UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de la aplicación web Warhammer 40K Tracker

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Alberto Pelufo Barquero

Tutor: Vicente Pelechano Ferragud

2020 - 2021

Introducción

El presente proyecto tiene como objetivo el desarrollo de una aplicación web que permita a sus usuarios subir trabajos de modelismo con el fin de compartirlos con la comunidad e inspirarse de aquellos trabajos publicados por otras personas. Por otra parte, la aplicación permite a sus usuarios acceder a la compra directa de miniaturas y piezas directamente a los fabricantes que puedan ayudarles en la creación de nuevas obras.

El desarrollo de la aplicación ha sido realizado siguiendo una metodología iterativa, usando herramientas como Mercurial y Sonar, que nos permitirán controlar las versiones del programa, así como mantener un control sobre la calidad del código desarrollado.

En cuanto a las tecnologías empleadas para el desarrollo, se ha escogido GWT para el frontend, ya que permite la customización de elementos mediante la herencia de widgets de librerías propias, así como Java para el desarrollo de los modelos de datos que se encargarán de trabajar con la base de datos.

Palabras Clave: GWT, Base de datos, SQL, Warhammer, Web, DAOModel.

Tabla de contenido

Índice de ilustraciones

Índice de tablas

1. Resumen

Warhammer es un juego de miniaturas de estrategia por turnos diseñado por Bryan Ansell, Richard Halliwell, Jervis Johnson y Rick Priestley a finales de la década de los 70.

En 1987, como complemento futurista para dicho juego, Rick Priestley y Andy Chambers crearon Warhammer 40.000, ambientado en un futuro distópico donde se mezclaban elementos de la ciencia ficción con elementos más típicos de las fantasías heroicas medievales.

Para dicho juego, los jugadores deben previamente ensamblar y pintar unas miniaturas de 28 mm de altura que representan soldados, vehículos o alienígenas, y, debido a la complejidad actual del juego, (ya que constantemente salen al mercado nuevas ediciones del juego, con nuevas miniaturas y reglas que aprender), hay muchos aficionados que disfrutan únicamente con el proceso de ensamblado y pintado de dichas miniaturas.

El presente proyecto tiene como objetivo el desarrollo de una herramienta para que los aficionados a este hobbie puedan organizar de manera rápida y eficaz los distintos proyectos de modelismo que tengan en mente, así como comprar aquellas piezas que necesiten para sus obras de una forma sencilla.

1.1. Motivación

Actualmente se ha presenciado cómo debido a la pandemia que ha asolado la sociedad, la población mundial se ha tenido que confinar en sus hogares, de forma que buscar maneras con las que pasar las horas sin salir de casa se volvió una prioridad para la gran mayoría de las personas.

Como consecuencia, provocó un aumento de personas aficionadas a montar y pintar Warhammer, una afición que proporciona numerosas horas de entretenimiento, a la par que permite desarrollar el lado artístico de uno mismo.

(Nota personal) Éste fue mi caso, que tras haberme aficionado brevemente a pintar Warhammer durante mi niñez, decidí darle otra oportunidad, ya que disponía de mucho más tiempo libre.

La motivación para la realización de este proyecto surge de la intención de aprovechar este nuevo auge de la afición por el modelismo, viendo una clara oportunidad para poder desarrollar una herramienta mediante la cual los usuarios puedan cubrir las siguientes necesidades.

- Búsqueda de imágenes de alta calidad de miniaturas, tanto pintadas según los esquemas de color tradicionales, como diseños personalizados creados por artistas de la comunidad.
- Recopilación de ideas a partir de otros modelos encontrados, lo cual permitirá al aficionado crear miniaturas totalmente customizadas.
- Posibilidad de compartir con la comunidad obras propias de cada usuario, aumentando así la cantidad de fotografías de miniaturas de las que se puedan beneficiar los usuarios.
- Posibilidad de conocer las piezas con las que se han compuesto cada una de las miniaturas expuestas, así como conocer todas las piezas que actualmente están en el mercado (clasificadas por categorías).

- Posibilidad de que el usuario cree sus propias listas con las piezas que le interesen / le puedan interesar en un futuro, lo cual le será de gran valor en el momento de trabajar sus diseños de modelismo.
- Posibilidad de acceder a la compra de todas las piezas que se muestran en la plataforma (la web redirigirá a través de cada pieza a los fabricantes de miniaturas que cuenten con la pieza en su stock)

1.2. Descripción del negocio

Warhammer 40k Tracker es un proyecto creado con el fin de ayudar a los aficionados del mundo Warhammer a tener un conocimiento amplio de todas las miniaturas que hay actualmente en el mercado.

Esto será de gran ayuda para ellos en el momento de decidir en qué nuevo Warhammer ponerse a trabajar. A través de la aplicación podrán hacerse una idea de qué piezas necesitan para llevar a cabo el proyecto de modelismo, así como coger inspiración de otras miniaturas de distintas ediciones.

Podrán crear sus propias listas con las piezas que les sean de interés, lo cual les permitirá trabajar mejor en la creación de sus obras de modelismo. También podrán comprar a través de la web aquellas piezas que necesiten de una forma rápida y fácil.

Estas funcionalidades serán de gran valor para el usuario puesto que actualmente el único modo de acceder a esta información es a través de la tienda oficial de Games Workshop, una web con una interfaz compleja y lenta, muy poco usable.

Para la realización de esta aplicación, se deberá desarrollar una plataforma usando Google Web Toolkit, siguiendo una implementación cliente-servidor. La plataforma contará con dos vistas, una para administrador, desde la que se podrá modificar la base de datos; y otra para usuarios, que será con registro, desde la que se podrá visualizar la base de datos, crear listas propias, y acceder a la compra de piezas.

1.3. Estructura

La presente Memoria para el TFG consta de nueve secciones principales. A continuación, se realizará un resumen de cada una de ellas, indicando qué se espera encontrar en su contenido:

Introducción: en este primer punto se aprecia el problema a resolver y las motivaciones que han llevado a la realización de este proyecto.

Estado del arte: en esta sección se documenta el estado actual del mercado al que este proyecto trata de apuntar, así como competidores y herramientas con funcionalidades parecidas.

Análisis de la idea de negocio: en este punto se presenta una imagen global del proyecto, así como de sus objetivos principales y los factores que serán claves para el éxito de este.

Metodología: en esta sección se explica y se analiza toda la metodología empleada para el desarrollo del proyecto.

Requisitos: en este punto se especifican los requisitos a cumplir por el proyecto, marcando así la dirección que se deberá tomar en el siguiente punto.

Diseño: en este punto, tomando como base los requisitos especificados en el punto anterior, se realizan los diseños de la base de datos, la arquitectura, y la interfaz.

Desarrollo: en el desarrollo se documenta toda la implementación de la herramienta, explicando la estructura que ha seguido el desarrollo y mostrando ejemplos de las distintas clases que componen el proyecto.

Aplicación final: en este punto se observa todo el comportamiento de la aplicación desarrollada, explicando su uso.

Conclusiones: para finalizar, se explican las conclusiones obtenidas del desarrollo del proyecto, así como una reflexión sobre los problemas acaecidos durante el desarrollo y cómo se han abordado.

2. Estado del arte

Con la Investigación del Estado del Arte se trata de recuperar y reflexionar sobre el conocimiento acumulado del objeto de estudio de este proyecto, aplicaciones relacionadas con el Mundo Warhammer. De este modo, será posible alcanzar un conocimiento crítico sobre la comprensión que se tiene actualmente del mismo, generar nuevas comprensiones y así poder orientar debidamente el proyecto.

Para ello, se estudiarán los diferentes proyectos desarrollados previamente en los que se hayan trabajado algunas de las funcionalidades que se ofrezcan en el proyecto plantado Warhammer 40k Tracker.

Ahora bien, puesto que Warhammer 40k Tracker busca posicionarse en el mercado como compañía destinada a satisfacer "x" necesidades de los aficionados del mundo Warhammer que actualmente no encuentran cubiertas por ningún otro proveedor, para este estudio, se analizará a su competidor espejo, Games Workshop.

2.1. Análisis del competidor espejo, Games Workshop.

Games Workshop es la plataforma Warhammer por excelencia.

A través de esta plataforma, el aficionado Warhammer puede acceder a los siguientes servicios:

- La web cuenta con un sistema de usuarios que permite a las personas acceder a sus perfiles. También te permite como usuario poder acceder a las tiendas online de Games Workshop y Forge World, así como estar al tanto de todas las novedades de producto nuevo en el mercado. Por otra parte, tener un usuario también te permite poder crear tus propias listas de productos favoritos; así como acceder con un identificador a las listas creadas por otras personas.

- La web cuenta con un localizador de centros de Hobby (centros físicos dónde comprar productos Warhammer). Introduces el código postal y sobre qué radio quieres filtrar y automáticamente se muestran las tiendas oficiales Warhammer y los centros independientes a través de los cuales también se puede comprar el producto.

- La web cuenta con un contactar con el Equipo Games Workshop y con la comunidad Warhammer.

    - Con el servicio de atención al cliente se facilita un correo electrónico y un teléfono con un horario de servicio. A través de los contactos facilitados, el usuario puede satisfacer todo tipo de incidencias relacionadas con su proceso de compra, o cualquier otra.
    - Warhammer Community (Canal a través del cual los aficionados del Warhammer pueden compartir ideas de artículos, así como miniaturas que hayan creado, para que esto sea compartido con la comunidad).

- Acontecimientos (Canal a través del cual se anuncian eventos de juego y hobby de Warhammer Community, así como convenciones y torneos en todo el mundo).
- White Dwarf (White Dwarf es una revista a través de la cual se publican cartas de los lectores; estas cartas las envían los aficionados a través del mail junto con comentarios, fotos, preguntas…).
- Redes Sociales (canales sociales enfocados en crear comunidad, impulsar la venta Warhammer y potenciar la marca Warhammer).
- Preguntas Frecuentes (apartado creado para solventar cuestiones frecuentes).
- Incidencias (canal de atención creado para solventar incidencias que puedan surgir - de distinta tipología).

- La web te permite realizar pedidos, iniciar una devolución de pedido y también reclamar un pedido.

- La web te da acceso al catálogo de juegos.

