y jugar nuevas partidas. Además puede consultar su historial de partidas jugadas y eliminar partidas que haya jugado.

3.1.2 Usuario Administrador (A)

Es el tipo de actor tiene acceso a la administración del sistema. Estas funcionalidades administrativas son las siguientes:

- **Gestión de usuarios**: únicamente los administradores tienen permisos para eliminar otros usuarios.

- **Gestión de Facciones**: puede crear, modificar y eliminar facciones del sistema. Además puede consultar las existentes y ver los detalles de las mismas.

- **Gestión de Comandantes**: puede crear, modificar y eliminar Comandantes del sistema. Además puede consultar los existentes y ver los detalles de los mismos.

- **Gestión de Rasgos de Mando**: puede crear, modificar y eliminar Rasgos de Mando del sistema. Además puede consultar los existentes y ver los detalles de los mismos.

- **Gestión de Formaciones**: puede crear, modificar y eliminar Formaciones del sistema. Además puede consultar las existentes y ver los detalles de las mismas.

- **Gestión de Misiones**: puede crear, modificar y eliminar Misiones del sistema. Además puede consultar las existentes y ver los detalles de las mismas.

- **Gestión de Objetivos**: puede crear, modificar y eliminar Objetivos del sistema. Además puede consultar los existentes y ver los detalles de los mismos.

- **Gestión de Estratagemas**: puede crear, modificar y eliminar Estratagemas del sistema. Además puede consultar las existentes y ver los detalles de las mismos.

3.2 Casos de uso

A continuación se propone un listado de los casos de uso que recogen las especificaciones y el comportamiento que debe tener la aplicación. Tomando como punto de partida los requisitos funcionales del proyecto, se han especificado para cada una de ellos diferentes actores: Jugador (J), Usuario no registrado (UNR) y Administrador (A). Para facilitar una documentación intuitiva y fácil de comprender, los casos de uso se han agrupado en determinadas funcionalidades, por ser estas muy similares, evitando así entrar en excesivo detalle.

Los casos de uso son los siguientes:

Identificación

CU01 - Identificarse en el sistema (J,A)

CU02 - Salir del sistema (J,A)

CU03 - Registrarse en el sistema (UNR)

Gestión usuarios

CU04 - Gestionar usuarios (Alta, Baja) (A)

Usuario

CU05 - Modificar datos de usuario (J,A)

CU06 - Crear una nueva partida (J)

CU07 - Gestionar Partidas (Baja, Consulta) (J)

Parámetros del Sistema

CU08 - Gestionar Facciones (Alta, Baja, Modificación, Consulta) (A)

CU09 - Gestionar Comandantes (Alta, Baja, Modificación, Consulta) (A)

CU10 - Gestionar Rasgos de Mando (Alta, Baja, Modificación, Consulta) (A)

CU11 - Gestionar Formaciones (Alta, Baja, Modificación, Consulta) (A)

CU12 - Gestionar Misiones (Alta, Baja, Modificación, Consulta) (A)

CU13 - Gestionar Objetivos (Alta, Baja, Modificación, Consulta) (A)

CU14 - Gestionar Estratagemas (Alta, Baja, Modificación, Consulta) (A)

Partida

CU15 - Actualizar el estado de una partida (J)

CU16 – Usar una estratagema (J)

CU17 – Lograr un objetivo (J)

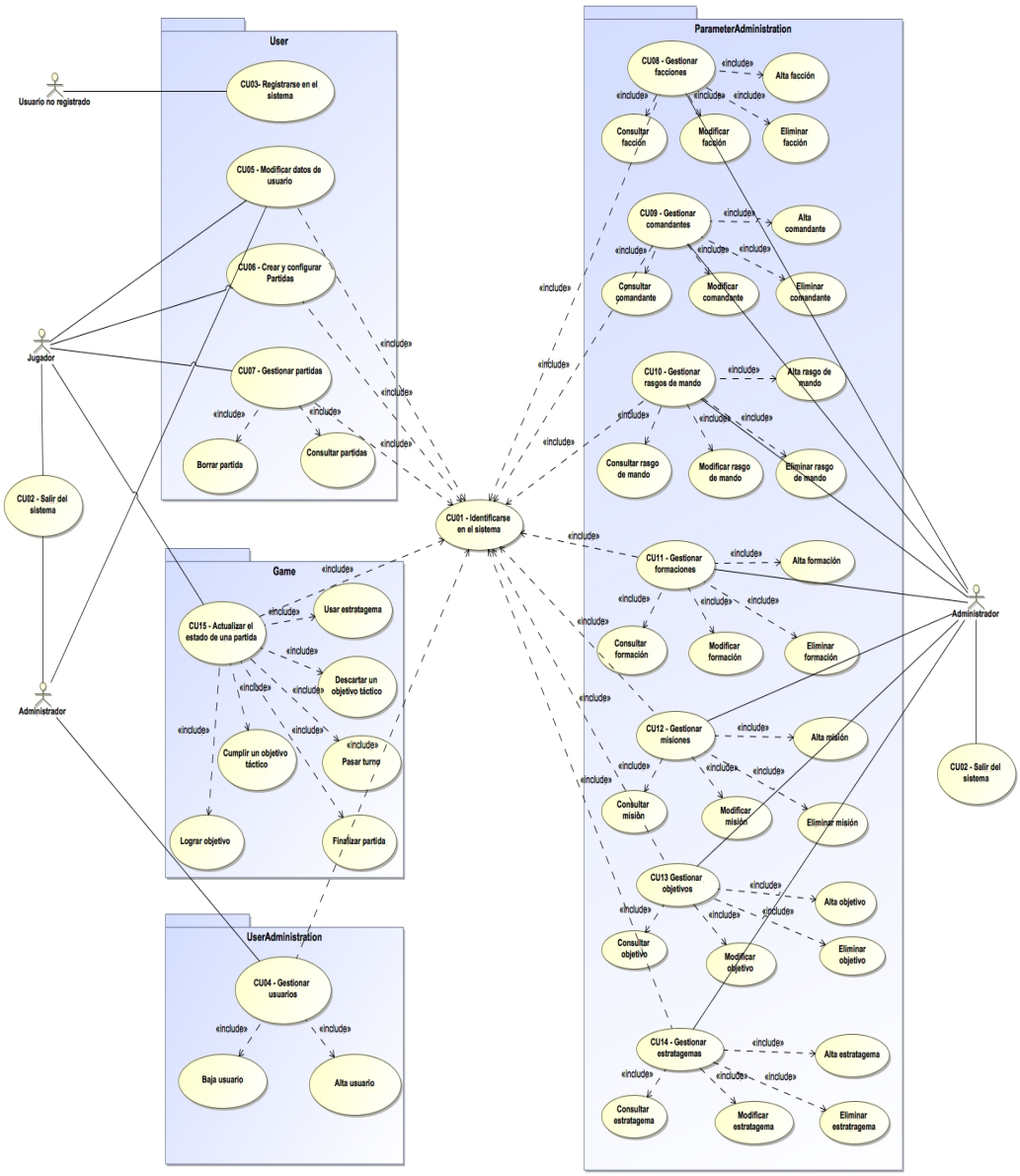
CU18 – Cumplir un objetivo táctico (J)

CU19 – Pasar turno (J)

CU20 – Finalizar partida (J)

3.3 Diagrama de casos de uso

A continuación se propone el diagrama diagrama de casos de uso. Este, muestra las funcionalidades requeridas agrupando los casos de uso anteriormente listados, en paquetes, teniendo en cuenta la funcionalidad que describen.



5. Diagrama de casos de uso de la aplicación.

3.4 Fichas de casos de uso

Finalmente, se realizan las fichas correspondientes al análisis de los diferentes casos de uso de la aplicación. Estas fichas se emplearán como documento de apoyo en la fase de implementación de la aplicación.

Caso de uso: Identificarse en el sistema	
Identificador	CU01
Actor principal	Jugador, Administrador
Ámbito	Sistema
Pre-condición	El usuario debe estar registrado en el sistema (CU03)
Descripción	Un usuario accede al sistema proporcionando su *login* y su contraseña a la aplicación.
Escenario Principal	1. El usuario accede al dominio de la aplicación (URL). 2. El sistema muestra la pantalla de *login*. 3. El usuario rellena los campos de *login* y *password* y pulsa aceptar. 4. El sistema da acceso al usuario y muestra la pantalla principal de la aplicación, en función del tipo de usuario.
Escenario Alternativo	**3.a El usuario rellena los datos de forma errónea o los datos que envía no se encuentran registrados en el sistema**. 3.a.1 El sistema muestra un mensaje de error. 3.a.2 El caso de uso vuelve al punto 2. **3.b El usuario ha olvidado su contraseña y se lo indica al sistema.** 3.b.1 El sistema solicita al usuario su dirección de correo. 3.b.2 El usuario introduce el correo y pulsa enviar. 3.b.3 El sistema informa al usuario de que recibirá un correo con la contraseña. 3.b.4 El sistema envía al usuario un correo electrónico con la contraseña **3.b.4.a El correo electrónico no está registrado en la base de datos de la aplicación.** 3.b.4.a.1 El sistema avisa del error mostrando un mensaje. 3.b.4.a.2 El caso de uso vuelve al punto 2. 3.b.5 El caso de uso vuelve al punto 2.

Caso de uso: Salir del sistema	
Identificador	CU02
Actor principal	Jugador, Administrador
Ámbito	Sistema
Pre-condición	El Jugador o el Administrador deben de haber iniciado sesión en el sistema (CU01)
Descripción	Un usuario sale del sistema.
Escenario Principal	1. El usuario indica al sistema que quiere cerrar su sesión. 2. El sistema cierra la sesión y redirecciona al usuario a la página de *login*.

Caso de uso: Registrarse en el sistema	
Identificador	CU03
Actor principal	Usuario no registrado
Ámbito	Sistema
Pre-condición	Ninguna
Descripción	Un usuario no registrado se registra en el sistema.
Escenario Principal	1. El usuario no registrado accede al dominio de la aplicación (URL). 2. El sistema muestra la pantalla de *login*. 3. El usuario no registrado indica al sistema que quiere registrarse en el sistema 4. El sistema muestra un formulario con diferentes campos para que el usuario o rellene con sus datos (*nombre, apellidos, usuario, password, email*). 5. El usuario no registrado rellena los campos del formulario y pulsa enviar. 6. El sistema muestra un mensaje al usuario confirmando su registro, le informa de que su petición están cursándose y lo redirecciona a la página de *login*. 7. El sistema envía un correo electrónico al administrador con los datos de registro que ha indicado el usuario no registrado en el formulario. 8. El Administrador da de alta al usuario en el sistema y le da permisos de usuario (CU04)
Escenario Alternativo	**5.a El usuario rellena los datos de forma errónea o los datos que envía no se encuentran registrados en el sistema**. 5.a.1 El sistema muestra un mensaje de error. 5.a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar Usuarios	
Identificador	CU04
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El usuario administrador da de alta nuevos usuarios o elimina usuarios ya existentes en el sistema.
Escenario Principal	1. El Administrador selecciona el módulo de “gestión de usuarios”. 2. El sistema muestra un listado con los usuarios que hay en el sistema. 3. El Administrador indica al sistema que quiere crear un nuevo usuario. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes (*user, password, email*) con los del nuevo usuario y lo confirma al sistema. 5. El sistema registra al usuario en el sistema y muestra de nuevo el listado con los usuarios existentes en el sistema.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar un usuario.** 3a.1 El sistema muestra un mensaje informando de los datos del usuario que se va a eliminar y solicitando confirmación. 3a.2 El Administrador confirma que desea eliminar el usuario. 3a.3 El sistema elimina el usuario y muestra el listado de usuarios.

Caso de uso: Modificar datos de Usuario	
Identificador	CU05
Actor principal	Administrador, Jugador
Ámbito	Sistema
Pre-condición	El Administrador o el Jugador deben estar identificados en el sistema (CU01)
Descripción	El Administrador o el Jugador modifican sus datos de usuario del sistema.
Escenario Principal	1. Un usuario, (Administrador o Jugador), indica al sistema que quiere modificar sus datos de usuario. 2. El sistema muestra un formulario en modo edición con los datos del usuario. 3. El usuario edita los datos del formulario para realizar los cambios. 4. El sistema muestra un mensaje de confirmación de los datos introducidos y vuelve a la página principal de la aplicación.
Escenario Alternativo	**3a. El usuario rellena mal los datos (no rellena todos los campos o cumple los requisitos).** 3a.1 El sistema muestra un mensaje informando de que se ha producido un error. 3a.2 El caso de uso vuelve al punto 3. **3b. El usuario cancela el proceso de modificación de sus datos.** 3b.1 El sistema muestra la pantalla principal de la aplicación.