  - Warhammer 40,000 (categoría de producto).
  - Warhammer Age of Sigmar (categoría de producto).
  - Horus Heresy (categoría de producto).
  - Middle Earth (categoría de producto).

- La web te proporciona información sobre sugerencias de regalos.

  - Regalos para todo el mundo.
  - Juegos de caja.
  - Vales virtuales.
  - Lista de regalos.
  - Productos con licencia Warhammer.
  - Forge World.

(Games Workshop, 2021)

2.2. Análisis del proyecto OPTC Data Base.

Se ha considerado interesante analizar el proyecto OPTC Data Base también puesto que a nivel usabilidad se asemeja mucho al objetivo que se plantea para Warhammer 40k Tracker. Además, cuenta con una interfaz que también cumple con requisitos que serán clave en el desarrollo de la aplicación planteada para este proyecto.

Dicho esto, cabe aclarar que este proyecto no es competencia puesto que está dirigido a un mercado distinto y trata de satisfacer otro tipo de necesidad.

Por contextualizar, la aplicación trata de ayudar a los aficionados de un juego de *Smartphone* (*One Piece Treasure Cruise*), a que puedan realizar sus partidas de forma óptima pudiendo acceder a través de la plataforma a todos los personajes que existen, y allí filtrar entre todos ellos según sus necesidades.

Cuanto entras en la plataforma, puedes filtrar por los siguientes atributos:

- Filtro de afinidad.
- Filtro de tipo de personaje.
- Filtro de clase.
- Filtro de rareza.
- Filtro de coste.
- Filtro de drop (en qué parte del juego pueden darte ese personaje).
- Filtro de exclusión.
- Filtro de mapa específico.
- Filtro de temporalidad.
- Filtro de habilidad.

Lo cierto es que es una herramienta con un funcionamiento muy simple, pero a la vez muy útil para los aficionados a este juego. Funciona de la siguiente manera: se cuenta con una base de datos con un amplio número de personajes, cada uno de ellos etiquetado con distintos atributos. Así pues, cuando el usuario filtra, escoge los atributos que le son de interés, y la aplicación le muestra únicamente los que él estaba buscando. Una vez le aparece la selección de personajes filtrados, al hacer clic en cada uno de ellos, lo que encuentra es una amplia descripción de estos (nombre del personaje + foto + información del propio personaje).

(OPTC Data Base, 2021)

2.3. Análisis DAFO de los competidores + Análisis CAME.

Una vez estudiados los proyectos Games Workshop y OPTC Data Base (competidores principales del proyecto Warhammer 40k Tracker), se presenta a continuación un análisis DAFO de las mismas plataformas.

Este análisis ha sido realizado con el objetivo de detectar las debilidades que tienen actualmente los modelos de negocio que competirán con Warhammer 40k Tracker, las posibles amenazas que les podrían afectar, los puntos que las hacen fuerte y también las oportunidades que deben tenerse en consideración de cara a futuro.

Ahora bien, lo cierto es que este análisis principalmente aportaría valor a la plataforma Games Workshop y a la plataforma OPTC Data Base; sin embargo, en esta ocasión se ha realizado cómo parte de la investigación de Warhammer 40k Tracker, con el objetivo de poder así definir al 100% las funcionalidades y configuración del proyecto teniendo en cuenta el contexto.

La idea es que Warhammer 40k Tracker mantenga las funcionalidades de sus competidores principales, además de sumar otras muchas funcionalidades que corrijan las debilidades que sus competidores tienen, que pueda afrontar posibles amenazas que puedan surgir y que explote aquellas oportunidades que todavía no se han trabajado.

DAFO de los competidores principales

Tabla 1. Análisis DAFO.

DEBILIDADES ○ Plataforma poco usable. Es difícil manejarse por ella y localizar aquello que estás buscando. (Debilidad de Games Workshop) ○ Plataforma poco atractiva visualmente. (Debilidad de Games Workshop y de OPTC Data Base) ○ Plataforma no traducida a otros idiomas. (Debilidad de OPTC Data Base)	AMENAZAS Games Workshop ○ Que pueda llegar un competidor que solvente la debilidad que tiene la plataforma y que mantenga sus fortalezas. Actualmente la compañía no tiene ninguna amenaza presente puesto que no tiene competencia. (Amenaza para Games Workshop) *No se analizan las amenazas de OPTC Data Base porque no aportarían valor en este análisis.
FORTALEZAS ○ Sistema de usuarios. ○ Tienda propia. ○ Localizador de tiendas físicas. ○ Acceso a un muestrario Warhammer que sirve de inspiración para el usuario (posibilidad de crear listas, acceder a listas de otros usuarios, crear favoritos…). ○ Buen servicio de atención al cliente y resolución de problemas. ○ APP. ○ Warhammer Community + eventos (servicio creado para unir aficionados). (Fortalezas de Games Workshop) ○ Plataforma traducida a otros idiomas. (Fortaleza de Games Workshop) ○ Plataforma muy intuitiva y fácil de usar. (Fortaleza de OPTC Data Base)	OPORTUNIDADES Games Workshop ○ Mejorar el servicio de venta online. ○ Ofrecer un mejor servicio de venta a través de tiendas físicas. ○ Ofrecer cursos de modelismo para aficionados. ○ Crear una versión Premium para los más aficionados. (Oportunidades para Games Workshop) *No se analizan las amenazas de OPTC Data Base porque no aportarían valor en este análisis.

CAME (enfocado en el proyecto Warhammer 40k Tracker).

Tabla 2. Análisis CAME.

¿Cómo Warhammer 40k Tracker puede corregir las debilidades de sus competidores? ○ Creando una interfaz intuitiva y por lo tanto, fácil de usar. ○ Creando una imagen de marca atractiva (que se refleje en la plataforma). ○ Creando una plataforma capaz de ser usada por usuarios de todo el mundo (traducida a otros idiomas).	¿Cómo Warhammer 40k Tracker puede afrontar las amenazas de Games Workshop? ○ Puesto que Games Workshop no tiene competidores, Warhammer 40k Tracker, será su primer competidor.
¿Cómo Warhammer 40k Tracker puede mantener las fortalezas de Games Workshop? ○ Creando un sistema de usuarios que permita a cada uno usar la plataforma de forma más personalizada. ○ Ofreciendo la oportunidad a los usuarios de comprar online las piezas / Warhammer que deseen (en este caso, lo comprarán de cualquier tienda Warhammer que ofrezca sus productos a través de la plataforma). ○ Ofreciendo contenido de mucho valor para los aficionados Warhammer (fotos y videos de proyectos, posibilidad de intercambiar trabajos realizados con otros usuarios, capacidad de buscar aquello que les interese de forma fácil, etc.) ○ Ofreciendo un excepcional servicio de atención al cliente y resolución de problemas.	¿Cómo Warhammer 40k Tracker puede explotar las oportunidades de Games Workshop? ○ Trabajando de forma excepcional como intermediario de venta entre las tiendas vendedoras de Warhammer y los usuarios interesados en comprar. ○ En una segunda etapa, se podría plantear la opción de ofrecer un servicio adicional: modelismo para aficionados. ○ Crear una versión Premium para los más aficionados (esta versión daría al usuario más facilidades en la compra y acceso a más contenido). (Oportunidades para Games Workshop)

3. Análisis de la idea de negocio

A continuación, se analiza la Idea de Negocio Warhammer 40k Tracker. Para ello, en primer lugar, se explica la oportunidad de negocio detectada; seguido de una matriz Canvas en la que quedan reflejados todos los aspectos clave a tener presentes para comprender una idea de negocio de forma holística. Y ya, por último, se presentan los objetivos definidos a tener presentes en el momento de crear el proyecto.

3.1. Oportunidad de negocio

Actualmente la posibilidad de acceder a una web a través de la cual poder conocer todas las piezas Warhammer que están a la venta en el mercado, inspirarse con diseños creados por artistas y aficionados, y poder acceder a la compra de aquellas piezas que sean de interés para un usuario, no existe.

Sólo existe una web que ofrece algunas de las funciones del proyecto que se plantea; pero lo cierto es que se trata de una página cuya interfaz es compleja y lenta (poco usable), según se ha apreciado después de realizar una detallada auditoría de *User Experience*.

Sin embargo, sí se ha detectado que existe un mercado (aficionados del mundo Warhammer), interesado en una web que le facilite disfrutar de su afición al completo (inspirarse, conocer todo el catálogo de piezas disponibles, y por supuesto, poder acceder a su compra). Por otra parte, se ha estudiado que estos aficionados al mundo Warhammer tienden a comprar las piezas para sus obras en tiendas físicas o a través de plataformas de compra venta de productos (wallapop); y una gran mayoría coinciden en que en una gran cantidad de ocasiones no logran encontrar las piezas que desean en las tiendas de modelismo de su ciudad y les es muy complejo poder acceder a la compra del producto deseado para la creación de sus proyectos.

Dicho esto, todos estos puntos llevan a la conclusión de que es una gran oportunidad de negocio para Warhammer 40k Tracker. El proyecto agradará a los aficionados del mundo Warhammer satisfaciendo todas las necesidades que actualmente les cubre la web Games Workshop + las que no encuentran cubiertas. Y paralelamente, ayudará a los fabricantes de las piezas Warhammer a incrementar sus ventas. Los fabricantes pagarán un fee mensual que les permitirá acceder a vender sus productos a través de una plataforma digital externa dirigida al 100% a su público objetivo, así como podrán despreocuparse de los gastos de envío puesto que esta gestión la cubrirá Warhammer 40k Tracker. En definitiva, se presenta una muy buena oportunidad de negocio.

3.2. Modelo Canvas

A continuación, se presenta el modelo Canvas de Osterwalder realizado para el proyecto Warhammer 40k Tracker. En él es posible visualizar de forma clara de qué trata el modelo de negocio. Como explica Alexander, “la mejor manera de describir un modelo de negocio es dividirlo en nueve módulos que reflejen la lógica que sigue una empresa para conseguir sus ingresos”.

Estos módulos cubren perfectamente las cuatro áreas clave de un negocio (clientes, oferta, infraestructura y viabilidad económica.” (Economiatic, 2021)

¿Quiénes son nuestros socios clave?