Caso de uso: Crear una nueva partida	
Identificador	CU06
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01)
Descripción	El Jugador crea una nueva partida, configurando los parámetros que intervendrán en ella (Misión, Estratagemas, etc.)
Escenario Principal	1. El Jugador selecciona el módulo de “Gestión de partidas”. 2. El sistema muestra un listado con las partidas que el Usuario ha jugado anteriormente. 3. El Jugador indica al sistema que quiere jugar una nueva partida. 4. El sistema muestra un formulario con los diferentes parámetros a configurar para ambos jugadores (Comandantes, Formaciones, Misión) 5. El sistema muestra un resumen de la misión seleccionada y los datos de los ejércitos del Jugador y del Enemigo y solicita confirmación para iniciar la partida. 6. El Jugador confirma que quiere iniciar la partida.
Escenario Alternativo	**3a. El Usuario cancela la creación de una nueva partida** 3a.1 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar partidas	
Identificador	CU07
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01).
Descripción	El Jugador consulta los detalles de una partida que ha jugado anteriormente.
Escenario Principal	1. El Jugador indica al sistema que quiere ver las partidas que ha jugado. 2. El sistema muestra un listado con las partidas que ha jugado el Jugador. 3. El Jugador selecciona una partida del listado. 4. El sistema muestra la información de la partida jugada. 5. El Usuario pulsa el botón de aceptar. 6. El sistema vuelve a la lista de partidas del jugador.
Escenario Alternativo	**5a. El usuario indica al sistema que quiere borrar una partida.** 5a.1 El sistema muestra un mensaje de confirmación. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar facciones	
Identificador	CU08
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevas facciones, modifica existentes o elimina facciones existentes en el sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar las facciones que hay en el sistema. 2. El sistema muestra un listado con las diferentes facciones que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir una nueva facción al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade una nueva facción al sistema y muestra al Administrador un listado con todas las facciones existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar una facción.** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar una facción** 3b.1 El sistema muestra un formulario con los datos de la facción correspondiente en modo edición. 3.b.2 El Administrador modifica los datos de la facción y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar comandantes	
Identificador	CU09
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevos comandantes, modifica existentes o elimina comandantes del sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar los comandantes que hay en el sistema. 2. El sistema muestra un listado con los diferentes comandantes que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir un nuevo comandante al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade un nuevo comandante al sistema y muestra al Administrador un listado con todos los comandantes existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar un comandante.** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar un comandante** 3b.1 El sistema muestra un formulario con los datos del comandante correspondiente en modo edición. 3.b.2 El Administrador modifica los datos del comandante y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar rasgos de mando	
Identificador	CU10
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevos rasgos de mando, modifica existentes o elimina rasgos de mando del sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar los rasgos de mando que hay en el sistema. 2. El sistema muestra un listado con las diferentes rasgos de mandos que hay en el sistema. 3. El Administrador indica al sistema que quiere añadir un nuevo rasgo de mando al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade una nueva facción al sistema y muestra al Administrador un listado con todas las facciones existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar un rasgo de mando** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar un rasgo de mando** 3b.1 El sistema muestra un formulario con los datos del rasgo de mando correspondiente en modo edición. 3.b.2 El Administrador modificar los datos del rasgo de mando y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar formaciones	
Identificador	CU11
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevas formaciones, modifica existentes o elimina formaciones existentes del sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar las formaciones que hay en el sistema. 2. El sistema muestra un listado con las diferentes formaciones que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir una nueva formación al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade una nueva formación al sistema y muestra al Administrador un listado con todas las formaciones existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar una formación.** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar una formación.** 3b.1 El sistema muestra un formulario con los datos de la formación correspondiente en modo edición. 3.b.2 El Administrador modifica los datos de la formación y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar misiones	
Identificador	CU12
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevas facciones, modifica existentes o elimina facciones existentes en el sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar las misiones que hay en el sistema. 2. El sistema muestra un listado con las diferentes misiones que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir una nueva misión al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema una nueva misión al sistema y muestra al Administrador un listado con todas las misiones existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar una misión** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar una misión** 3b.1 El sistema muestra un formulario con los datos de la misión correspondiente en modo edición. 3.b.2 El Administrador modifica los datos de la misión y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar objetivos	
Identificador	CU13
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevas facciones, modifica existentes o elimina facciones existentes en el sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar los objetivos que hay en el sistema. 2. El sistema muestra un listado con los diferentes objetivos que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir un nuevo objetivo al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade un nuevo objetivo al sistema y muestra al Administrador un listado con todos los objetivos existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar un objetivo.** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar un objetivo.** 3b.1 El sistema muestra un formulario con los datos del objetivo correspondiente en modo edición. 3.b.2 El Administrador modifica los datos del objetivo y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Gestionar estratagemas	
Identificador	CU14
Actor principal	Administrador
Ámbito	Sistema
Pre-condición	El Administrador debe estar identificado en el sistema (CU01)
Descripción	El Administrador da de alta nuevas facciones, modifica existentes o elimina facciones existentes en el sistema.
Escenario Principal	1. El Administrador indica al sistema que quiere consultar las estratagemas que hay en el sistema. 2. El sistema muestra un listado con los diferentes estratagemas que existen en el sistema. 3. El Administrador indica al sistema que quiere añadir una nueva estratagema al sistema. 4. El sistema muestra un formulario con diferentes campos para que el Administrador lo rellene con los datos correspondientes. 5. El Administrador rellena todos los campos del formulario y pulsa el botón de crear. 6. El sistema añade una nueva estratagema al sistema y muestra al Administrador un listado con todas las estratagemas existentes.
Escenario Alternativo	**3a. El Administrador indica al sistema que quiere eliminar una estratagema.** 3a.1 El sistema registra los cambios que se han producido. 3a.2 El caso de uso vuelve al punto 2 **3b. El Administrador indica al sistema que quiere modificar una estratagema** 3b.1 El sistema muestra un formulario con los datos de la estratagema correspondiente en modo edición. 3.b.2 El Administrador modifica los datos de la estratagema y pulsa el botón de actualizar. 3.b.3 El sistema registra los cambios que se han producido. 3.b.4 El caso de uso vuelve al punto 2. **5a. El usuario rellena los datos de manera incorrecta (deja campos sin rellenar)** 5a.1 El sistema muestra un mensaje de error. 5a.2 El caso de uso vuelve al punto 2.

Caso de uso: Actualizar el estado de la partida	
Identificador	CU15
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01) y haber creado una partida
Descripción	El Jugador va indicando al sistema los diferentes eventos que se producen en la partida y este actualiza los datos de las variables que monitoriza.
Escenario Principal	1. El Jugador va indicando al sistema las diferentes acciones que van realizando los jugadores a lo largo de una partida: a) Usar una estratagema (CU16) b) Lograr un objetivo (CU17) c) Cumplir un objetivo táctico (CU18) d) Pasar turno (CU19) e) Finalizar la partida (CU20) 2. El sistema va actualizando los datos de la partida

Caso de uso: Usar una estratagema	
Identificador	CU16
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01), y haber creado la partida (CU06)
Descripción	El Jugador indica al sistema que uno de los jugadores desea emplear una estratagema.
Escenario Principal	1. El Jugador indica al sistema que un jugador quiere emplear una estratagema. 2. El sistema muestra un listado con las estratagemas disponibles para un determinado jugador en la misión correspondiente. 3. El Jugador selecciona la estratagema que desea emplear uno de los jugadores. 4. El sistema muestra un mensaje de confirmación y actualiza el estado del marcador. 5. El caso de uso acaba.
Escenario Alternativo	**3a. El jugador que selecciona una estratagema para la que no tiene puntos suficientes para usar.** 4a.1 El sistema muestra un mensaje de error 4a.2 El caso de uso pasa al punto 2.

Caso de uso: Lograr un objetivo	
Identificador	CU17
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01), y haber creado la partida (CU06)
Descripción	El Jugador indica al sistema que uno de los jugadores ha conseguido un objetivo de la misión que están jugando.
Escenario Principal	1. El Jugador indica al sistema que un jugador ha completado un objetivo. 2. El sistema muestra un listado con objetivos a cumplir de la misión correspondiente. 3. El Jugador indica el objetivo que se ha cumplido. 4. El sistema muestra un mensaje de confirmación y actualiza el estado del marcador. 5. El caso de uso acaba.

Caso de uso: Cumplir un objetivo táctico	
Identificador	CU18
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01), y haber creado la partida (CU06)
Descripción	El Jugador indica al sistema que uno de los jugadores ha logrado completar un objetivo táctico de los generados.
Escenario Principal	1. El Jugador indica al sistema que un jugador ha logrado un objetivo táctico. 2. El sistema muestra un listado con los objetivos tácticos de el jugador concreto. 3. El Jugador indica el objetivo táctico que se ha cumplido. 4. El sistema muestra un mensaje de confirmación, genera un nuevo objetivo táctico y actualiza el estado del marcador. 5. El caso de uso acaba.

Caso de uso: Pasar turno	
Identificador	CU19
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01), y haber creado la partida (CU06)
Descripción	El Jugador indica al sistema que el cambio de turno de la partida.
Escenario Principal	**1.** El Jugador indica al sistema que se produce un cambio de turno. **2.** El sistema muestra un mensaje de confirmación y actualiza el estado del marcador. **3.** El caso de uso acaba.
Escenario Alternativo	**2a. La partida ha alcanzado el máximo de turnos de juego** 3a.1 El sistema muestra un mensaje informando de que la partida ha finalizado. 3a.2 El Jugador pulsa el botón de aceptar. 3a.3 El sistema muestra información del resultado de la partida. 3a.4 El Jugador pulsa el botón de aceptar. 3a.5 El caso de uso acaba.

Caso de uso: Finalizar partida	
Identificador	CU20
Actor principal	Jugador
Ámbito	Sistema
Pre-condición	El Jugador debe estar identificado en el sistema (CU01), y haber creado la partida (CU06)
Descripción	El Jugador indica al sistema que la partida ha finalizado.
Escenario Principal	1. El Jugador indica al sistema que la partida ha finalizado. 2. El sistema muestra un mensaje, pidiendo confirmación al usuario. 3. El Jugador pulsa continuar. 4. El sistema muestra información con los resultados de la partida. 5. El usuario pulsa el botón de aceptar. 6. El caso de uso acaba.
Escenario Alternativo	**3a. El jugador cancela la finalización de la partida.** 3a.1 El caso de uso acaba.

4. Prototipos

Este capítulo introduce los prototipos de las vistas o pantallas que se crearon y que supusieron un primer borrador de las mismas. Su función es la de proporcionar una guía visual de diseño a la hora de implementar dichas vistas. Al tratarse de prototipos, se espera que el aspecto final pueda diferir del presentado aquí.

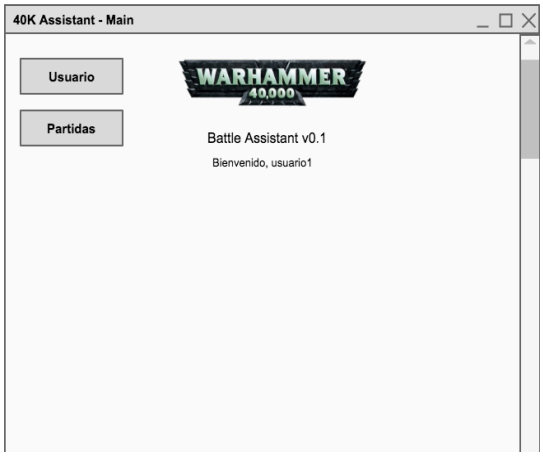
Otra de sus funciones será la de reducir el tiempo de implementación de las vistas. Al tener previsto y planificado el diseño de las mismas, se ahorrará un tiempo considerable a la hora de realizarlas, con lo cual que se podrá avanzar más rápidamente en la fase de implementación.

A continuación se muestran los prototipos realizados:

4.1. Login

4.1.2 Registro de nuevos usuarios

4.2. Pantalla principal USUARIO



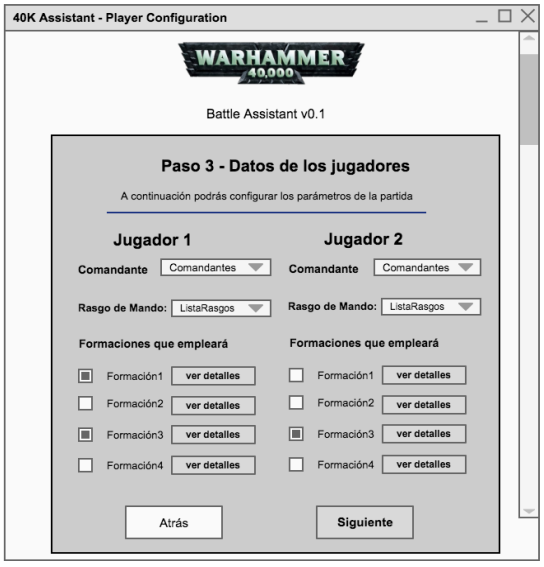
4.2.1. Datos de usuario

4.2.2. Modificar datos de usuario

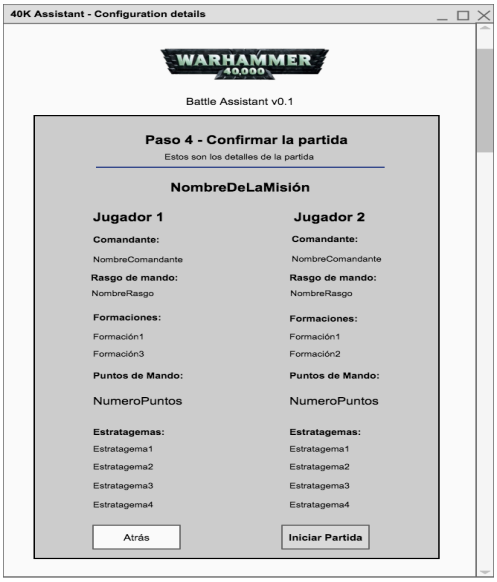
4.3. Listado de partidas jugadas

4.3.1. Detalle de partida jugada

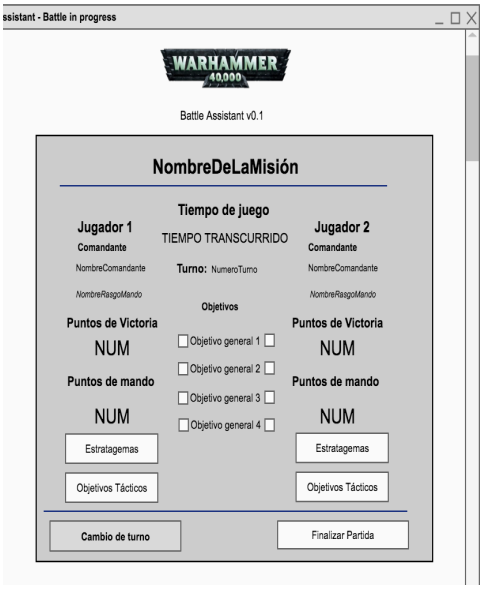
4.4. Nueva partida – Jugadores



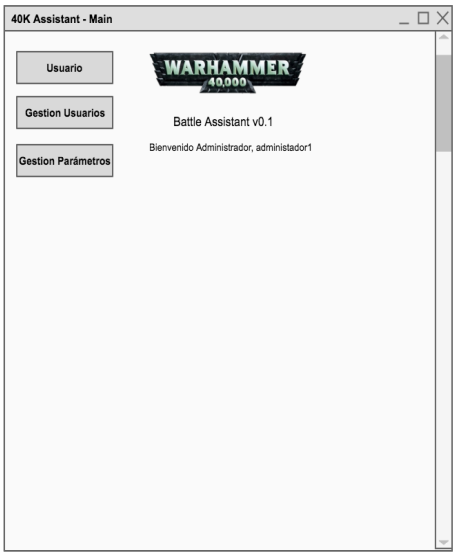
4.5. Nueva Partida – Detalles



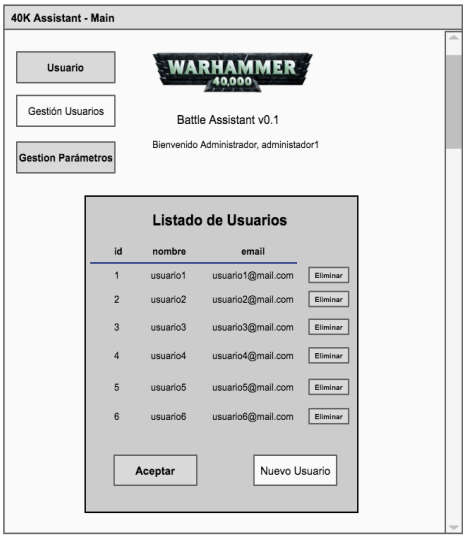
4.6. Partida en curso



4.7. Pantalla principal ADMINISTRADOR

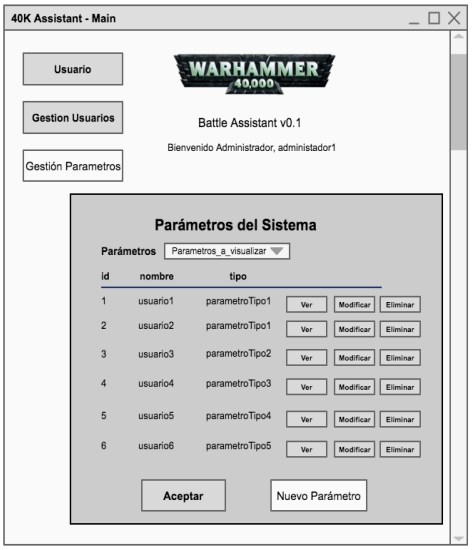


4.7.1 Listado de Usuarios



4.7.2 Crear nuevo Usuario

4.8 Listado Parámetros del sistema



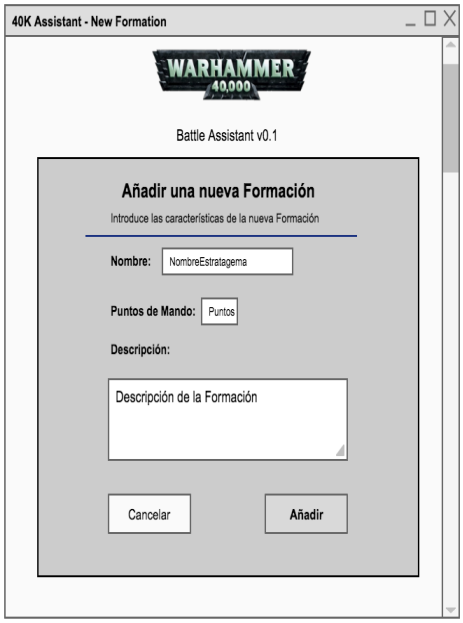
4.8.1 Crear nuevo Parámetro

4.8.1.1 Nuevo Comandante

4.8.1.2 Nueva Facción

4.7.1.3 Nuevo Rasgo de Mando

4.7.1.4 Nueva Formación

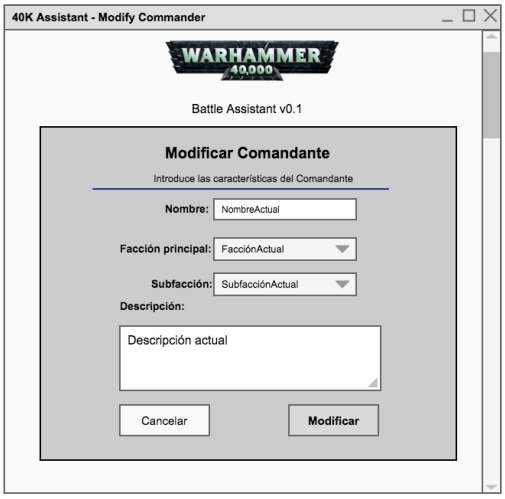


4.7.1.5 Nueva Misión

4.7.1.6 Nueva Estratagema

4.7.1.8 Nuevo Objetivo Táctico

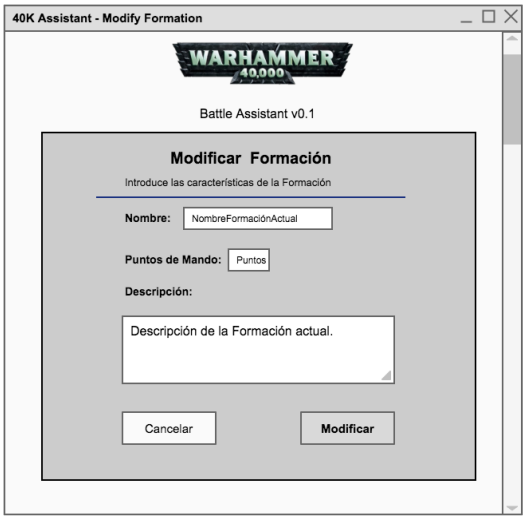
4.7.2.1 Modificar Comandante



4.7.2.2 Modificar Facción

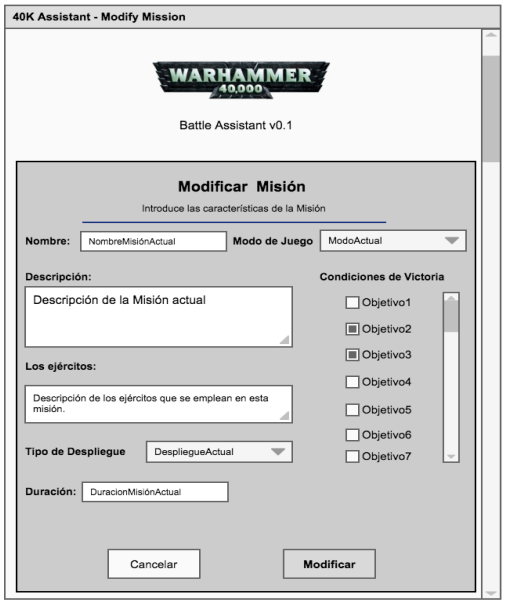
4.7.2.3 Modificar Rasgo de Mando

4.7.2.4 Modificar Formación



4.7.2.5 Modificar Estratagema

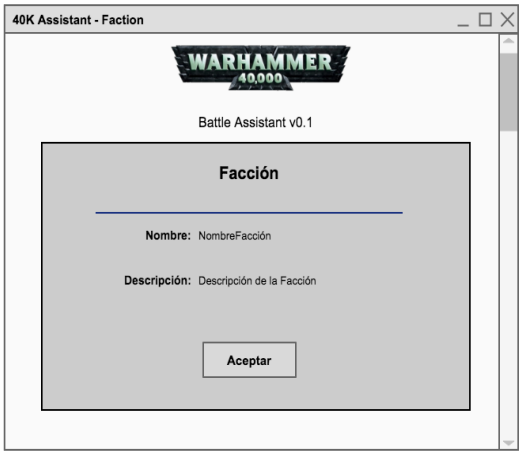
4.7.2.6 Modificar Misión



4.7.2.7 Modificar Objetivo Táctico

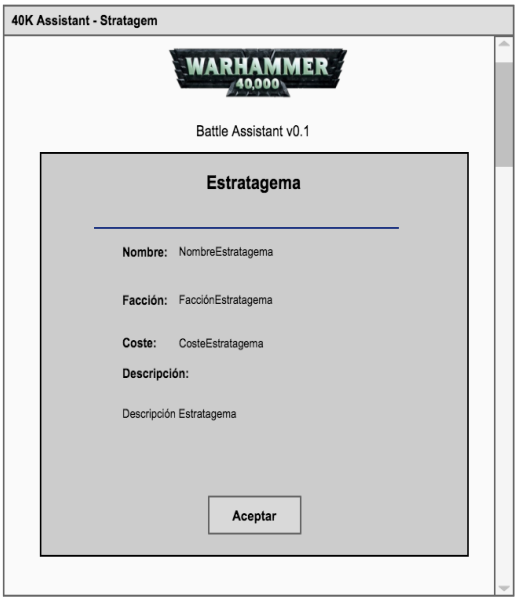
4.7.3.1 Ver Comandante

4.7.3.2 Ver Facción

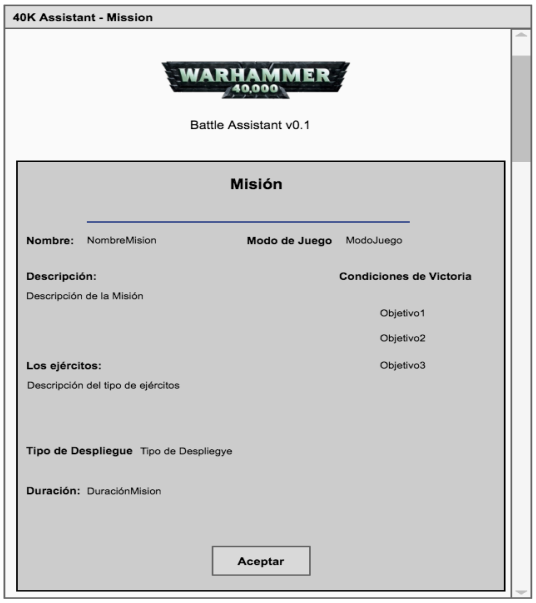


4.7.3.3 Ver Rasgo de Mando

4.7.3.4 Ver Estratagema

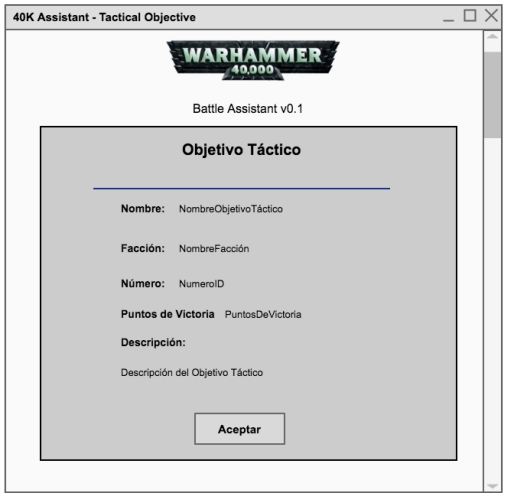


4.7.3.5 Ver Misión



4.7.3.6 Ver Formación

4.7.3.7 Ver Objetivo Táctico

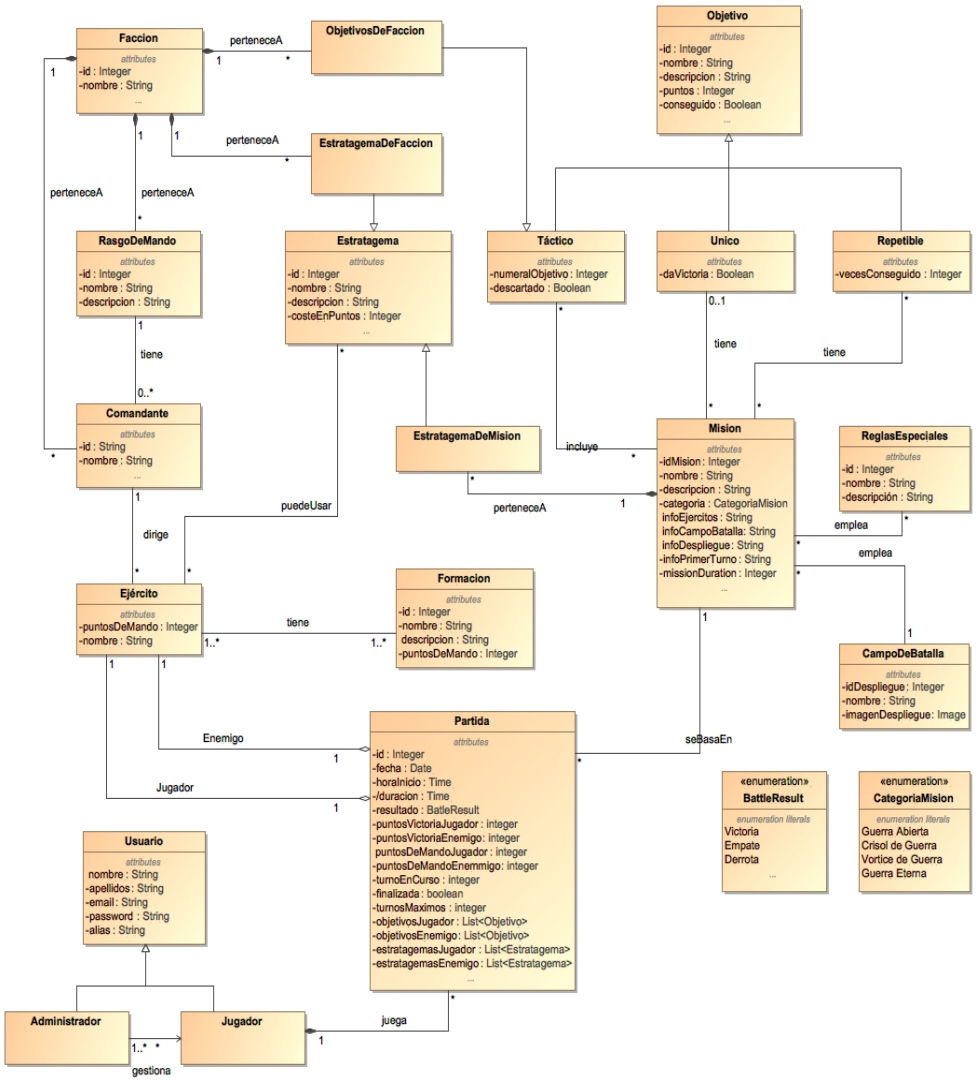


5. Diseño

En este capitulo analizamos los diferentes diagramas que se han elaborado para que, junto con el resto de elementos que hemos tratado en capítulos anteriores, nos sirvan de estructura y guía en la fase de implementación. Los diferentes diagramas que se muestran a continuación se crearon a partir de las especificaciones e información que se obtuvo en las las fases de Análisis y Diseño, siempre teniendo en cuenta las tecnologías que se iban a emplear en la realización de aplicación.

5.1 Diagrama de clases principal

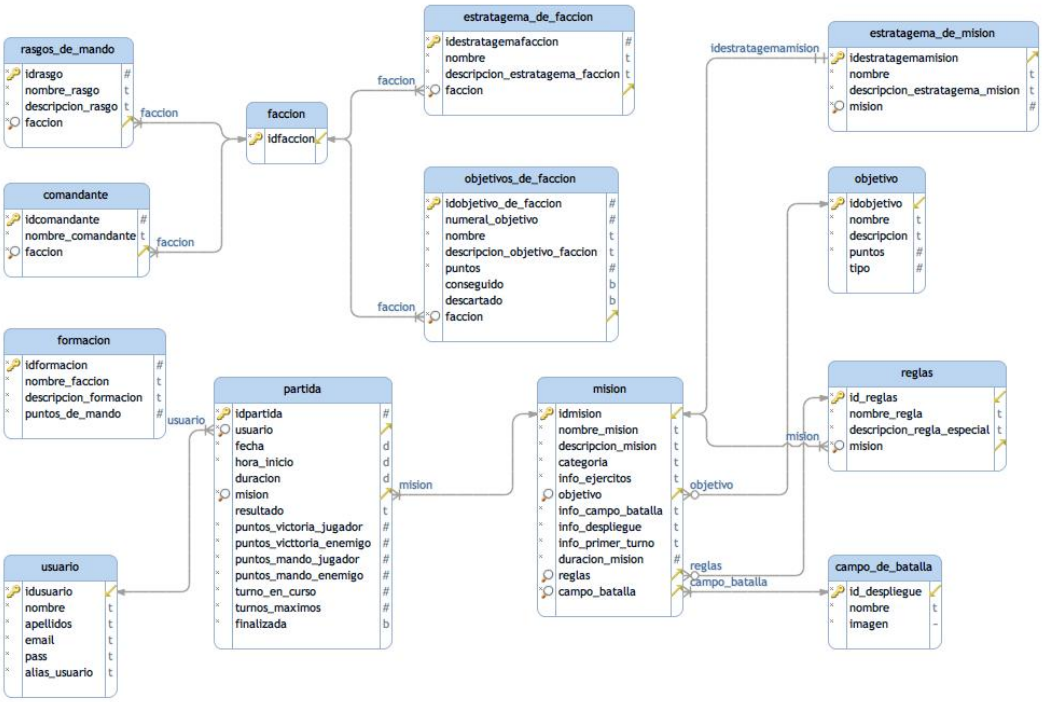
Tomando como punto de partida los requerimientos y el diagrama de casos de uso, se elabora el siguiente diagrama de clases correspondiente:



6. Diagrama de clases principal.

5.2 Diseño relacional de la base de datos

Una vez definidos los requisitos, el diagrama de casos de uso y el diagrama de clases, pasamos al modelado del conjunto de datos que se guardarán de manera persistente y que empleará el sistema. Como resultado se obtiene el siguiente esquema de diseño relacional de la base de datos:



7. Modelo relacional de la base de datos.

5.3 Modelado de la arquitectura de la aplicación

Para modelar la aplicación se empleará un modelo de tipo cliente/servidor, con una arquitectura de 3 capas. En este modelo de arquitectura en 3 capas, la aplicación se divide en capa de presentación, capa de negocio y capa de integración. En la capa de presentación se aplicará el patrón de diseño MVC (Modelo-Vista-Controlador). Este patrón emplea 3 tipos de componentes diferentes: modelos, vistas y controladores. Se emplea este patrón, MVC, para poder tener una interfaz gráfica, en este caso web, desacoplada del resto del sistema. Esto permitirá que, en un futuro, podamos modificar la interfaz gráfica, sin necesidad de alterar el resto de componentes del sistema.