Los profesionales escogidos para liderar el proyecto Warhammer 40k Tracker. Profesionales que liderarán la parte técnica (*back end developers* y *front end developers*), y también profesionales que trabajarán enfocados en el área empresarial (CEO, *Chief Executive Officer* y *Chief Marketing Officer*).

¿Cuáles son nuestras actividades clave?

A través de la aplicación Warhammer 40k Tracker es posible consultar las miniaturas Warhammer del mercado (todas ellas clasificadas por categorías).

A través de la plataforma, también es posible que el usuario pueda crear sus propias listas con el fin de poder trabajar mejor sus proyectos de modelismo.

Por último, el usuario tiene la posibilidad de acceder a través de la web a la compra de las piezas que le sean de interés (la web le redirigirá exactamente a la tienda de modelismo en la que tengan la pieza que desee).

¿Cuáles son los recursos clave?

Recursos tecnológicos: Página web; Recursos físicos: ordenadores y servidor; Recursos humanos: desarrolladores, marketing, comercial, finanzas y dirección ejecutiva; Recursos legales: registro de marca y protección de datos; Recursos económicos: sueldos y salarios, seguridad social, registro web, alojamiento web y financiación del proyecto

¿Cuál es la propuesta de valor?

Warhammer 40k Tracker cuenta con una interfaz usable que pone al servicio del usuario una completa base de datos en la que se muestran todas las piezas que hay actualmente en el mercado para el desarrollo de la actividad de modelismo Warhammer. La base de datos está clasificada en categorías que han sido creadas pensando en las necesidades que puede tener un aficionado del modelismo. También se muestran imágenes de alta calidad y a todo color de Warhammers creados por artistas, así como por usuarios de la plataforma; sin duda ambos recursos serán fuente de inspiración para los usuarios de Warhammer 40k Tracker cuando quieran crear una nueva obra.

La web permite al usuario, (más allá de conocer qué piezas existen en el mercado y de inspirarse), que pueda crear sus propias listas; las cuales le permitirán trabajar sus proyectos de modelismo de una forma más cómoda y por supuesto, crear obras más de su agrado. Además, la plataforma posibilita al

usuario poder acceder a la compra de las piezas que desee de forma rápida y sencilla. La aplicación (app), ofrece un servicio a los aficionados del modelismo Warhammer que les sería de una gran utilidad y que actualmente no se ofrece en el mercado.

¿Cómo serán las relaciones con los clientes?

Contacto a través de la web (formulario de contacto)**;** Contacto vía mail (mail que se comunicará en la propia página web, así como en las redes sociales) Contacto vía redes sociales (*Facebook* e *Instagram*)

¿Cuáles serán los canales de comunicación?

La plataforma web, las redes sociales (*Facebook, Instagram* y *Pinterest*) y el correo electrónico

¿Quién es el público objetivo?

Hombres principalmente y mujeres, aficionados al mundo Warhammer; con una edad comprendida entre los 23 y los 55 años; de habla hispana; con un nivel económico medio - medio alto.

¿Cuáles serán las fuentes de ingresos?

Fuente de ingresos principal: las empresas que quieran poner a la venta sus piezas de modelismo a través de la plataforma 40k Tracker deberán pagar un fee mensual. Fuente de ingresos secundaria: anuncios insertados dentro de la plataforma web.

3.3. Objetivos del modelo de negocio

Se plantean los siguientes objetivos:

Objetivo uno: Diseñar una aplicación capaz de satisfacer las necesidades de los aficionados del mundo Warhammer que actualmente demandan (las que encuentran cubiertas y las que no están disponibles en el mercado).

- Usable y visualmente atractiva.
- A través de la cuál el aficionado pueda consultar todas las miniaturas Warhammer del mercado (filtradas por categorías).
- A través de la cuál el aficionado pueda crear sus propias listas con sus Warhammer favoritos.
- A través de la cuál el aficionado pueda acceder a la compra de las piezas que le sean de interés.

Objetivo dos: Diseñar una aplicación funcional a nivel técnico.

- Desarrollada mediante tecnologías libres.
- Cuyo código trabajado sea fácilmente modificable, legible, y por supuesto, que quede debidamente documentado.

- Cuyo sistema pueda ejecutarse en diferentes entornos sin perder sus capacidades.
- Con un entorno fácilmente testeable mediante una debida estructura de dependencias.

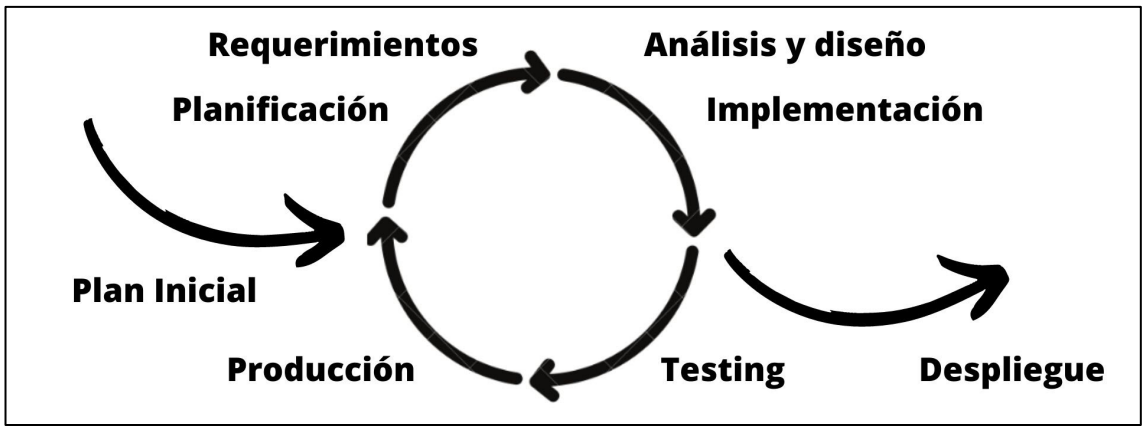
Objetivo 3: Diseñar un proyecto económicamente solvente.

- Se deberán controlar los costes (costes en recursos).
- Se deberán controlar los ingresos (ingresos proporcionados por las empresas que quieran poner a la venta sus piezas de modelismo a través de la plataforma 40k Tracker, las cuales pagarán un fee mensual + ingresos proporcionados por la publicidad que los anunciantes mostrarán en la plataforma Warhammer 40k Tracker.
- Se deberán garantizar los beneficios.

4. Metodología

En este apartado se especifica la metodología empleada para el desarrollo de la aplicación. La ilustración muestra de forma gráfica la metodología que se va a explicar en los siguientes puntos, aportando así una perspectiva general y comprensible sobre las diferentes fases de las que se compondrá la realización de la plataforma. (Wikipedia, 2021).

Ilustración 1. Modelo de desarrollo Iterativo.



4.1. Metodología iterativa e incremental

En contrapunto a la metodología de desarrollo tradicional que supone el desarrollo en cascada, la metodología iterativa e incremental surge para cubrir la necesidad de disponer de un producto usable por el usuario durante las distintas iteraciones del desarrollo. De esta manera, se consigue un desarrollo ágil donde el cumplimiento de requerimientos se realiza de forma más eficaz, dado que es una metodología que acepta mejor los cambios.

En el contexto de este proyecto, se realizará una primera iteración del desarrollo, teniendo como objetivo el desarrollo de la base de datos, así como las funcionalidades básicas que nos permitan trabajar sobre ella y visualizarla correctamente, dejando para futuras iteraciones la implementación de funcionalidades más complejas como son la creación de listas propias y la implementación en código de las piezas de las miniaturas.

4.2. Metodología iterativa e incremental - ventajas y desventajas

Ventajas de la metodología iterativa.

- Gestión de expectativas a corto plazo, ya que los objetivos a corto plazo podrán ser controlados de manera regular, con la posibilidad de tomar decisiones en cada iteración.
- Disposición de versiones usables desde las primeras iteraciones.
- Minimización de pérdidas, ya que, en el peor de los casos, solo se desaprovecha el trabajo correspondiente a una sola iteración.
- Se puede realizar una gestión natural de los cambios. Ya que, al finalizar cada iteración, se dispondrá del feedback para examinar el resultado

obtenido. Y con esta información se podrán planificar los cambios necesarios en puntos concretos de la plataforma.

- Permite mantener un control real del progreso desde las primeras iteraciones, y en base a ello realizar una estimación de si los plazos programados son asequibles.
- Posibilita la mitigación de riesgos detectados en las primeras iteraciones.
- Permite gestionar la complejidad del proyecto, ya que, en cada iteración, se trabajarán los requisitos que más interesen en ese momento.
- Como consecuencia de que cada iteración tiene que dar como resultado una versión operativa, se minimiza el riesgo de errores y se aumenta la calidad del servicio.

Desventajas de la metodología iterativa.

- Se ha de mantener en todo momento control del *feedback* para detallar la dirección del trabajo. Sobre todo al inicio y al final de cada iteración.
- Cada iteración debe finalizar con requisitos funcionales, sin dejar tareas pendientes para futuras iteraciones.
- El equipo deberá disponer de herramientas con las que realizar cambios fácilmente.
- La entrega temprana a veces produce sistemas demasiado simples.
- La finalización parcial de una plataforma puede afectar a su robustez, haciéndola más vulnerable.

4.3. Fases de la metodología

Fase 1, preparación:

Esta fase consistirá en la aceptación del proyecto. Seguidamente se le asignará una fecha de inicio y una fecha final. Se realizará la asignación inicial de recursos, tanto humanos como tecnológicos. En esta fase suele ser oportuna la definición de un calendario de trabajo; disponiendo así de un cronograma de actividades y tareas.

Fase 2, análisis y diseño:

En esta fase se analizará la situación actual, para así tener una perspectiva para discernir donde está el proyecto en el momento del análisis y hacia dónde debe avanzar. También se realizará el diseño de las funcionalidades necesarias que se convertirán en futuros requisitos para la finalización de la iteración.

Fase 3, implementación:

Durante esta fase, se realizará la implementación de las funcionalidades. El equipo llevará a cabo la división de objetivos en pequeñas tareas, de esta manera se podrá mantener

Fase 4, despliegue:

Se realizará la implementación de las funcionalidades en la plat. operativa.