Se emplearemos la tecnología de *JavaServerFaces* (JSF) para la capa de Presentación, siendo el componente que actuará de controlador en esta capa el que

nos proporciona el propio *JSF*, el *FacesServlet*. Los componentes de las vistas, se modelarán con *Facelets* y finalmente los modelos se implementarán mediante el uso de *ManagedBean.*

Para la capa de Negocio, se emplearán *EnterpriseJavaBeans* de sesión, sin estado. También se aplicará el patrón de diseño *Facade* (Fachada) para la capa de Negocio, de esta manera se reducirá el acoplamiento entre la capa de presentación y la capa de negocio y, se simplificará el acceso a los diferentes subsistemas, ofreciendo un único punto de acceso al mismo.

Finalmente, para la capa de Integración se empleará la tecnología de *Java Persistence API* (JPA) en las que las diferentes tablas de la base de datos que emplea la aplicación se modelarán como entidades JPA. También se tendrá tener en cuenta que la capa de integración se conectará a una base de datos *PostgreSQL* mediante el conector JDBC (Java Data Base Conectivity) apropiado.

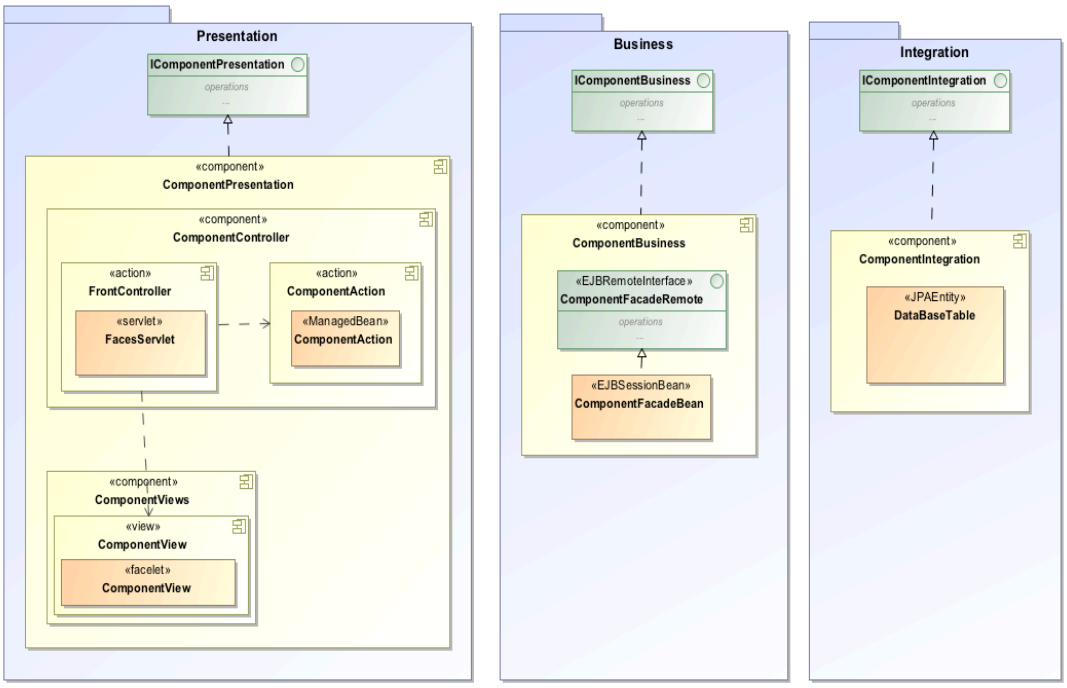
5.4 Diagrama de componentes

La aplicación empleará una arquitectura basada en componentes distribuidos. Dichos componentes, se comunicarán entre sí por medio de unas interfaces que incorporan las firmas de los servicios que ofrece cada uno de los componentes.

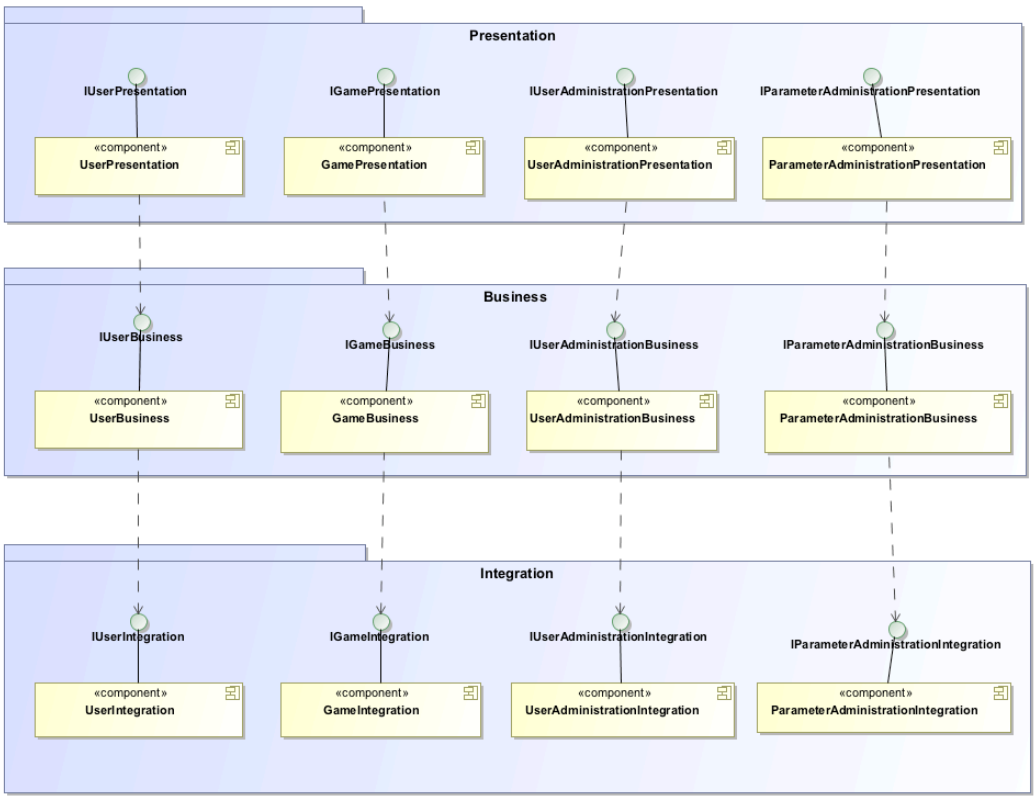
El diagrama de casos de uso visto en el capítulo 3 de esta memoria nos permitía visualizar en componentes las funcionalidades que deberá tener nuestra aplicación y que pueden agruparse en 3 componentes diferentes: ***Game, UserAdministration, y ParameterAdministration***.

Con toda esta información se elaborara un diagrama de componentes genérico, sin especificar el tipo de componente, pero teniendo en cuenta el perfil tecnológico que se va a emplear, Java EE. Este diagrama se utilizará como guía para crear los diferentes componentes que conforman la aplicación

A continuación se muestra el diagrama de componentes de la aplicación, estructurado en las tres capas de la arquitectura y con los diferentes componentes e interfaces que la conforman:



8. Estructura de capas genérica de un componente de la aplicación.



9. Diagrama de componentes de la aplicación.

6. Implementación

Este capítulo contiene toda la información y los detalles relativos a la fase de implementación del producto software que acompaña a este TFG. En él se expondrán las herramientas que se emplearon para su construcción y se irá detallando la implementación de las diferentes capas que componen la aplicación.

6.1 Herramientas software

Nuestra aplicación es un proyecto Java EE, desarrollado a partir de cero. Para su desarrollo se ha empleado el siguiente software:

- **IDE de desarrollo**: Eclipse en su versión *Oxygen* 4.7.0 para desarrolladores Java EE, la versión de JDK que se ha empleado es la 1.8.

- **Base de datos**: los datos físicos se almacenarán una base de datos PostgreSQL, en su versión 9.6.5-1. Como herramienta visual para gestionar la base de datos se empleó PGAdmin, en su versión 4, que vienen incluida dentro del paquete de instalación de PostgreSQL.

- **Servidor de aplicaciones web**: el servidor elegido es JBoss en su versión Wildfly (10.1) como servidor en el que desplegar la aplicación y poder probar y testear su funcionamiento.

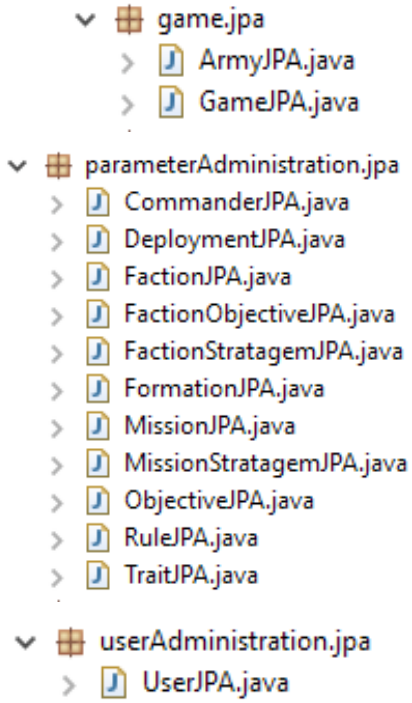
En el anexo localizado al final de este documento, se encuentra un manual de instalación y configuración de las herramientas empleadas para la el desarrollo de la aplicación.

6.2 Análisis de la capa de integración

La capa de datos o de integración, se ha implementado utilizando el *framework* de persistencia de datos recomendado por el estándar de Java EE, el Java Persistence Api o JPA.

Las clases java que componen la capa de datos se han modelado empleando las anotaciones correspondientes, que las identifican como entidades de una base de datos y describen sus campos y relaciones con otras entidades.

Debido a que los diferentes componentes que constituyen la aplicación acceden y emplean entidades JPA diferentes, se han agrupado en tres paquetes diferentes: game.jpa, parameterAdministration.jpa y userAdministration.jpa



10. Clases java que modelan las entidades de la base de datos.

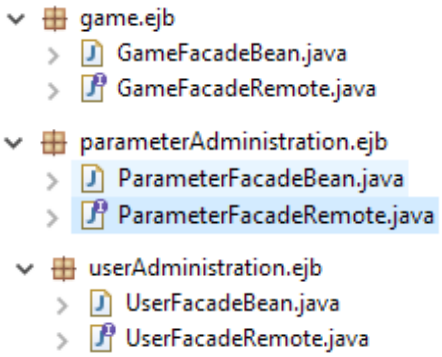
6.3 Análisis de la capa de negocio

La capa de negocio se ha implementado tal y como hemos comentando en otros capítulos mediante el uso de los Enterprise Java Beans, dado que además de ser el estándar recomendado por Java EE, es la tecnología de la que más conocimientos teníamos en el momento de iniciar el proyecto.

Cada capa de negocio, de cada componente, está formada por una interfaz remota y un Enterprise JavaBean de sesión sin estado, ya que en base a los requisitos de la aplicación no era necesario emplear uno con estado.

La interfaz remota, contiene la firma de los métodos que exponen los componentes y que pueden ser invocados por otros componentes a su vez, mientras que el EJB (Enterprise Java Beans) contiene la implementación de la lógica de negocio que será invocada desde otros componentes.

Por lo tanto, cada componente tiene una interfaz remota y un EJB que la implementa, como podemos ver a continuación:



11. Clases java que modelan la capa de negocio de los componentes.

Los métodos implementados por cada uno de estos componentes de negocio son los siguientes:

6.3.1 Componente ParameterFacade

```
public Collection<FactionJPA> listAllFactions();

public void addFaction(String name);

public void updateFaction(int id, String newFactionName);

public void deleteFaction(int factionId);

public FactionJPA getFaction(int factionId);

public Collection<TraitJPA> listAllCommandTraits();

public void addTrait (String name, String description, String faction);

public void updateTrait (int id, String newTraitName, String newDescription,
String newFaction);

public void deleteTrait (int traitId);

public TraitJPA getTrait (int traitId);

public Collection<CommanderJPA> listAllCommanders();

public CommanderJPA getCommander(int commanderId);

public void addCommander(String name, String faction, String trait);

public void updateCommander(int id, String name, String faction, String
trait);


public void deleteCommander(int commanderId);
```

```
public Collection<FactionObjectiveJPA> listAllFactionObjectives();

public Collection<FactionObjectiveJPA> getFactionObjectivesByFaction(int
factionId);

public void addFactionObjective (String name, String description, int points,
int number, String factionName);

public void updateFactionObjective (int id,  String name, String description,
int points, int number, String factionName);

public void deleteFactionObjective (int factionObjectiveId);

public FactionObjectiveJPA getFactionObjective(int factionObjectiveId);


public Collection<FactionStratagemJPA> listAllFactionStratagems();

public Collection<FactionStratagemJPA>

getFactionStratagemsByFaction(int factionId);

public void addFactionStratagem (int cost, String description, String name,
String faction);

public void updateFactionStratagem (int factionStratagemId,  int cost, String
description, String name, String faction);

public void deleteFactionStratagem (int factionStratagemId);

public FactionStratagemJPA getFactionStratagem(int factionStratagemId);

public List<FormationJPA> listAllFormations();

public void addFormation (String name, String description, int
command_points);

public void updateFormation (int formation_id, String newName, String
newDescription, int newPoints);

public void deleteFormation (int formationId);

public int calculateCommandPoints(List<String> formations);

public FormationJPA getFormation(int formationId);

public Collection<DeploymentJPA> listAllDeployments();

public void addDeployment (String name, String map);

public void updateDeployment (int id, String newName, String newMap);

public void deleteDeployment (int deploymentId);

public DeploymentJPA getDeployment(int deploymentId);

public Collection <RuleJPA> listAllRules();
```

```
public void addRule (String name, String description);

public void updateRule (int id,  String newName,  String newDescription);

public void deleteRule (int ruleId);

public RuleJPA getRule (int ruleId);


public Collection<ObjectiveJPA> listAllObjectives();

public void addObjective (String name, String description, int points, String
type);

public void updateObjective (int id,  String name, String description, int
points, String type);

public void deleteObjective (int objectiveId);

public ObjectiveJPA getObjective (int objectiveId);


public Collection<MissionJPA> listAllMissions();

public MissionJPA getMissionByName(String missionName);

public MissionJPA getMission(int missionId);

public void addMission (String name, String category, String description,
String armies_info, String deployment_info, String turn_info, String
duration, String deployment, List<String> objectives, List<String> rules);

public void updateMission (int missionId, String name, String category,
String description, String armies_info, String deployment_info, String
turn_info, String duration, String deployment, List<String> objectives,
List<String> rules);

public void deleteMission (int missionId);

public Collection<ObjectiveJPA> getMissionObjectives (int missionId);

public Collection<RuleJPA> getMissionRules (int missionId);

public Collection<MissionStratagemJPA> listAllMissionStratagems();

public Collection<MissionStratagemJPA> getMissionStratagemsByMission(int
missionId);

public void addMissionStratagem (String name, String description, int cost,
Boolean attacker, String mission);

public void updateMissionStratagem (int id,  String newName, String
newDescription, int newCost, Boolean newAttacker, String newMission);

public void deleteMissionStratagem (int missionStratagemId);


public MissionStratagemJPA getMissionStratagem (int missionStratagemId);
```

6.3.2 Componente GameFacade

```
public Collection<GameJPA> listAllGames();

public Collection<GameJPA> gamesByUser(int userId);

public void addGame(String user, String fecha, int missionId, String result,
int playerVictoryPoints, int enemyVictoryPoints, int turn);


public GameJPA getGame(int gameId);

public void deleteGame(int gameId);

public int addArmy(String name, String commanderName, int commandPoints, int
powerPoints);

public void deleteArmy(int armyId);

public void updateArmy(int armyId, String name);

public ArmyJPA getArmy(int armyId);
```

6.3.3 Componente UserFacade

```
public Collection<UserJPA> listAllUsers();

public UserJPA getUser (int userId);

public void addUser (String name, String surname, String email, String
password, String alias, Boolean admin);

public void updateUser (int userId, String email, String password, String
alias);

public void deleteUser (int userId);

public boolean checkMail(String mail);

public UserJPA checkUser(String alias, String password);

public boolean checkAlias(String alias);

public boolean isAdmin(String alias);
```

6.4 Análisis de la capa de presentación

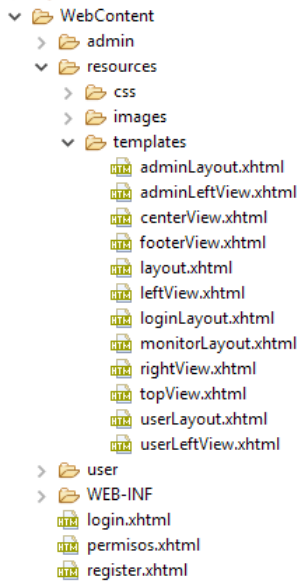
Para la implementación de la capa de presentación, en esta caso, capa Web, se emplearon dos *frameworks*, ambos basados en el estándar de JavaEE.