Fase 5, testeo:

En esta fase se realizarán diversas pruebas para comprobar el correcto funcionamiento del sistema.

Fase 6, evaluación:

En esta fase se analizará el *feedback* obtenido para poder realizar la toma de decisiones en cuanto a la dirección que tomará la próxima iteración.

5. Requisitos

En lo referente al análisis de requisitos, en el desarrollo de software se puede interpretar como la creación de un documento que sirva de guía a la hora de definir cómo es el sistema pero que no defina cómo se desarrollará el mismo.

Según el estándar IEEE 1074 (IEEE, 1991), la fase de requisitos puede dividirse en tres fases:

- Definición de los requisitos del software.
- Definición de los requisitos de la UI respecto al resto del sistema.
- Integración de los requisitos en un documento de especificación.

Teniendo en cuenta que en este caso la información será gestionada por la misma persona, no siendo necesaria así la transferencia de esta entre equipos especializados, se hará un análisis centrado únicamente en las dos primeras fases, definiendo así los Requisitos Funcionales y los No Funcionales.
(Medium, 2018)

5.1. Requisitos funcionales

- **Registro de usuario.** Los usuarios podrán registrarse en la aplicación mediante un correo electrónico.
- **Login:** Tanto los usuarios como los administradores podrán hacer Login en la herramienta usando su correo electrónico y su contraseña.
- **Borrar usuario.** Los administradores podrán eliminar usuarios de la base de datos.
- **Logout.** Los usuarios podrán desconectarse de la sesión, así como salir del sistema.
- **CRUD miniatura.** Los usuarios de la aplicación podrán crear y leer entradas de miniaturas en la base de datos, aparte de actualizar y eliminar las miniaturas creadas por ellos mismos.
- **CRUD fabricante.** Los administradores podrán crear, leer, actualizar y eliminar fabricantes de la base de datos de miniaturas.
- **CRUD pieza.** Los administradores podrán crear, leer, actualizar y eliminar tanto accesorios como armas de la base de datos de miniaturas.

(Ionos, 2019)

5.2. Requisitos no funcionales

- **Usable.** La herramienta deberá ser intuitiva, de forma que cualquier nuevo usuario pueda aprovecharse de todas las funcionalidades que ofrece sin necesidad de invertir tiempo en familiarizarse con la misma.
- **Disponibilidad.** La herramienta deberá de estar disponible de manera ininterrumpida, ya que en cualquier momento un usuario puede necesitar acceder a la base de datos.
- **Escalabilidad.** La plataforma deberá de ser fácilmente escalable. Dado que los usuarios pueden crear entradas en la base de datos, deberán de

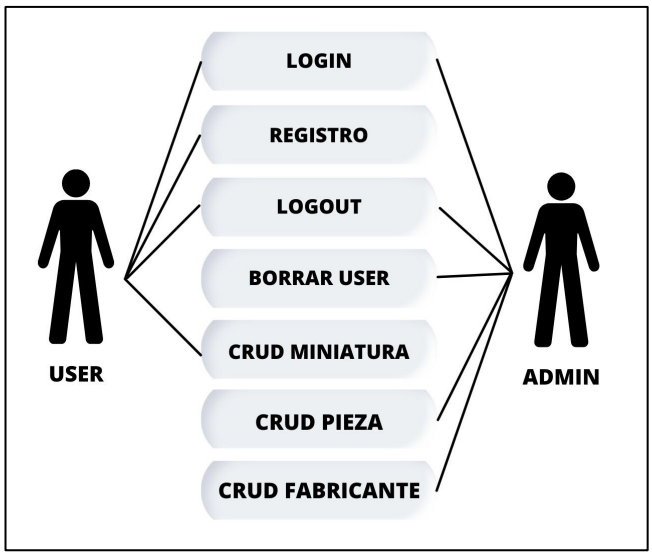
poder soportar el mayor número de operaciones sobre la base de datos de forma simultánea.

- **Compatibilidad.** La aplicación tendrá que ser compatible tanto con los distintos navegadores que pueda usar el usuario, como con dispositivos móviles.
- **Seguridad.** Los datos de la aplicación solo deberán de ser modificables por las personas autorizadas para ello. En el caso de las miniaturas, solo deberán ser modificables por el propio creador de ella y los administradores de la plataforma.
- **Rendimiento.** La base de datos deberá disponer de un pool de conexiones configurables en número para que la aplicación sea escalable en función de los recursos hardware y software disponibles.

5.3. Diagrama de casos de uso

En un diagrama de casos de uso, se presenta a continuación en lenguaje de modelado unificado, también los procesos empresariales, así como sistemas y procesos de programación orientada a objetos. Seguidamente, se presenta el diagrama de casos de uso elaborado para Warhammer 40k Tracker.

Ilustración 2. Diagrama de casos de uso.



5.4. Descripción de casos de uso

A continuación, se presentan de manera más detallada los casos de uso más relevantes para tener en cuenta en el desarrollo de la aplicación.

Tabla 3. Caso de Uso - Login.

Caso de uso	Login
Actores	Administrador, usuario
Propósito	Realizar la autenticación para acceder a la herramienta

Resumen	Se introduce el correo y la contraseña en la vista de Login y el sistema comprueba que sean correctos
Precondiciones	El correo y la contraseña pertenecen a un usuario existente
Pos condiciones	El usuario queda autenticado en la aplicación

Ilustración 3. Diagrama de flujo de Caso de Uso – Login.

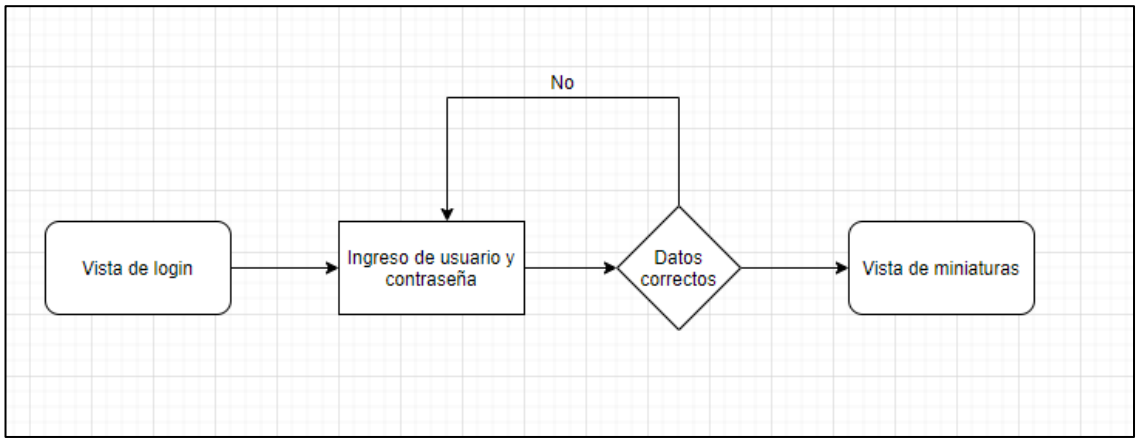


Tabla 4. Caso de Uso - Registro.

Caso de uso	Registro
Actores	Usuario
Propósito	Crear un nuevo usuario en la base de datos de usuarios
Resumen	El sistema dará de alta a un nuevo usuario
Precondiciones	-
Pos condiciones	El usuario queda registrado en la base de datos

Tabla 5. Caso de Uso - Borrar usuario.

Caso de uso	Borrar usuario
Actores	Administrador
Propósito	Eliminar un usuario existente de la base de datos
Resumen	El administrador seleccionará un usuario y el sistema lo borrará de la base de datos
Precondiciones	El administrador debe de haber hecho Login en la plataforma

Pos condiciones	-

Tabla 6. Caso de Uso - Logout.

Caso de uso	Logout
Actores	Administrador, usuario
Propósito	Cerrar la sesión previamente iniciada
Resumen	-
Precondiciones	-
Pos condiciones	-

Dentro de los casos de uso más relevantes, se muestran a continuación cuatro tablas específicas que tratan los casos de usos asociados al CRUD MINIATURA (Create, Read, Update and Delete); además de mostrarse el diagrama de flujo de “CRUD MINIATURA”.

Tabla 7. Caso de Uso - Crear miniaturas.

Caso de uso	Crear miniatura
Actores	Usuario
Propósito	Crear una entrada nueva de miniatura en la base de datos
Resumen	El usuario autenticado crea una nueva miniatura
Precondiciones	El usuario está autenticado
Pos condiciones	-

Tabla 8. Caso de Uso - Listar miniaturas.

Caso de uso	Listar miniaturas
Actores	Usuario
Propósito	Listar las miniaturas creadas por el usuario
Resumen	El usuario autenticado accede a su lista personal de miniaturas creadas por el
Precondiciones	El usuario está autenticado, existen miniaturas creadas por el usuario
Pos condiciones	-

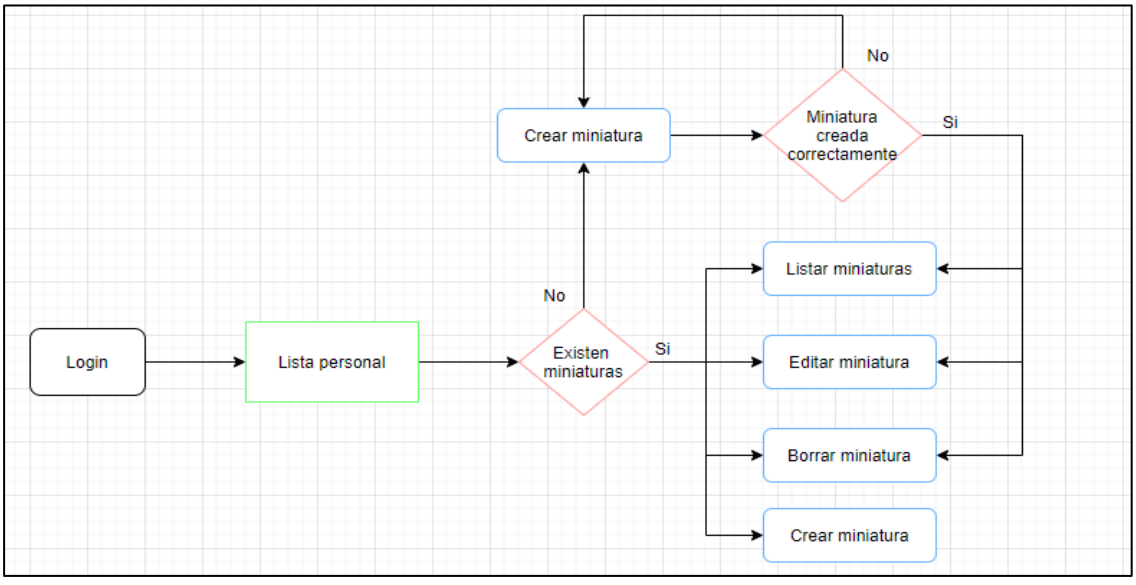
Tabla 9. Caso de Uso - Editar miniatura.