El primero de ello es JavaServer Faces (JSF), que es el *framework* recomendado por Java EE para el desarrollo de la capa de presentación. JavaServer Faces implementa un patrón de tipo Modelo-Vista-Controlador, tal y como se ha comentado anteriormente, en el que el Controlador lo proporciona el propio *framework*, en este caso es el Faces Server, y las vistas y acciones del modelo se implementan mediante páginas .xhtml y managed beans respectivamente.

Finalmente y con la intención de dotar de mayor funcionalidad a las vistas y facilitar el desarrollo de determinadas acciones, también se empleó el *framework Primefaces*, en su versión 6.1. Este *framework* está desarrollado sobre el estándar de JavaServer Faces e incorpora etiquetas xhtml propias que modelan acciones u objetos en las vistas de una manera sencilla y visualmente atractiva, sin necesidad de emplear demasiado código.

A la hora de construir las correspondientes páginas .xhtml, se crearon una serie de plantillas para cada componente principal de las vistas (cabecera, izquierda, centro, derecha y pie), de manera que se evitó replicar código en cada página y además de ser mucho más sencillas de actualizar y mantener.

Se crearon diferentes plantillas para cada uno de los elementos de los que se componen las vistas, tomando como referencia para la implementación, los prototipos que se realizaron en la fase de diseño y que podemos encontrar en el capítulo 4 de esta memoria.



12. Esquema con las diferentes plantillas de los componentes de las vistas de la aplicación.

Tal y como nos mostraban los prototipos, tanto el jugador como el administrador, tenían en la parte izquierda de la pantalla las opciones disponibles, es por ello que se implementaron dos partes izquierdas diferentes, cada una en función del tipo de usuario.

Una modificación que se llevó a cabo durante la implementación de las vistas, fue la de intentar reducir el número de las mismas, ya que la cantidad de parámetros que controla el sistema y las diferentes operaciones que se podían realizar sobre ellos, disparaban el número de páginas .xhtml a realizar. Se decidió por tanto, sustituir las vistas que detallaban información, o se empleaban para actualizar o borrar datos, por paneles emergentes dentro de la vistas que mostrasen el listado de los parámetros.

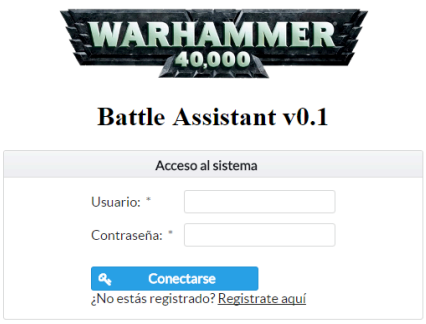
Esta modificación requirió desarrollar un primer modelo base, para después ser replicado en las siguientes vistas, consiguiendo finalmente reducir el número de vistas a codificar.

Para reducir también la gran cantidad de parámetros que debían controlar las acciones del modelo, en concreto, las correspondientes a la creación de una nueva partida y la creación de nuevas misiones, se empleó inyección de dependencias en los correspondientes managed beans, utilizando para ello la anotación `@ManagedProperty`.

6.5 Resultado final

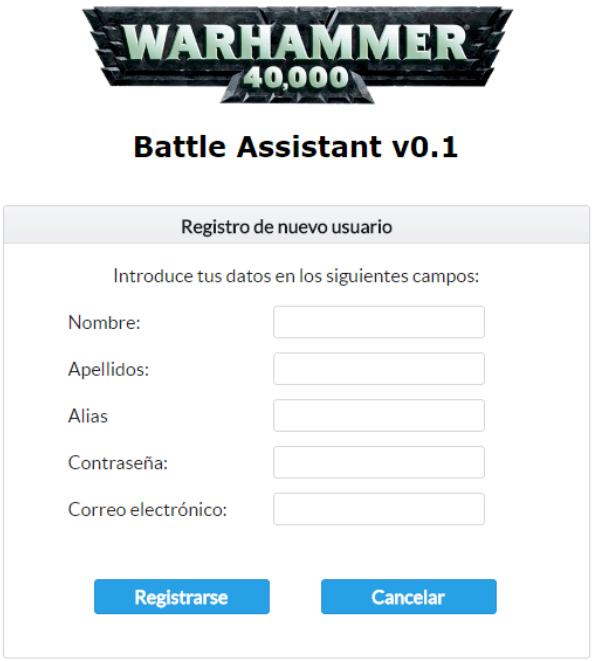
A continuación se muestra el resultado final de la aplicación que se ha desarrollado, con las opciones disponibles, tanto para el jugador como para el administrador.

6.4.1. Pantalla de acceso



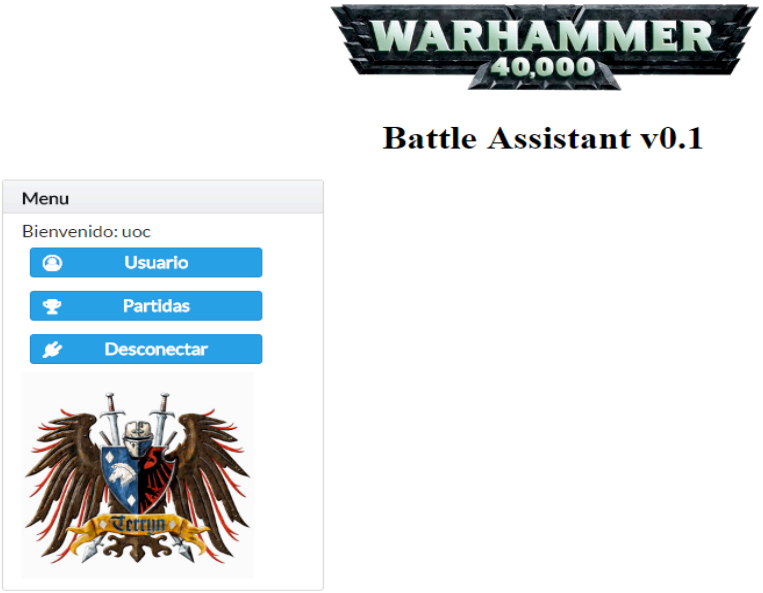
13. Pantalla de acceso o *login* de la aplicación.

6.4.2. Registro de nuevo usuario



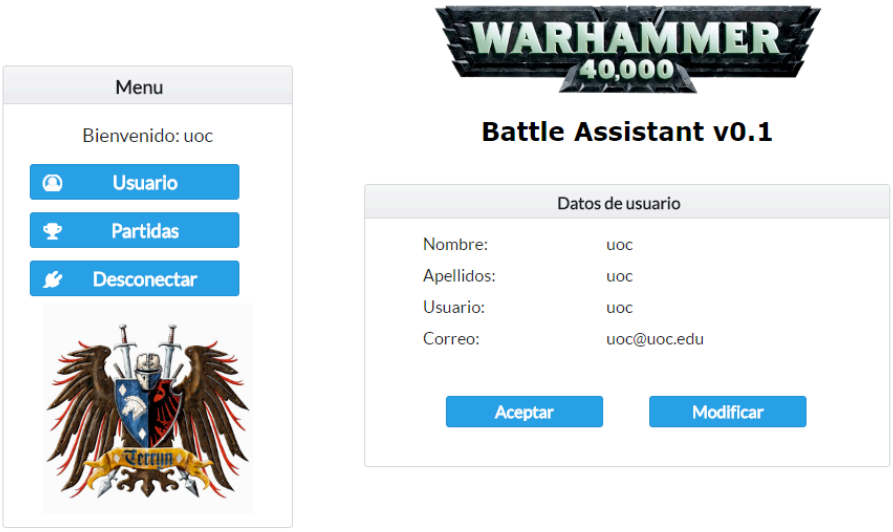
14. Pantalla de registro de un nuevo usuario.

6.4.3. Pantalla principal usuario



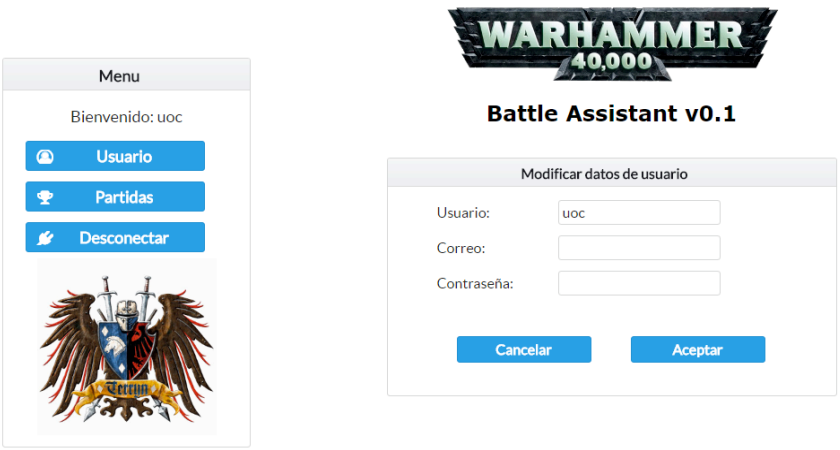
15. Pantalla principal de un usuario identificado correctamente.

6.4.4. Pantalla datos usuario



16. Datos personales de un usuario.

6.4.5. Pantalla modificar datos usuario



17. Pantalla de modificación de los datos de un usuario.

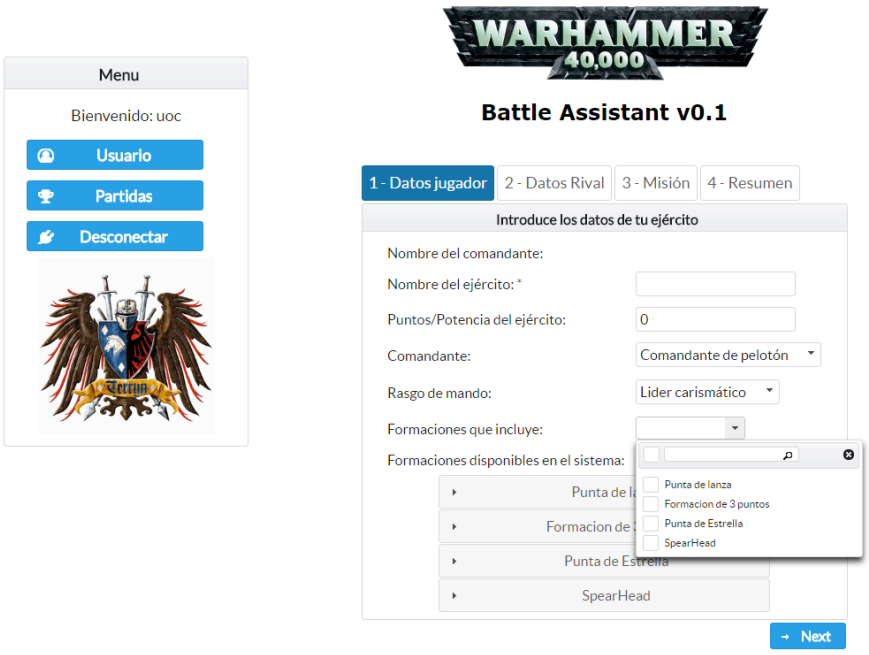
6.4.6. Pantalla listado de partidas jugadas

18. Listado de partidas jugadas por el usuario.

6.4.7. Pantalla detalle partidas jugadas

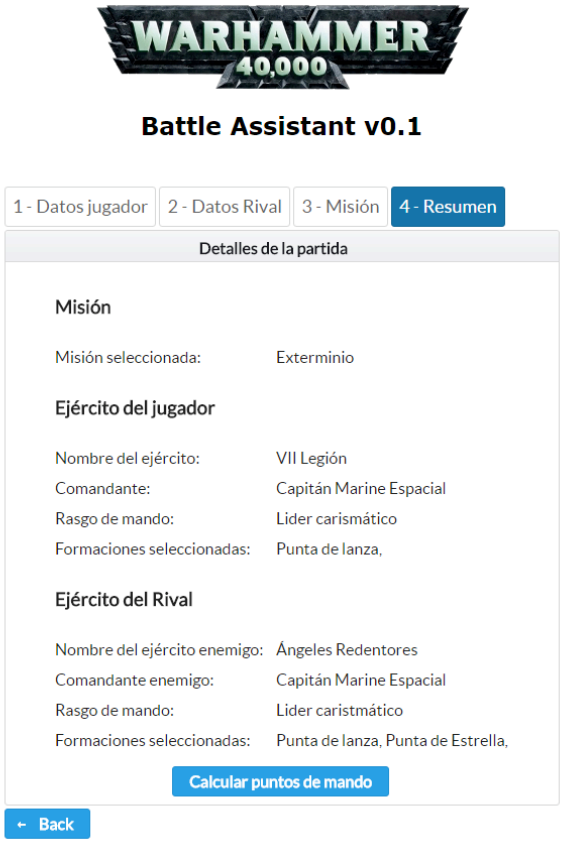
19. Panel con los detalles de una partida jugada.

6.4.8. pantalla nueva partida



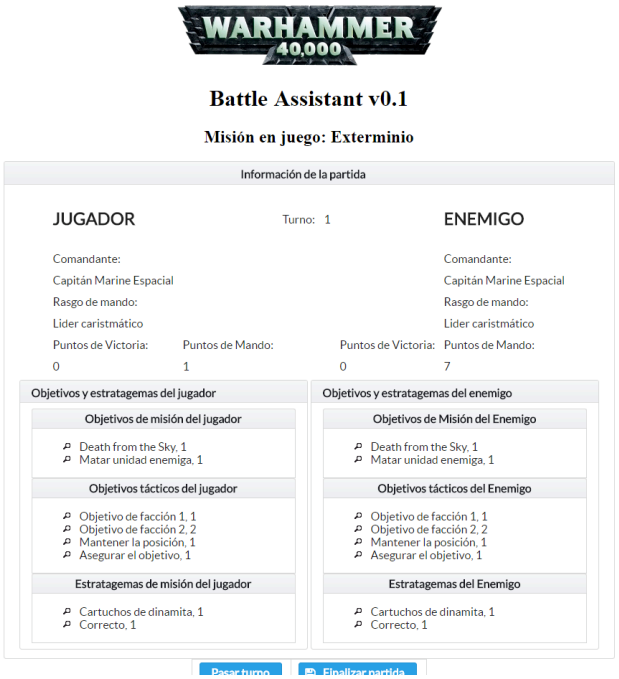
20. Asistente para configurar una nueva partida.

6.4.9. Pantalla resumen configuración partida



21. Resumen con los datos de configuración de la partida.

6.4.10. Pantalla partida en curso

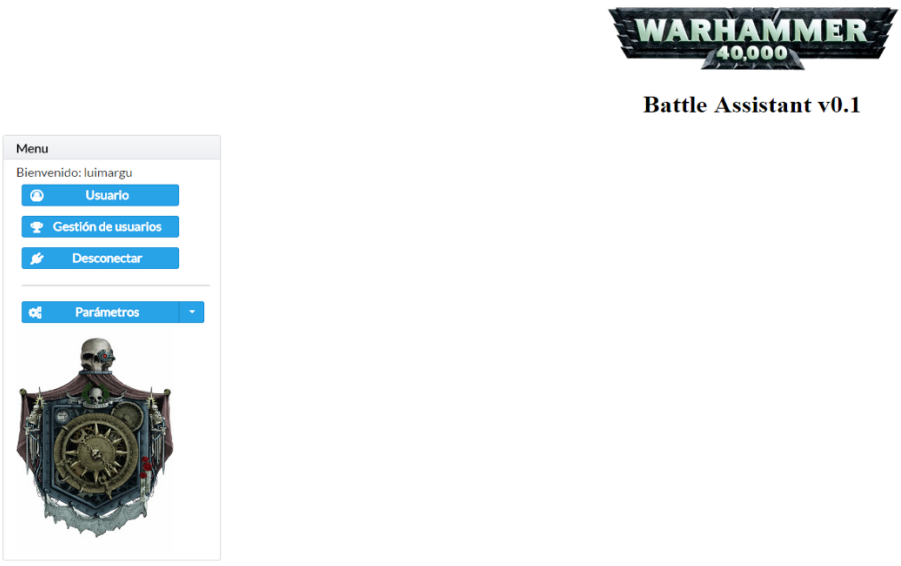


23. Pantalla con la información de control de la partida.

6.4.11. Pantalla detalle uso de objetivos en una partida

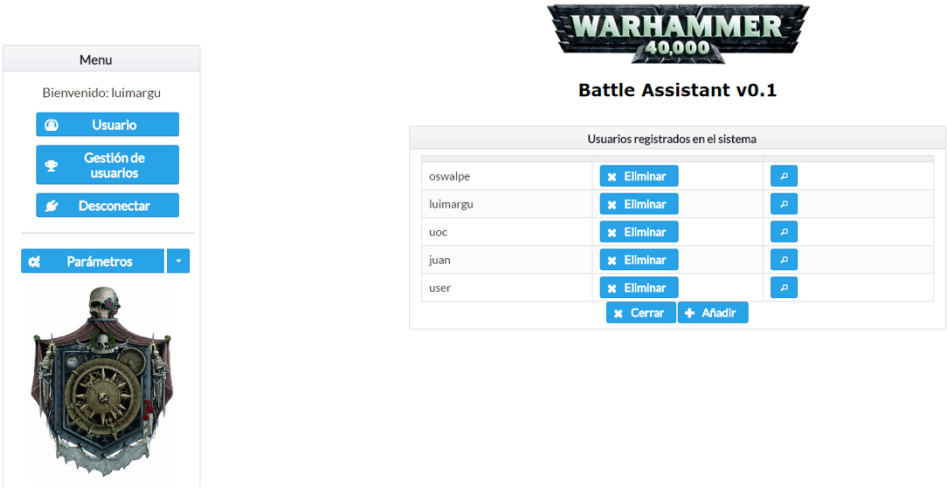
24. Detalle de un objetivo de misión del jugador.

6.4.12. Pantalla principal administrador



25. Pantalla principal de un administrador.

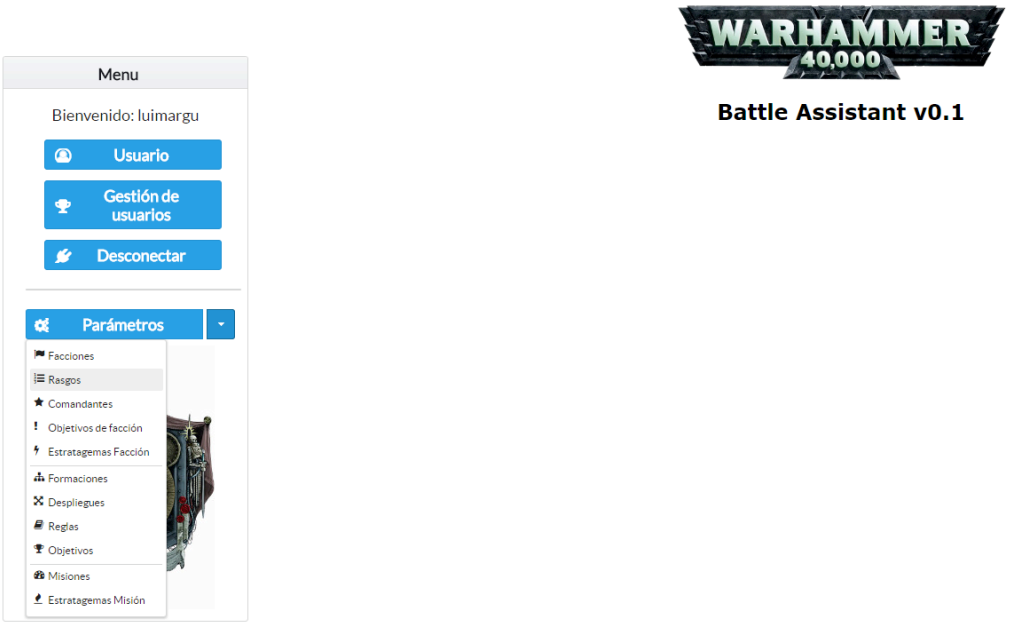
6.4.13. Pantalla listado usuarios del sistema



26. Listado de usuarios registrados en el sistema.

6.4.14. Pantallas relacionadas con los parámetros del sistema

A continuación se encuentra la pantalla principal del administrador con el desplegable de los parámetros activo, mostrándonos el acceso a los mismos:

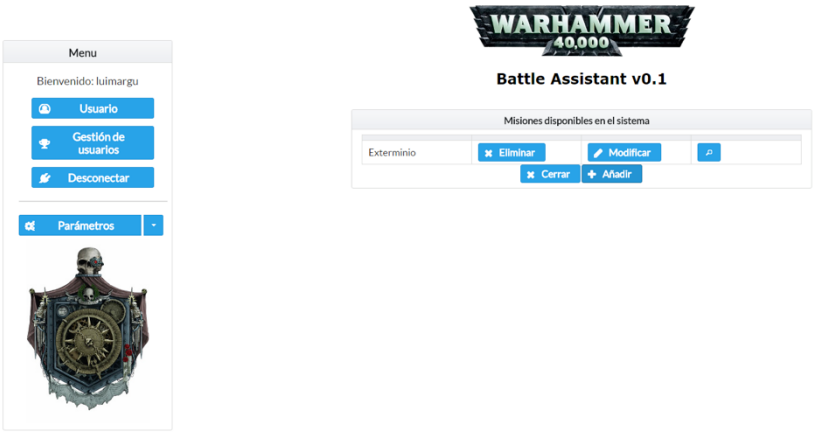


27. Botón desplegable de acceso a los parámetros del sistema.

En la fase de diseño realizada previa implementación de la aplicación, se diseñaron los prototipos de las vistas. Estos primeros prototipos se realizaron teniendo en cuenta los conocimientos y capacidades técnicas con respecto a la tecnología de *JavaServer Faces* que teníamos en aquel momento. No obstante, debido al desarrollo y aprendizaje obtenidos durante la fase de implementación, se optó por por modificar el acceso a los parámetros del sistema. Debido a esto, los parámetros del sistema se

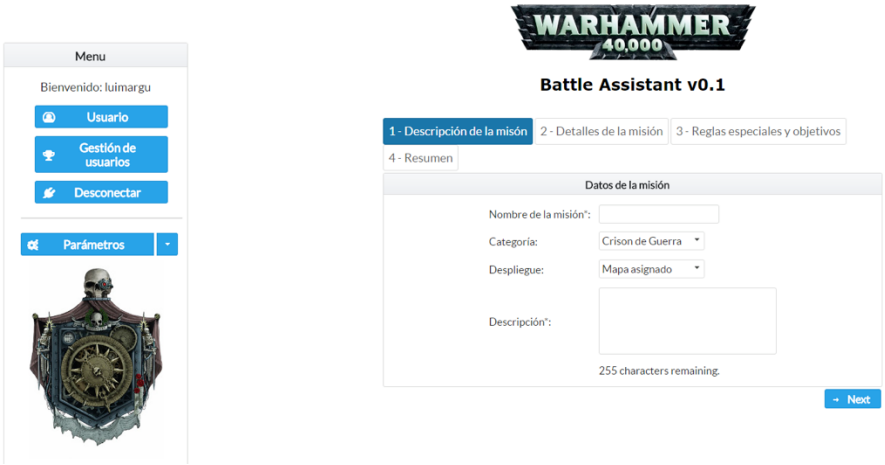
engloban dentro de un mismo botón desplegable que da acceso a aquel que elijamos y que junto con los paneles emergentes permitió reducir el número de vistas necesarias para implementar la aplicación.

Al emplear plantillas *.xhtml* para construir las vistas, las vistas asociadas a los parámetros del sistema son prácticamente idénticas, cambiando únicamente los títulos de las tablas de datos y el nombre de los botones.



28. Listado de misiones disponibles en el sistema.

La única vista de parámetros del sistema, diferente al resto, es la referente a las misiones. Esto se debe a que las misiones que los jugadores seleccionar para jugar, contenían tantos parámetros que fué necesario rediseñar la vista para presentar mejor la información y facilitar la creación de las mismas.



29. Asistente para crear una nueva misión.

6.6 Requisitos no implementados

A pesar de que a la fecha de entrega de este TFG la aplicación es completamente funcional, uno de los requisitos no funcionales que se marcó en la fase de análisis no se ha logrado completamente. Este requisito es el de la seguridad.

En el capítulo 2 se expuso que uno de los requisitos no funcionales era que los usuarios registrados únicamente podrían tener acceso a sus datos personales, a los datos relacionados con sus partidas y a la creación de nuevas partidas.

Aunque a primera vista, si interactuamos con la aplicación, parece actuar de la manera que se espera, no se ha conseguido implementar un mecanismo para impedir que un usuario registrado pueda escribir directamente la dirección de una de las páginas a las que en teoría sólo tiene acceso un administrador, y acceder a ellas. Esto representa una gran vulnerabilidad de seguridad, pues empleando este método un usuario registrado puede acceder al área de administración y eliminar usuarios, parámetros, etc.

Sin embargo, aunque se ha identificado como el método más óptimo el implementar el acceso a los recursos mediante JAAS, el servicio de autentificación del que dispone el estándar Java EE para los servidores de aplicaciones, no se ha logrado hacer que funcione en la distribución de JBoss que elegida para este proyecto. No se ha logrado que obtenga correctamente la información de las tablas de la base de datos para identificar a los diferentes usuarios.

Debido a esto, se optó por implementar el acceso mediante un filtro web. Este filtro, impide que cualquier usuario no registrado pueda acceder a las páginas de la aplicación. Junto a este filtro, se implementó un método de identificación que obliga a introducir un nombre de usuario y contraseña, los contrasta con la información de la base de datos y, en caso satisfactorio redirige al usuario a la zona correspondiente (Jugador o Administrador).

7. Conclusiones

Este proyecto ha supuesto la oportunidad de poner en práctica gran parte de los conocimientos adquiridos a lo largo de la carrera y crear un proyecto software desde cero. A pesar de poder tener una sólida base teórica, el enfrentarme a un reto real, con un proyecto elegido por mí y diseñado completamente desde cero, me ha mostrado cuánto me queda por aprender y descubrir.

Mi experiencia profesional está relacionada con las tiendas, y con la gestión de equipos y proyectos y es pero eso que las fases de planificación y análisis de requisitos me han permitido explotar mis habilidades y competencias en esa área.

A pesar de que las fases iniciales del proyecto han funcionado bastante bien, he podido comprobar que, llegada la fase de implementación la realidad del mundo profesional se hace patente. Mientras reflexiono y escribo esta memoria, pienso que lo que me ha hecho perseverar han sido mis ganas de aprendizaje, los ánimos de mis profesores, y la determinación de poder crear un producto software desde cero.

Como ya he comentado en capítulos anteriores de este TFG mi nivel de conocimientos de las tecnologías Java EE era muy básico, por no decir mínimo. Partir de esa base me ha generado intensos momentos de frustración y satisfacción al mismo tiempo, ya que me ha forzado sumergirme un la búsqueda de documentación, a estudiar especificaciones, buscar soluciones y aplicar los nuevos conocimientos adquiridos.

Gracias a largas noches de estudio, prueba y esfuerzos puedo decir que ahora sé un poco más sobre los *frameworks* y tecnologías que he empleado, y que dispongo de una buena base sobre la que ampliar y construir nuevos conocimientos.

En el capítulo anterior explicaba que, a pesar de que la aplicación es completamente funcional, no doy por cumplidos todos los objetivos que me marqué inicialmente. El grave agujero de seguridad que presenta, aunque no sea obvio a ojos de los usuarios, está ahí y para mí representa un fracaso. Creo que la causa de este fracaso se debe a una mezcla de exceso de confianza y desconocimiento.

Exceso de confianza en el sentido de pensar que no iba a tener ningún tipo de problema a la hora de plasmar en una aplicación lo que tenía en mente, y

desconocimiento sobre la tecnología que manejaba, y mi conocimiento real sobre lo que sabía o no sabía hacer.

Esta situación me ha forzado a introducir cambios en la metodología inicialmente propuesta, ya que lo que había previsto que me iba a llevar varios días, después terminó convirtiéndose en varias semanas. Por lo tanto, tuve que introducir la metodología mencionada en el capítulo 1, el modelo de cinco etapas para cada componente.

Como mejoras pendientes para una futura versión de este *Battle Assistant* están el implementar el acceso mediante *JAAS, (Java Authentication and Authorization Service*), una herramienta para subir imágenes de despliegues para el administrador y un asistente que permita crear y acceder a las listas detalladas de los ejércitos que emplean los jugadores.

En resumen y para concluir, este proyecto ha supuesto una gran experiencia para mí, me ha permitido ver la realidad del desarrollo del software y ha puesto a prueba mi determinación y deseo por continuar aprendiendo y dedicarme profesionalmente al mismo.

8. Glosario

Base de datos: en este contexto nos referimos a un sistema informático que mantiene un conjunto estructurado de datos.

Diagrama de Gantt: es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado

Enterprise Java Beans (EJB): componente de negocio distribuido, desarrollado en Java, que cumple unas especificaciones concretas y por tanto se puede desplegar en cualquier contenedor de aplicaciones Java EE de un servidor de aplicaciones compatible.

Eclipse: IDE de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma.

Eclipse Marketplace: es un repositorio centralizado para todas las herramientas o *plugins* que están construidos sobre el IDE Eclipse.

Facade (Fachada): patrón de diseño estructural. Se emplea para estructurar entornos de programación y reducir la complejidad del mismo, dividiéndolo en subsistemas y minimizando las comunicaciones y dependencias entre estos.

Faces Servlet: servlet que emplea el framework JavaServer Faces

Facelets: sistema empleado por defecto por JavaServer Faces (JSF) para crear vistas. Son documentos similares a una página HTML, pero pueden contener etiquetas de diferentes tipos para generar componentes de la interfaz de usuario y llevar a cabo otras tareas.