Caso de uso	Editar miniatura
Actores	Usuario
Propósito	Modificar una miniatura creada previamente por el usuario
Resumen	El usuario autenticado podrá acceder a su lista personal, en la cual seleccionará una miniatura creada por él y podrá modificarla.
Precondiciones	El usuario está autenticado, existen miniaturas creadas por el usuario
Pos condiciones	-

Tabla 10. Caso de Uso - Borrar miniatura.

Caso de uso	Borrar miniatura
Actores	Usuario
Propósito	Eliminar de la base de datos una miniatura creada previamente por el usuario
Resumen	El usuario puede seleccionar una miniatura de su lista de miniaturas personales y eliminarla.
Precondiciones	El usuario está autenticado, existen miniaturas creadas por el usuario
Pos condiciones	La miniatura queda borrada.

Ilustración 4. Diagrama de flujo de "CRUD MINIATURA.



Por otra parte, a continuación, se presentan cuatro tablas específicas que tratan los casos de usos asociados al CRUD PIEZA (Create, Read, Update and Delete); además de mostrarse el diagrama de flujo de “CRUD PIEZA”.

Tabla 11. Caso de Uso - Crear pieza.

Caso de uso	Crear pieza
Actores	Administrador
Propósito	Crea una nueva pieza
Resumen	El administrador introducirá los datos necesarios para la creación de una pieza en la base de datos
Precondiciones	El administrador está autenticado
Postcondiciones	-

Tabla 12. Caso de Uso - Listar piezas.

Caso de uso	Listar piezas
Actores	Administrador
Propósito	Lista las piezas actuales
Resumen	El administrador podrá acceder a una vista de las piezas existentes en el sistema, en la que dispondrá de filtros.
Precondiciones	El administrador está autenticado, existen piezas en la base de datos
Postcondiciones	-

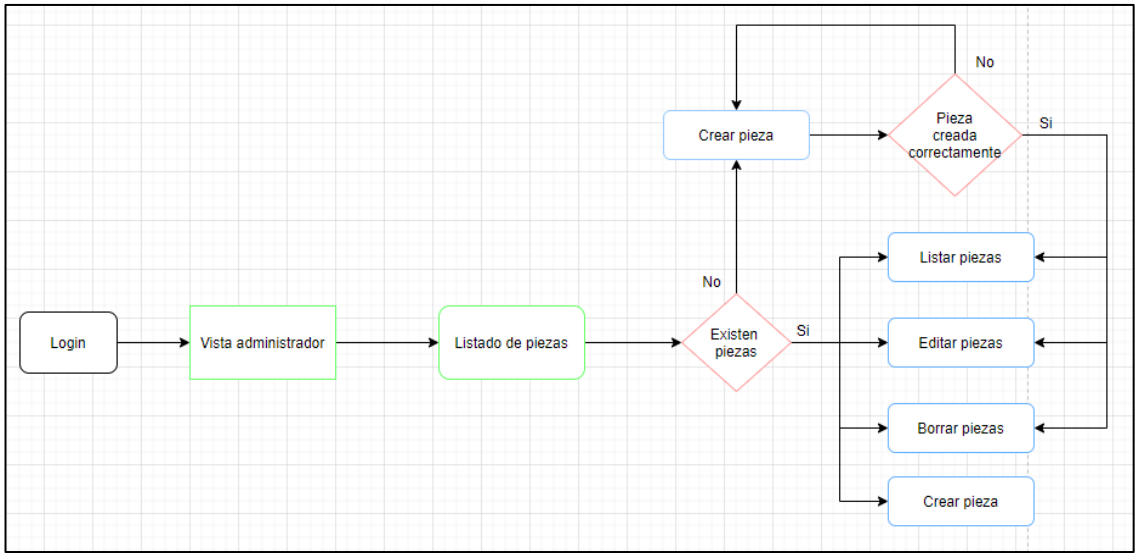
Tabla 13. Caso de Uso - Editar pieza.

Caso de uso	Editar pieza
Actores	Administrador
Propósito	Edita los datos de una pieza
Resumen	El administrador podrá cambiar datos de piezas existentes en la base de datos
Precondiciones	El administrador está autenticado, existen piezas en la base de datos
Postcondiciones	-

Tabla 14. Caso de Uso - Borrar pieza.

Caso de uso	Borrar pieza
Actores	Administrador
Propósito	Borra una pieza
Resumen	El administrador podrá eliminar piezas del sistema
Precondiciones	El administrador está autenticado, existen piezas en la base de datos
Postcondiciones	La pieza queda borrada

Ilustración 5. Diagrama de flujo de "CRUD PIEZA"



Y, para terminar, se presentan cuatro tablas específicas que tratan los casos de usos asociados al CRUD FABRICANTE (Create, Read, Update and Delete); además de mostrarse el diagrama de flujo de estos.

Tabla 15. Caso de Uso - Crear fabricante.

Caso de uso	Crear fabricante
Actores	Administrador
Propósito	Crea un nuevo fabricante
Resumen	El administrador introducirá los datos necesarios para la creación de un fabricante en la base de datos
Precondiciones	El administrador está autenticado
Pos condiciones	-

Tabla 16. Caso de Uso - Listar fabricante.

Caso de uso	Listar fabricante
Actores	Administrador
Propósito	Lista los fabricantes actuales
Resumen	El administrador podrá acceder a una vista con filtros disponibles para la visualización de los datos existentes de fabricantes
Precondiciones	El administrador está autenticado, existen fabricantes en la base de datos
Pos condiciones	-

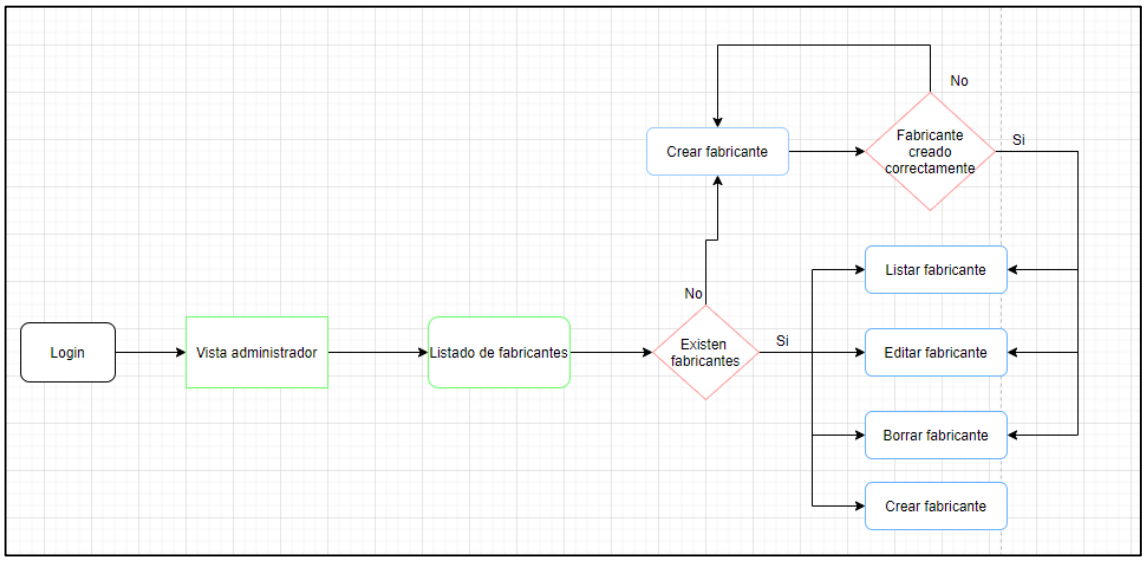
Tabla 17. Caso de Uso - Editar fabricante.

Caso de uso	Editar fabricante
Actores	Administrador
Propósito	Edita la información de un fabricante
Resumen	El administrador podrá cambiar la información existente de fabricantes en el sistema
Precondiciones	El administrador está autenticado, existen fabricantes en la base de datos
Pos condiciones	-

Tabla 18. Caso de Uso - Borrar fabricante.

Caso de uso	Borrar fabricante
Actores	Administrador
Propósito	Borra un fabricante
Resumen	El administrador podrá eliminar fabricantes del sistema
Precondiciones	El administrador está autenticado, existen fabricantes en la base de datos
Pos condiciones	El fabricante queda borrado

Ilustración 6. Diagrama de flujo de "CRUD FABRICANTE".



6. Diseño de la herramienta

6.1. Aspectos técnicos

En esta sección se analizarán en profundidad las herramientas utilizadas para la realización del proyecto, así como su aplicación en el desarrollo de este. También se analizará el patrón arquitectónico que se ha de seguir en la herramienta, especificando qué componentes formarán cada capa.

6.1.1. Herramientas utilizadas

GWT (google web toolkit): es una tecnología de Google para desarrollar aplicaciones RIA (Rich Internet Applications) con tecnología AJAX utilizando directamente un lenguaje de alto nivel como Java. Sus puntos fuertes son:

- Facilitar el desarrollo de aplicaciones AJAX, ocultando las incompatibilidades de los navegadores.
- Java: Permitir el desarrollo de aplicaciones con un lenguaje de propósito general como Java, tanto en la parte cliente como en la parte servidor.
- Framework completo: Las herramientas y APIs que proporciona Google forman un framework completo para el desarrollo de todos los aspectos de una aplicación Web.
- Al hilo del punto anterior, GWT proporciona soporte para internacionalización, test unitario e historial del navegador.
- Depuración: Posibilidad de depurar el código.
- Comunicación: Proporciona mecanismos internos de intercomunicación (RPC) para desarrollo AJAX.
- Librerías de controles visuales avanzados(widgets):
- Abstracción del DOM (código portable).

(Wikipedia, 2021)

Sonarqube: Sonar es una herramienta de software libre la cual permite evaluar la calidad del código fuente mediante el uso de herramientas de análisis de dicho tipo de código como CheckStyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código trabajado.
(Wikipedia 2021).

Mercurial: Mercurial es un sistema de control de versiones multiplataforma para el desarrollo de software. Está implementado principalmente en Python, aunque incluye una implementación binaria de diff en C. Rust. Para el acceso a repositorios mediante red, Mercurial usa un protocolo eficiente, basado en HTTP, que persigue reducir el tamaño de los datos a transferir, así como el aumento de peticiones y conexiones nuevas. Por otra parte, Mercurial puede funcionar también sobre Secure Shell, siendo el protocolo muy similar al basado en HTTP. (Perforce, 2019).

SQL: Structured Query Language (SQL) es un lenguaje de programación específico para administrar y recuperar información de sistemas de gestión de

bases de datos relacionales. SQL consiste en un lenguaje de definición de datos, un lenguaje de manipulación de datos y un lenguaje de control de datos. Algunas de las principales características de dicho lenguaje son:

- **Lenguaje de definición de datos (LDD):** El LDD de SQL proporciona comandos para definir esquemas de relación, borrar relaciones y modificar los esquemas de relación.
- **Lenguaje interactivo de manipulación de datos (LMD):** El LMD de SQL incluye lenguajes de consultas basado tanto en álgebra relacional como en cálculo relacional de tuplas.
- **Integridad:** El LDD de SQL incluye comandos para especificar las restricciones de integridad que deben cumplir los datos almacenados en la base de datos.
- **Definición de vistas:** EL LDD de SQL dispone de comandos con los que definir vistas mediante consultas.
- **Control de transacciones:** SQL dispone de comandos con los que especificar el comienzo y el final de cada transacción
- **Incorporación fácil y sencilla en lenguajes de programación:** las instrucciones de SQL se integran con facilidad en lenguajes como java, C, C++, PHP...

(Microsoft, 2017) (Wikipedia, 2021)

6.1.2. Arquitectura utilizada

A continuación, se detalla la arquitectura empleada para el desarrollo de la herramienta Warhammer 40k Tracker. En primera instancia se presenta dicha arquitectura mediante un esquema donde se puede observar el patrón diseñado, para después, capa por capa, analizar de forma más detallada cada uno de los componentes que forman la capa.

Ilustración 7. Esquema arquitectura para el desarrollo de la herramienta.

Capa de presentación: La capa de presentación estará formada por las páginas en GWT, la cuales se diseñarán desde la librería de frontend GWT Material la cual facilitará mucho el creado de tablas para visualizar bases de datos, ofreciendo múltiples herramientas para la customización de estas. De la misma manera GWTMaterial ofrece un diseño visual muy agradable para el usuario, minimizando así el tiempo a invertir en el desarrollo de la interfaz. (GWT Material, 2021).

Capa de negocio: Para la capa de negocio, se implementará un Manager Model, en el cual se implementarán las reglas de negocio del servicio. Cada una de clases de negocio o "Managers" deberán de usarse para enlazar la capa de presentación y la capa de datos, de forma que la gestión de la información sea independiente de la capa de presentación, permitiendo de esta forma tener una plataforma multicanal.

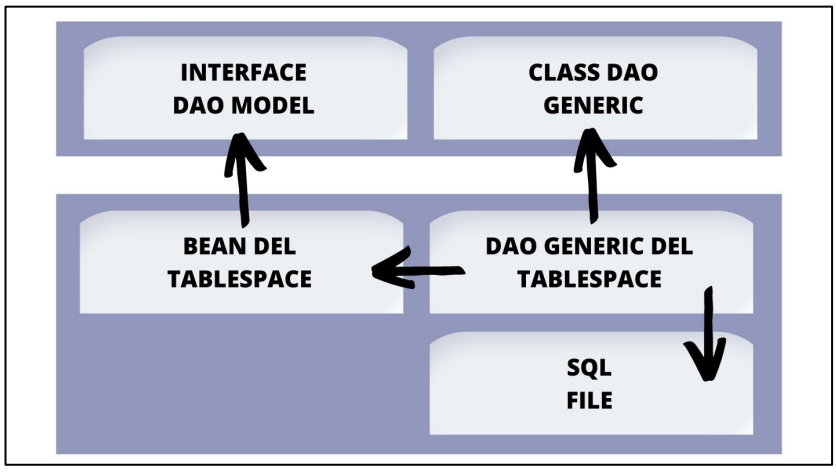
Capa de datos: La capa de datos, como su propio nombre indica, será la encargada de acceder a los datos de la aplicación. Dicha capa estará formada por las clases implementadas siguiendo el modelo DAO (Data Access Object) y por las interfaces de JDBC (Java Database Connectivity).

- **DAO Model:** La implementación del Modelo DAO deberá de cumplir las siguientes características:

    - Cada tabla de la base de datos dispondrá de su paquete "tablespace" con una clase DAOGeneric para la ejecución de consultas, un "bean" de datos para comunicar la información desde las clases DAO a las clases de negocio y un fichero XML con las consultas.
    - En los beans se deberán de poder introducir datos relacionados de otras tablas.
    - Consultas SQL parametrizables.
    - Ocultación de la implementación JDBC a las clases superiores.

(Oscar Blancarte, 2018).

Ilustración 8. DAO Model.



- **JDBC:** Java Database Connectivity (JDBC) es la especificación JavaSoft de una interfaz de programación de aplicaciones (API) estándar que permite que los programas Java accedan a sistemas de gestión de bases de datos. JDBC envía sentencias SQL a la base de datos relacional, lo cual evita reescribir la aplicación para un SGBD distinto. Está basado en X/Open SQL CLI y ODBC de Microsoft. (IBM, 2021).

6.2. Diseño

En desarrollo de software, el diseño es la etapa en la cual se crean las especificaciones necesarias para un artefacto de software, teniendo en mente los objetivos a cumplir y estando este diseño sujeto a restricciones.

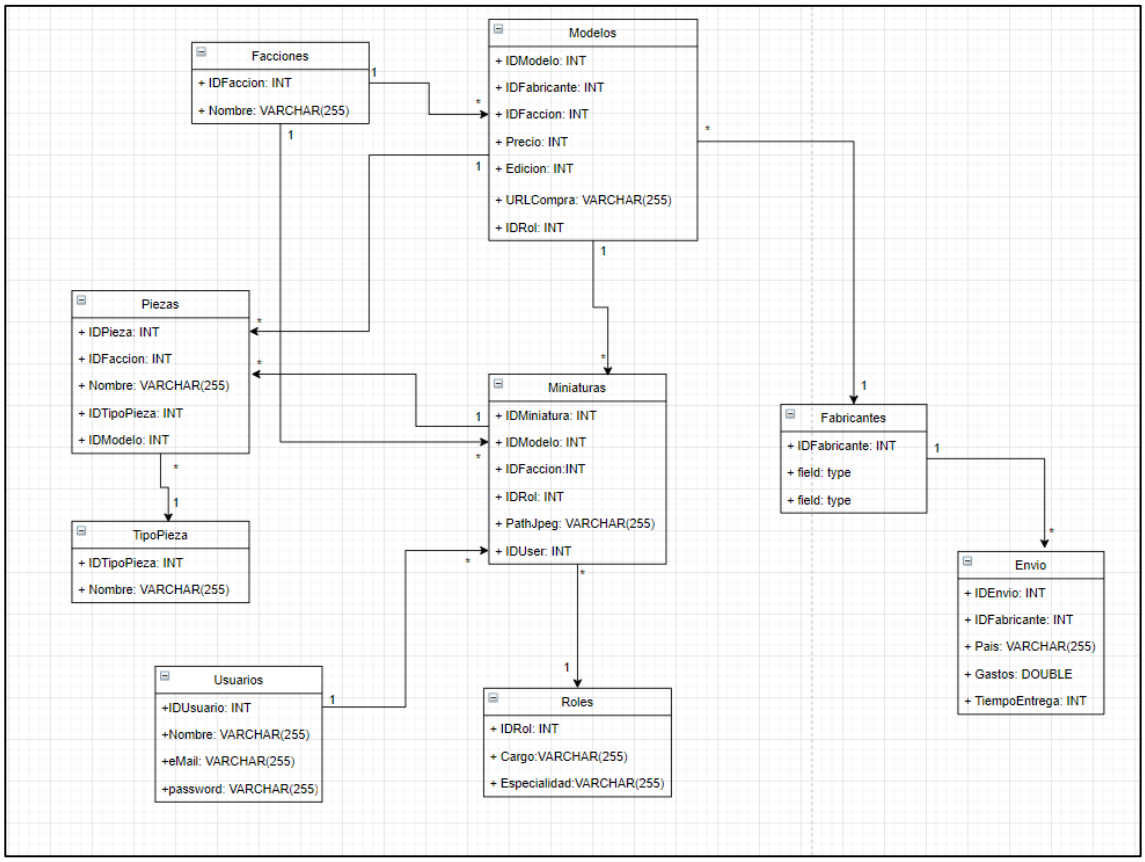
Teniendo esto en cuenta, a continuación, se enumeran algunas buenas prácticas para tener en cuenta a la hora de diseñar aspectos de un software:

- Enumerar los requisitos a cumplir por la herramienta en desarrollo, de manera que siempre habrá que tenerlos en cuenta. Perder de vista los requisitos más cruciales puede acarrear consecuencias en un futuro, ya que puede ser necesario mucho trabajo para introducir otras nuevas funcionalidades si no se han tenido en cuenta durante el diseño.
- Evitar la "visión de túnel" a la hora de enfocar problemas, ya que considerar puntos de vista alternativos para solucionar requisitos puede ser una práctica eficaz a la hora de resolver problemas estancados.
- Procurar que el diseño tenga la máxima uniformidad e integración, puesto que el diseño de un software es un proceso en el que se enumeran los requisitos a cumplir por la herramienta que se desarrollará, para después realizar dicho diseño enfocado en cubrir esas necesidades, de la forma más eficiente posible, aplicando cuando sea necesario algunos enfoques alternativos para solucionar problemas
- Diseñar con una estructura que se adapte a los cambios, basándose en conceptos como la modularidad, la jerarquía de control o la abstracción.
- Revisar de forma periódica el diseño para minimizar errores conceptuales. De esta forma se evitará que posibles fallos de interpretación perjudiquen el desarrollo.