Framework: en este contexto se refiere a una estructura de conceptos y tecnologías de asistencia, que suelen estar definidas con módulos concretos de software, y que pueden servir de base para la organización y desarrollo de software.

IDE: siglas de entorno de desarrollo integrado. Es una aplicación informática que proporciona una serie de servicios que facilitar al usuario el desarrollo de software.

JAAS (Java Authentication and Authorization Service): Interfaz de Programación de Aplicaciones que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso.

Java DataBase Connector (JDBC): es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

Java Development Kit (JDK): es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

Java EE: plataforma de desarrollo empresarial que define un estándar para el desarrollo de aplicaciones empresariales multicapa.

JavaServer Faces (JSF): framework que aplica el patrón MVC para el desarrollo de la capa web para aplicaciones Java EE.

Java Persistence Api (JPA): es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones, empleando para ello la plataforma Java, en sus ediciones estándar (Java SE) y Empresarial (Java EE).

JBoss: servidor de aplicaciones Java EE de código abierto implementado por completo en Java EE. Es multiplataforma y cumple con los estándares de servidor de aplicaciones que marca Java EE.

Managed Bean: clases Java estándar que deben tener un constructor público sin parámetros y acceso a las propiedades con métodos getter y setter.

Modelo-Vista-Controlador (MVC): es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

Plugin: aplicación o programa informático que se relaciona con otro para agregarle nuevas funcionalidades, las cuales suelen ser muy específicas.

PostgreSQL: es un sistema de gestión de bases de datos relacional orientado a objetos y libre.

Primefaces: biblioteca de componentes para JavaServer Faces (JSF) de código abierto que cuenta con un conjunto de componentes enriquecidos que facilitan la creación de las aplicaciones web.

Servlet: componente web escrito en lenguaje Java que extiende la funcionalidad de un servidor web, recibe peticiones http y genera contenido dinámico como respuesta a dichas peticiones.

Stakeholder: persona, organización o empresa que tiene interés en una empresa u organización dada. En este contexto, son las personas interesadas en la aplicación que estamos desarrollando.

Wargame: juego de guerra. Es un tipo de juego que recrea enfrentamientos armados de cualquier magnitud con un conjunto de reglas que representan la tecnología, estrategia y organización militar empleada en la época en la que está ambientado.

Warhammer 40.000: juego de guerra que emplea miniaturas, diseñado por Games Workshop y ambientado en un futuro gótico dentro de 40.000 años.

9. Bibliografía

Libros

- CAMPS Riba, Josep María. (2017) *Java EE.* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- FONT i Sagrista, Vicenç. (2017) *Caso práctico de estudio. Diseño* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- CASAS Roma, Jordi. (2017) *Diseño conceptual de bases de datos.* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- CABOT, J; GUITART, I; PRADEL, J; RAYA, J. (2017) *Análisis y diseño con patrones.* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- DEBRAUWER, Laurent. (2013) *Patrones de diseño en Java*. Barcelona: Ediciones ENI

- PRADEL, J.; RAYA, J. (2017) *Análisis UML.* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- PRADEL, J.; RAYA, J. (2017) *Requisitos.* Barcelona: UNIVERSITAT OBERTA DE CATALUNYA

- COULOURIS, G; DOLLIMORE, J; KINDBERG, T; BLAIR, G. (2012) *Distributed Systems, Concepts and Design*. Essex, England: PEARSON EDUCATION

- LEONARD, Anghel (2014) *Mastering JavaServer Faces 2.2.* Birmingham, England: PACKT PUBLISHING

- MIHALCEA, Vlad (2016) *High-Performances Java Persistence.* Cluj-Napoca, Romania: VLAD MIHALCEA

- ROMAN, E; W.AMBLER, S; JEWELL, T (2002) *Mastering Enterprise JavaBeans*. New York, Usa: WILEY COMPUTER PUBLISHING

- GAUCHAT, J.D (2017) *El gran libro de HTML5, CSS3 y JavaScript*. Barcelona: MARCOMBO

Web

- RAUH Stephan (2017) [en línea]. Madrid: Java UI: State of JavaServer Faces (JSF) in 2017 [Consulta: 1 de noviembre de 2017] <https://www.beyondjava.net/blog/java-uis-state-of-javaserver-faces-jsf-in-2016/>

- W3SCHOOLS (2017) [en línea]. Madrid: HTML File Paths [Consulta: 10 de diciembre de 2017] < https://www.w3schools.com/html/html_filepaths.asp>

- Stackoverflow (2017) [en línea]. Madrid: LazyInitializationException could not initialize proxy - no Session [Consulta: 1 de diciembre de 2017] <https://stackoverflow.com/questions/41419472/lazyinitializationexception-could-not-initialize-proxy-no-session>

- Primefaces ShowCase (2017) [en línea]. Madrid: PrimeFaces ShowCase [Consulta: 17 de noviembre de 2017] < https://www.primefaces.org/showcase/>

- Apache Tomcat documentation (2017) [en línea]. Madrid: Interface HttpServletRequest [Consulta: 30 de noviembre de 2017] <https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpServletRequest.html - getRequestURI()>

10. Anexos

10.1. Manual de instalación de la aplicación

La aplicación ha sido desarrollada y probada en un sistema operativo Windows 10, con licencia educativa de 64bits y JDK 1.8. El IDE de desarrollo es el Eclipse Oxygen 4.7, mientras que el servidor de aplicaciones web es Wildfly 10.1 (JBoss) y la base de datos que utiliza es PostgreSQL en su versión 9.6.5-1

Para poder llevar a cabo el despliegue y prueba de la aplicación es necesario tener instalado y configurado correctamente todo este software en el sistema operativo, por ello a continuación se detalla la instalación y configuración de cada una de estas herramientas.

10.2. Instalación y configuración del entorno Java

Para poder ejecutar y compilar Java necesitaremos el *Java Development Kit* o JDK. En nuestro caso usaremos la versión 1.8 para ello accedemos a la página de descargas de Oracle: http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html y procedemos a descargar la versión para Windows x64.

Java SE Development Kit 8u151

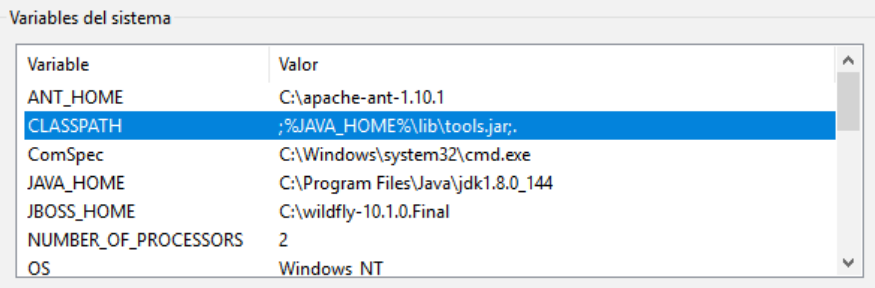
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.9 MB	jdk-8u151-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.85 MB	jdk-8u151-linux-arm64-vfp-hflt.tar.gz
Linux x86	168.95 MB	jdk-8u151-linux-i586.rpm
Linux x86	183.73 MB	jdk-8u151-linux-i586.tar.gz
Linux x64	166.1 MB	jdk-8u151-linux-x64.rpm
Linux x64	180.95 MB	jdk-8u151-linux-x64.tar.gz
macOS	247.06 MB	jdk-8u151-macosx-x64.dmg
Solaris SPARC 64-bit	140.06 MB	jdk-8u151-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.32 MB	jdk-8u151-solaris-sparcv9.tar.gz
Solaris x64	140.65 MB	jdk-8u151-solaris-x64.tar.Z
Solaris x64	97 MB	jdk-8u151-solaris-x64.tar.gz
Windows x86	198.04 MB	jdk-8u151-windows-i586.exe
Windows x64	205.95 MB	jdk-8u151-windows-x64.exe

Una vez descargada hacemos doble clic en el programa de instalación del JDK y si nos solicita reiniciar el ordenador, lo haremos.

El siguiente paso es configurar las variables de entorno adecuadamente, para ello, en las propiedades del sistema, dentro de variables de entorno, crearemos las variables JAVA_HOME y modificaremos las variables PATH y CLASSPATH de la siguiente manera:

JAVA_HOME: ponemos el directorio donde se encuentra JAVA:



PATH y CLASSPATH:

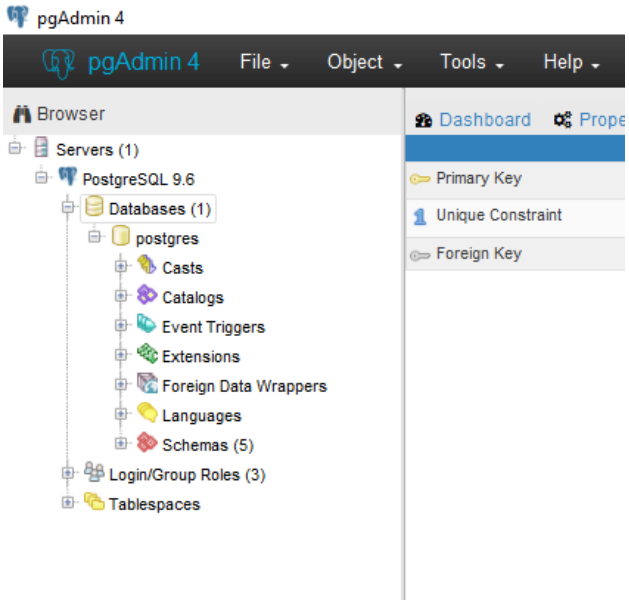
Añadimos "`;%\JAVA_HOME%\BIN`" al PATH

Añadimos al final de CLASSPATH "`;%JAVA_HOME%\lib\tools.jar;.`"

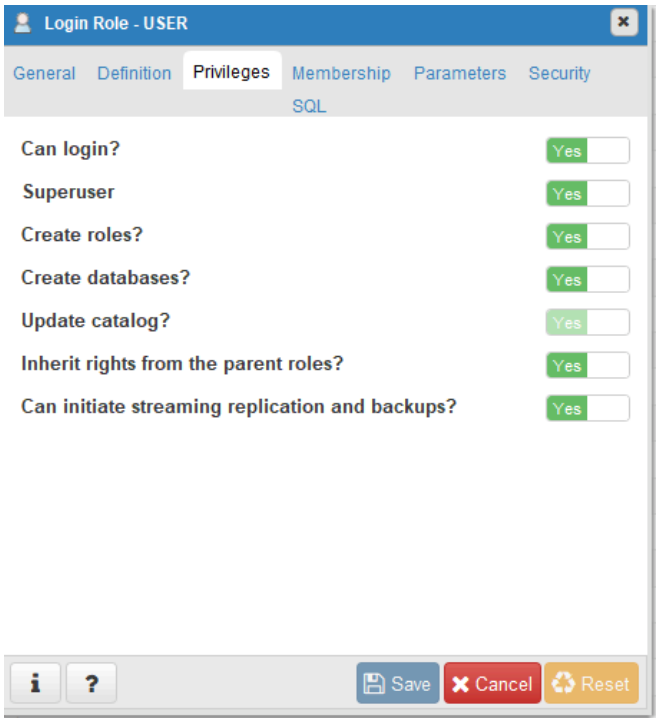
10.3. Instalación de PostgreSQL y la herramienta PGAdmin

Como hemos comentado anteriormente, la base de datos sobre la que funciona la aplicación es una base datos PostgreSQL. Para poder emplearla, instalaremos la versión 9.6.5-1 que nos podemos descargar de la página oficial dese el siguiente enlace: https://www.postgresql.org/download/

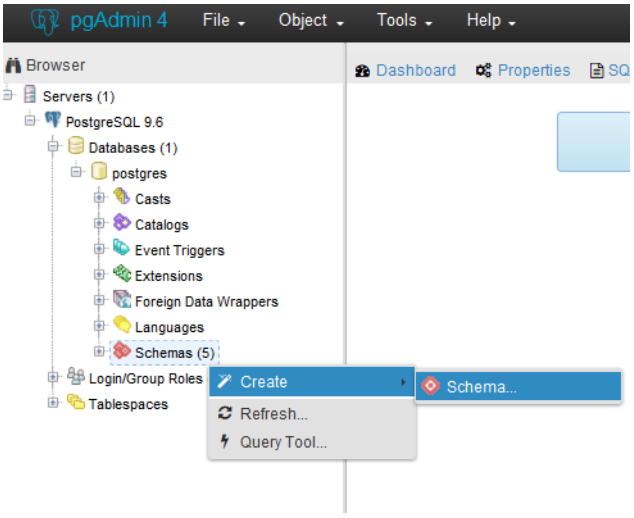
Junto con PostgreSQL también se nos instalará la herramienta PGAdmin4, que es la que emplearemos para configurar el entorno de trabajo.



Una vez instalado, accedemos a PostgreSQL mediante el PGAdmin y crearemos un nuevo rol de entrada en este caso de nombre USER y contraseña PASSWORD, y le daremos todos los privilegios.



Finalmente, crearemos un nuevo esquema (schema) en la base de datos al que llamaremos tfg:



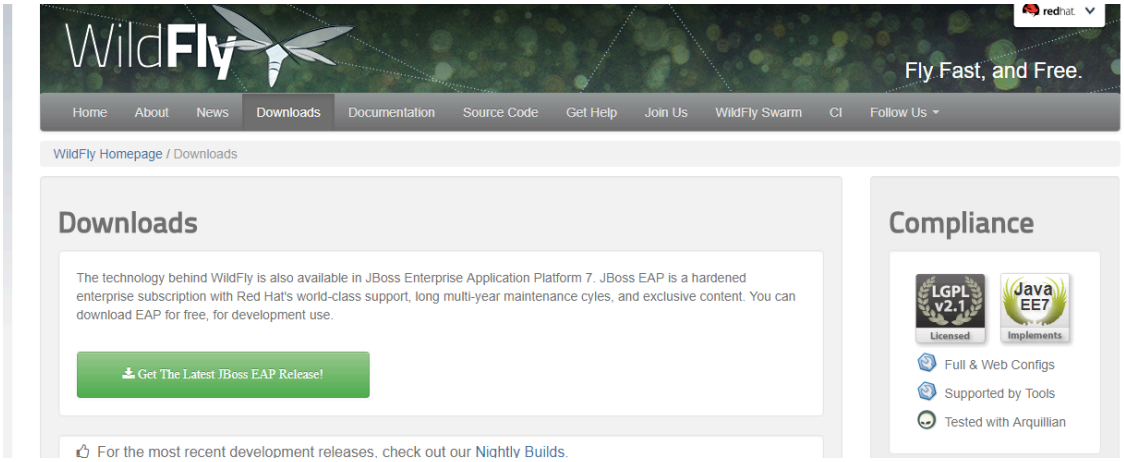
Con esto ya tenemos la base de datos preparada.

10.4. Instalación del JBoss y el conector a la base de datos PostgreSQL

A continuación vamos a instalar y configurar el servidor de aplicaciones en el que desplegaremos nuestra aplicación. La elección de JBoss (en su versión Wildfly 10.1) se debe a que es de código abierto e implementa el estándar de la arquitectura JavaEE. Como JBoss utiliza por defecto la base de datos Hypersonic, deberemos configurarlo para que emplee PostgreSQL, que es la que emplearemos nosotros.