(Wikipedia, 2021)

6.2.1. Diagrama de la base de datos

Ilustración 9. Diagrama de la base de datos.



Modelos: En esta tabla se almacenan todos los modelos creados por las compañías de miniaturas. Se compone por los siguientes campos:

- IDModelo: Identificador del modelo.
- IDFabricante: Clave ajena de fabricantes.
- IDFacción: Clave ajena de facciones.
- IDRol: Clave ajena de roles.
- Precio: Precio del modelo. Este precio se calculará de forma aproximada, ya que normalmente muchas miniaturas vienen en cajas de varias miniaturas de la misma unidad.
- Edición: Edición a la que pertenece el modelo.
- URLCompra: Url que nos dirigirá a la página web del fabricante para poder adquirir el modelo.

Miniaturas: En esta tabla se almacenan las miniaturas subidas por los usuarios, correspondiendo estas a un modelo de miniatura. No obstante, estas miniaturas pueden ser de distinta facción u ocupar un rol distinto al de la miniatura original, ya que muchos aficionados suelen hacerles modificaciones. Por ello, incluimos también en esta tabla las claves ajenas de Facción y de Rol.

- IDMiniatura: identificador de miniatura.
- IDModelo: Clave ajena de modelos.

- IDFacción: Clave ajena de facciones.
- IDRol: Clave ajena de roles.
- PathJpeg: Ruta a la imagen.

Piezas: En esta tabla se almacenan las distintas piezas que existen. Solo los administradores podrán ingresar piezas en la base de datos, ya que es conveniente asegurarse de que cada pieza no esté repetida. La tabla consta de los siguientes campos:

- IDPieza: Identificador de la pieza.
- IDFacción: Clave ajena de facciones.
- Nombre: Nombre de la pieza. Este campo deberá ser descriptivo a un nivel básico, como, “Piernas” o “Cabeza” ya que el modelo al que pertenece vendrá dado por la clave ajena IDModelo.
- IDTipoPieza: Clave ajena.
- IDModelo: Clave de ajena de modelo.

Facciones: En esta tabla se guarda la información sobre las facciones. Se compone de los siguientes campos:

- IDFaccion: Identificador de la facción.
- Nombre: Nombre de la facción.

Fabricantes: Esta tabla contiene los distintos fabricantes.

- IDFabricante: Identificador del fabricante.
- Nombre: Nombre del fabricante.
- País: País donde tiene la sede dicho fabricante. Ya que normalmente los fabricantes tienen un único punto de producción.

Envío: En esta tabla se almacena la información relacionada con envíos de los fabricantes a cada país.

- IDEnvio: Identificador de información de envío.
- IDFabricante: Clave ajena de fabricantes.
- País: País al que el fabricante realiza envíos.
- Gastos: Gastos de envío a dicho país.
- TiempoEntrega: Tiempo medio de entrega de un paquete en dicho país en días.

Roles: En esta tabla se almacenan las posibles combinaciones de cargo y especialidad, teniendo así lo que denominaremos como “rol” en el juego.

- IDRol: Identificador del rol.
- Cargo: Cargo que ocupa la unidad dentro del ejército. Por ejemplo, capitán de escuadrón.
- Especialidad: Especialidad de combate de la unidad. Por ejemplo, cuerpo a cuerpo.

Usuarios: En esta tabla se almacenan los usuarios de la plataforma.

- IDUsuario: Identificador del usuario.
- Nombre: Nombre del usuario.
- eMail: Dirección de correo electrónico del usuario.
- Password: Contraseña de acceso a la plataforma del usuario.

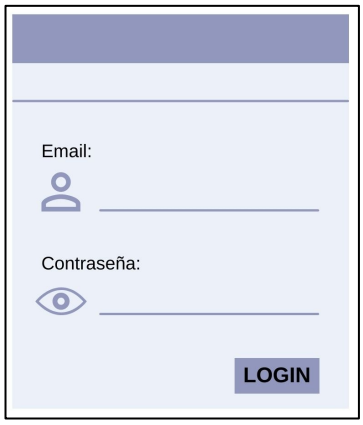
6.2.2. Diseño de la interfaz

En este apartado se ilustran los prototipos de la interfaz gráfica, diseñados con la herramienta online Canva.

Debido a que muchos elementos de la interfaz serán prácticamente iguales, cambiando pequeñas partes dependiendo del tipo de elemento con el que se esté trabajando en ese momento, se mostrarán únicamente una sola versión de cada elemento.

La vista de **Login**, como se puede observar, se compone de dos TextFields, que nos permitirán introducir el correo electrónico y la contraseña, y de un botón de Login mediante el cual se autenticarán las credenciales.

Ilustración 10. Vista del Login.



En caso de **no ser válidas las credenciales**, en el propio formulario se mostrará un mensaje de error.

Ilustración 11. Credenciales no válidas.

La **vista principal** tendrá el aspecto que se presenta a continuación. Contará con un menú superior mediante el cual el usuario podrá moverse entre las distintas secciones de la aplicación, así como se contará con un buscador y con botones para las acciones de creación, edición y borrado.

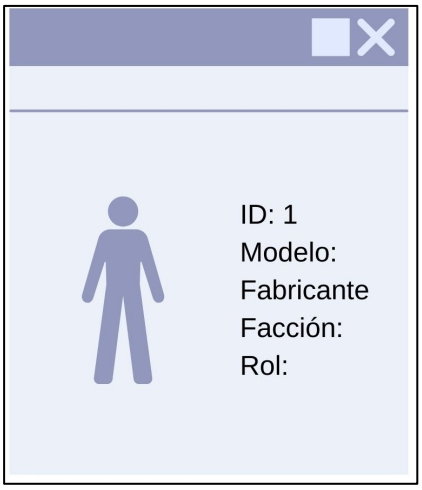
Los botones de Editar y Borrar permanecerán inactivos mientras el usuario no seleccione algún campo de la tabla.

Dado que las tablas de GWT permiten la interacción de los elementos mediante el doble clic, se prescindirá de un botón para ver en detalle una Miniatura, permitiendo así un uso más orgánico de la aplicación.

Ilustración 12. Vista principal.

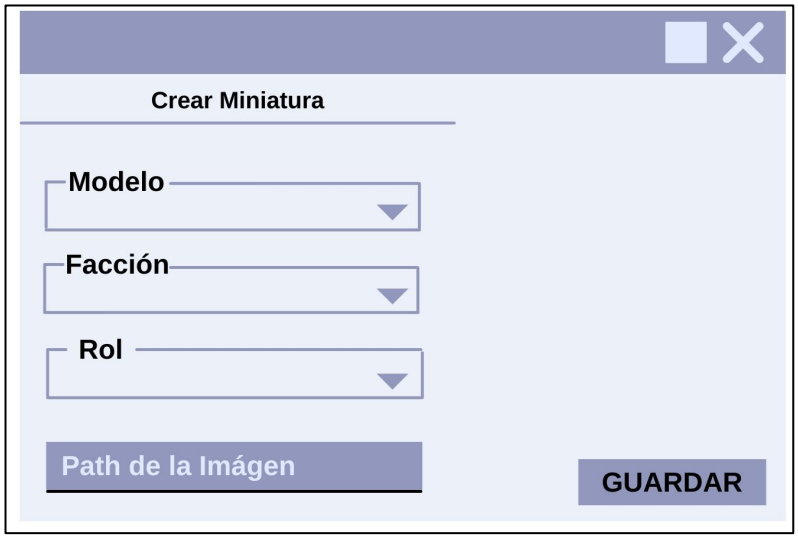
Por otra parte, se muestra en la imagen siguiente la ventana que aparecerá al hacer doble clic sobre una de las figuras de la tabla. En dicha ventana se mostrará **información más detallada** sobre dicha figura, como el fabricante del modelo que se ha usado. También es en esta ventana donde se podrá ver la imagen de la miniatura.

Ilustración 13. Información detallada sobre una figura.



La ventana que se visualizará para **crear una miniatura** será como la de la imagen próxima. Puesto que varios campos de la miniatura tienen que ser válidos (por ejemplo, el modelo tiene que ser uno existente en la base de datos) estos campos se cumplimentarán mediante desplegables que permitan realizar la búsqueda por texto - GWT posee dichos desplegables.

Ilustración 14. Crear Miniatura.



Y ya, por último, la imagen que se verá al **editar una miniatura** ya creada será como la imagen adjunta: una ventana en la que cambiar el Modelo, la Facción o el Rol que tiene asignada dicha miniatura. Mediante el botón de guardado, estos cambios serán introducidos en la base de datos.

Ilustración 15. Editar miniatura.

6.3. Conclusión del diseño.

Durante el diseño de la interfaz, se llegó a la conclusión que el listado de Piezas supondría un gran aumento de tiempo de desarrollo de la interfaz. Por ello, se decidió que, teniendo en cuenta el tiempo disponible, era procedente dejarlo para una próxima iteración, de manera que se eliminó del diseño inicial, a pesar de que se crearían las tablas en la base de datos para disponer de ellas con antelación.

De esta manera, se acortaría el trabajo dedicado a base de datos en siguientes iteraciones.

7. Desarrollo

7.1. Implementación

El uso de GWT para el desarrollo de este proyecto ha facilitado mucho el desarrollo de la aplicación web, dado que permite un desarrollo ágil de las librerías con las que acceder a la base de datos.

De la misma manera, se ha contado con la ayuda proporcionada por Indenova (empresa en la que trabaja el estudiante que presenta este trabajo) en forma de código, permitiendo el uso de ciertas librerías propias de la empresa, con el fin de agilizar el desarrollo del frontend con clases de IU que se han importado desde “com.indenova.client.widgets”.