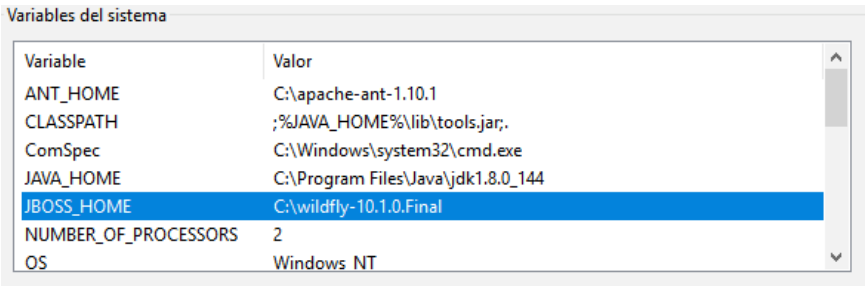
1. Wildfly

Podemos descargar e instalarlos *JBoss* desde este enlace:
http://wildfly.org/downloads/



Para instalarlo, bastará con que lo descomprimamos en la raíz de C:\ lo que hará que se nos cree una carpeta con el nombre C:/wildfly-10.1.0.Final

Lo siguiente es configurar una variable de sistema con el nombre de JBOSS_HOME y que tenga como valor la carpeta en la que está instalado JBoss:



A continuación reiniciamos el PC si nos lo solicita y pasaremos a configurar la consola de JBoss, para poder conectarnos luego.

Finalmente, para arrancar JBoss, desde la consola de Windows, iremos al directorio bin, dentro del directorio donde se encuentra instalado y ejecutaremos la instrucción:
`C:\wildfly-10.1.0.Final\bin\standalone.bat`

Sabremos que el servidor ha arrancado correctamente si accedemos desde nuestro navegador web a la dirección: http://localhost:8080

Lo siguiente es dar acceso a nuestro usuario para que pueda acceder de manera remota a los componentes que nos ofrece JBoss, para ello, desde la consola de Windows y en el directorio `C:\wildfly-10.1.0.Final\bin\` ejecutaremos la instrucción `add-user.bat`

Cuando nos pregunte el `username y password` le indicaremos USER y PASSWORD respectivamente, es decir, los mismos que tenemos configurados en PostgreSQL.

Cuando nos solicite los grupos a los que queremos que pertenezca el usuario, indicaremos `User, Trainer, Administrator`

Lo último que nos falta por configurar es el conector de PostgreSQL con JBoss, es decir el Java DataBase Conector para PostgreSQL.

2. Conector JDBC PostgreSQL

1. El conector de PostgreSQL con JBoss lo podemos descargar desde la siguiente página: https://jdbc.postgresql.org/

2. A continuación creamos la siguiente estructura: `\postgresql\main` en la carpeta en

   `C:\wildfly-10.1.0.Final\modules\system\layers\base\org` quedando pues de esta manera:

```
C:\wildfly-10.1.0.Final\modules\system\layers\base\org\postgresql\main
```

3. Aquí copiaremos el archivo .jar que nos hemos descargado desde la anterior dirección y a continuación crearemos un fichero llamado module.xml con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="org.postgresql">
   <resources>
          <resource-root path="postgresql-9.4.1209.jar"/>
   </resources>
   <dependencies>
          <module name="javax.api"/>
          <module name="javax.transaction.api"/>
   </dependencies>
</module>
```

Ahora hay que configurar *JBoss* para que reconozca esta *datasource*, por lo que tenemos que añadir al fichero de configuración de *JBoss*, `C:\wildfly-10.1.0.Final\bin\standalone\configurations\standalone.xml` el siguiente texto que hace referencia a la nueva *datasource*, colgando de la etiqueta `<datasources>`

```
<datasource jta="false" jndi-name="java:jboss/postgresDS" pool-
name="postgresDS" enabled="true" use-java-context="true" use-
ccm="false">
                    <connection-
url>jdbc:postgresql://localhost:5432/postgres</connection-url>
                    <driver-class>org.postgresql.Driver</driver-class>
                    <driver>postgresql</driver>
                    <security>
                        <user-name>USER</user-name>
                        <password>PASSWORD</password>
                    </security>
                </datasource>
```

Vemos que USER y PASSWORD son el usuario que hemos configurado en el *PostgreSQL*.

Finalmente, en el mismo fichero standalone.xml, añadimos los drivers que empleará el conector, colgando de la etiqueta `<drives>`

```
<driver name="postgresql" module="org.postgresql">
                        <xa-datasource-
class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
                    </driver>
```

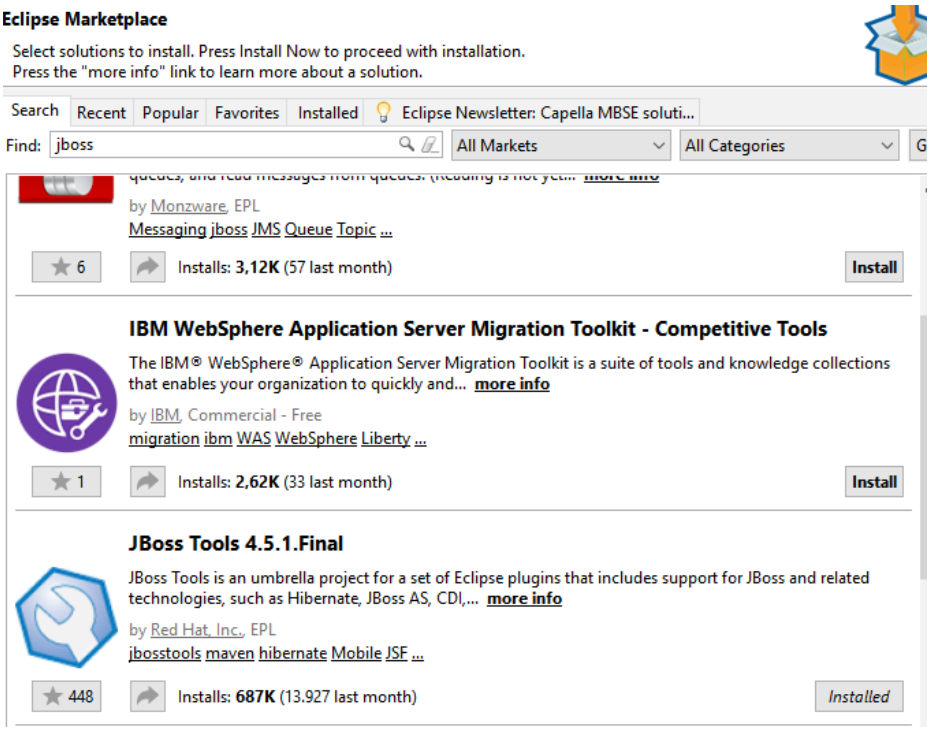
Quedándonos la etiqueta `<datasources>` del fichero standalone.xml de la siguiente manera:

```
<datasources>
                <datasource  jndi-name="java:jboss/datasources/ExampleDS"  pool-
name="ExampleDS" enabled="true" use-java-context="true">
                    <connection-url>jdbc:h2:mem:test;DB_CLOSE_DELAY=-
1;DB_CLOSE_ON_EXIT=FALSE</connection-url>
                    <driver>h2</driver>
                    <security>
                        <user-name>sa</user-name>
                        <password>sa</password>
                    </security>
                </datasource>
                <datasource  jta="false"  jndi-name="java:jboss/postgresDS"  pool-
name="postgresDS" enabled="true" use-java-context="true" use-ccm="false">
                    <connection-
url>jdbc:postgresql://localhost:5432/postgres</connection-url>
                    <driver-class>org.postgresql.Driver</driver-class>
                    <driver>postgresql</driver>
                    <security>
                        <user-name>USER</user-name>
                        <password>PASSWORD</password>
                    </security>
                </datasource>
                <drivers>
                    <driver name="h2" module="com.h2database.h2">
                        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-
datasource-class>
                    </driver>
                    <driver name="postgresql" module="org.postgresql">
                        <xa-datasource-
class>org.postgresql.xa.PGXADataSource</xa-datasource-class>
                    </driver>
                </drivers>
            </datasources>
```

10.5. Instalación y configuración de Eclipse

El último paso consiste en instalar el IDE de desarrollo Eclipse. La versión que se ha empleado para el desarrollo de la aplicación es la Oxygen 4.7 que podemos descargar desde este enlace: https://www.eclipse.org/

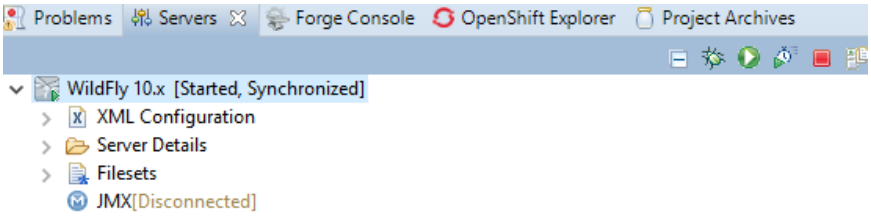
Una vez instalado, instalaremos el *plugin* para *JBoss* desde el *EclipseMarketplace*:



El *plugin* es el *JBoss Tools 4.5.1 Final*.

Una vez instalado, creamos un nuevo servidor *JBoss* en Eclipse, en concreto de la versión *Wildfly Application Server 10.x* y aceptamos todos los parámetros de la instalación por defecto.

Una vez hecho esto, podremos arrancar el servidor y dejarlo listo para el despliegue de la aplicación:

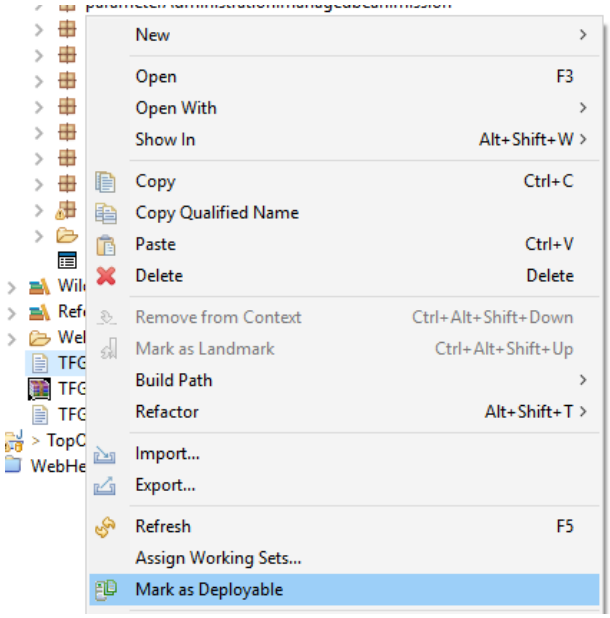


1. Despliegue de la aplicación

La aplicación viene empaquetada en un fichero ***.ear*** de nombre ***tfg.ear*** que contiene los ficheros ***tfg.war*** y ***tfg.jar*** que componen la presentación y la capa lógica de la aplicación.

Para desplegar la aplicación, podemos hacerlo desde la consola de administración de *Wildfly*: http://localhost:9990 en la que nos solicitará nuestro usuario y contraseña (USER y PASSWORD), o directamente desde el Eclipse.

Para hacerlo desde el Eclipse, importamos el proyecto al entorno y una vez importado hacemos clic derecho sobre el fichero tfg.ear y seleccionamos "Marks as Deployable".



De esta manera se producirá la sincronización y despliegue de la aplicación al servidor y ya podremos acceder a ella.

10.6. Inserción de datos y pruebas

En este último paso, emplearemos una serie de sentencias SQL para insertar unos datos iniciales en la BBDD, a fin de poder acceder y probar la aplicación. Esto se debe a que el acceso a la misma está vetado para usuarios no registrados, en concreto, aunque podamos crear usuarios normales, no podremos acceder a las funcionalidades de administración.

Por tanto emplearemos las siguientes sentencias para crear los usuarios user y admin:

```
insert into tfg.users (user_id, name, surname, email, password, alias, admin, role)
values(1, 'Juan', 'Perez', 'juan@uoc.edu', 'admin', 'admin', true, 'ADMIN');
insert into tfg.users (user_id, name, surname, email, password, alias, admin, role)
values(2, 'Pedro', 'Martinez', 'pedro@uoc.edu', 'user', 'user', false, 'USER');
```

De esta manera hemos creado los usuarios {usuario, password}, {admin, admin} y {user, user}.

Ahora vamos a crear un registro de cada tipo en la BBDD para poder probar las diferentes operaciones (Create, Retrieve, Update, Delete).

```
insert into tfg.faction (faction_id, name)
values (1, 'Marines Espaciales');
insert into tfg.faction (fation_id, name)
values (2, 'Orkos');
insert into tfg.faction (faction_id, name)
values (3, 'Astra Militarum');

insert into tfg.traits (trait_id, name, description, faction_id)
values (1, 'Lider caristmático', 'Las unidades amigas a 12UM suman +1 a su liderago', 1);
insert into tfg.traits (trait_id, name, description, faction_id)
values (2, 'Angel de la muerte', 'Repite las tiradas para impactar falladas', 1);
insert into tfg.traits (trait_id, name, description, faction_id)
values (3, 'Er Jeffe', 'Las unidades de Orkos a 12UM suman +1 a su liderago', 2);
insert into tfg.traits (trait_id, name, description, faction_id)
values (4, 'Voluntad de hierro', 'Gana la habilidad rechazar a la bruja', 3);

insert into tfg.commander (commander_id, name, trait_id, faction_id)
values (1, 'Capitán Marine Espacial', 1, 1);
insert into tfg.commander (commander_id, name, trait_id, faction_id)
values (2, 'Kaudillo Orko', 3, 2);
insert into tfg.commander (commander_id, name, trait_id, faction_id)
values (3, 'Teniente Cadiano', 4, 3);


insert into tfg.faction_objective (faction_objective_id, name, description, number,
points, faction_id)
values (1, 'Asegurar el objetivo', 'Suma 1 punto si al final de tu turno controlas un
objetivo', 1, 1, 1);
insert into tfg.faction_objective (faction_objective_id, name, description, number,
points, faction_id)
values (2, 'Loz maz duroz', 'Suma 1 punto si derrotas una unidad enemiga en CC', 1, 1,
2);

insert into tfg.faction_stratagem (faction_stratagem_id, name, description, cost,
faction_id)
values(1, 'Auspex calibrado', 'Puedes redesplegar una de tus unidades al final de la fase
de movimiento', 1, 1);
```

```
insert into tfg.faction_stratagem (faction_stratagem_id, name, description, cost, faction_id)
values(2, 'Maz dakka', 'Puedes repetir los resultados de 1’s para impactar de una de tus unidades', 1, 2);

insert into tfg.formation (formation_id, name, description, points)
values (1, 'Patrulla', 'Tu ejército debe incluir 1 CG y 2 unidades de tropas como mínimo', 1);
insert into tfg.formation (formation_id, name, description, points)
values (2, 'Batallón', 'Tu ejército debe incluir 2 CG y 4 unidades de tropas como mínimo', 3);

insert into tfg.game (game_id, enemy_vp, date, player_vp, resultado, turn, mission_id, user_id)
values (1, 3, '12/12/2017', 5, 'Victoria', 5, 1, 2);

insert into tfg.objective (objective_id, name, description, points, type)
values (1, 'Eliminar una unidad’, 'Elimina una unidad enemiga este turno', 1, 'Repetible');

insert into tfg.rule (rule_id, name, description)
values (1, 'Sabotaje', 'Al final del 5º turno, si todavía quedan vehículos enemigos en el campo de batalla, puedes hacerle 1 herida grave a uno de ellos');
```

Con estos datos, ya tenemos una base para empezar a interactuar con el sistema. Vemos que no se han creado misiones, ni estratagemas de misión, para que así se pueda ver la funcionalidad del asistente de creación de misiones.