Por otro lado, para la implementación de la librería, se ha seguido el modelo DAO para cada una de las clases principales que se tendrán en la base de datos. Así se podrá acceder de forma rápida y eficaz a la base de datos para realizar las operaciones necesarias.

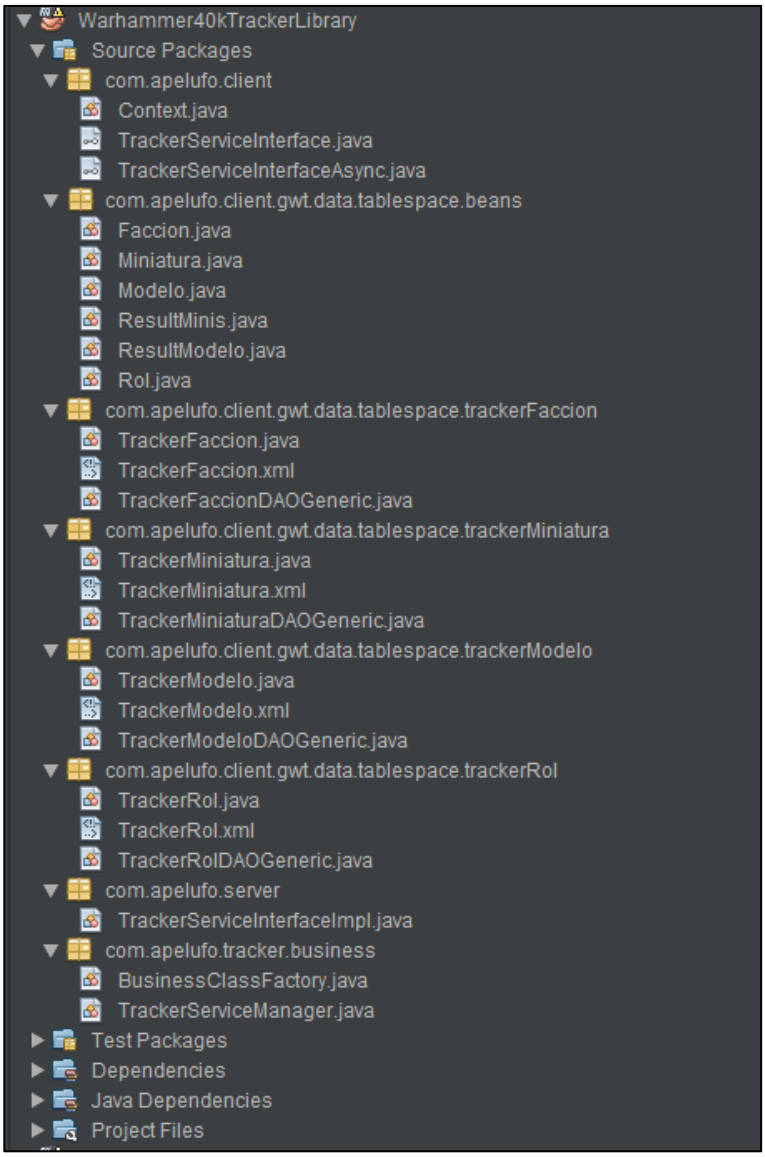
7.2. Estructura de la herramienta

Para la implementación de la plataforma, se han realizado dos proyectos complementarios.

Para el backend (lógica de negocio, modelos de datos, y clases intermedias entre cliente y servidor) se ha creado el proyecto Warhammer40kTrackerLibrary, en el cual se dispondrá de las distintas interfaces que servirán para “unir” este proyecto con su otra mitad, encargada del frontend, así como con las distintas clases de datos que se han implementado según la función que deban cumplir: en el caso de las clases alojadas en el paquete com.apelufo.client.gwt.data.tablespace.beans.

Estas clases, llamadas comúnmente “beans” de datos, serán las clases que se moverán entre el cliente y el servidor, para poder hacer más fácil la interacción con la lógica de negocio. Sin embargo, las clases que se encuentran en los paquetes com.apelufo.client.gwt.data.tablespace.*nombre de la clase*, son las clases de modelo DAO, las cuales se usarán para relacionar la plataforma con la base de datos (haciendo así más sencillo el uso de statements de SQL).

Ilustración 16. Estructura del proyecto de la librería.



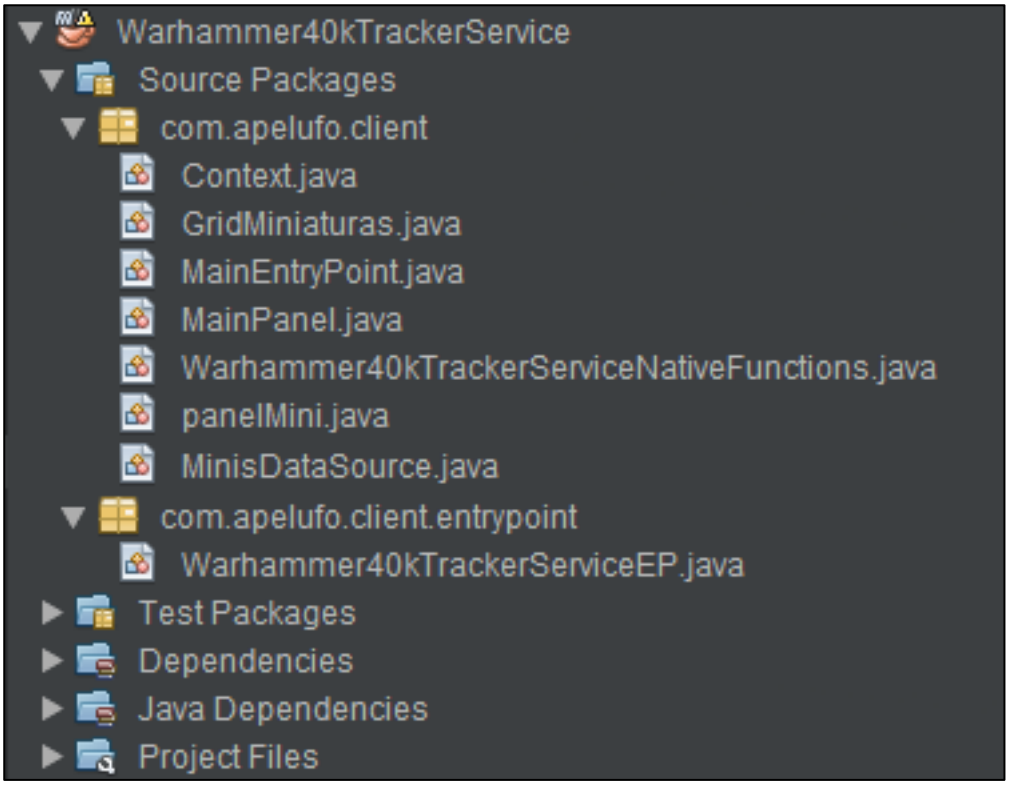
En contrapunto, para la interfaz de usuario, se ha desarrollado el proyecto Warhammer40kTrackerService, en el cual se ha desarrollado la interfaz de usuario haciendo uso de la librería GWTMaterial, la cual ofrece una multitud de elementos de interfaz a partir de los cuales podemos extender clases propias; las cuales nos permitirán adaptar dichos elementos a nuestras necesidades; este es el caso de clases como GridMiniaturas, que extiende de la clase Grid, o MainPanel, que extiende de la clase Panel.

Para la inserción de los datos y su correcta paginación en tablas paginadas, como es el caso de GridMiniaturas, se ha de crear una clase que extienda de ListDataSource <> tipando dicha extensión con el tipo de datos que vamos a mostrar en dicha tabla paginada.

También, en la clase MainEntryPoint, se especificará el punto inicial en el que cargará la aplicación, pudiendo ser una pantalla de logueo, por ejemplo.

Por razones de falta de tiempo, en este caso se ha decidido que la pantalla inicial fuera un panel en el que se ha añadido la tabla de miniaturas.

Ilustración 17. Estructura del proyecto del servicio.



7.2.1. Modelos

Para la realización de los modelos de datos, se ha seguido el patrón DAO (Data Access Object) de manera que, para cada tabla, se ha creado un conjunto de tres clases que a continuación procederemos a explicar en profundidad tomando como referencia la tabla Miniatura.

TrackerMiniaturaDAOGeneric.java:

En esta clase que extiende del modelo daoGeneric proporcionado por Indenova es donde se almacenará la lógica de la clase, así como será donde se "ejecutarán" las distintas consultas mediante la creación de statements de SQL para su posterior ejecución.

Ilustración 18. TrackerMiniaturaDAOGeneric.java

```java
public TrackerMiniatura insertRecord(TrackerMiniatura x) throws DAOSysException {
    ResultSet resultSet = null;
    PreparedStatement statement = null;
    try {

        //Se establece la conexion a la base de datos
        stablishConnection();

        //Se setea el valor de idMiniatura al ID mas bajo que no tiene entrada en la base de datos
        x.setFieldValue("idtMiniatura", getMAXRecord());

        //Se crea el array con los datos a insertar
        Object[] parameterValues = x.getParameterValues();

        //Se genera el statement de SQL
        statement = buildSQLStatement(dbConnection, XML_INS_RECORD, parameterValues);

        //Se ejecuta dicho statement
        statement.execute();

    } catch (Exception exception) {
        throw new DAOSysException(exception);
    } finally {
        closeAll(dbConnection, statement, resultSet);
    }
    return x;
}
```

TrackerMiniatura.xml:

En este fichero xml se almacenarán las distintas consultas de forma que quedarán preparadas a falta de que desde la clase DAOGeneric se le proporcione los parámetros necesarios para estar completas.

Ilustración 19. TrackerMiniatura.xml

```xml
<DAOStatements database="default">
    <SQLStatement method="INS_RECORD" parameterNb="6">
            <![CDATA[
        INSERT INTO miniatura
           (idminiatura, idmodelo, idfaccion, idrol, pathjpeg)
            values (?,?,?,?,?)
            ]]>
    </SQLStatement>
    <SQLStatement method="GET_RECORD" parameterNb="1">
            <![CDATA[
    SELECT * FROM miniatura WHERE idminiatura = ?
            ]]>
    </SQLStatement>
    <SQLStatement method="GET_ALL_RECORDS" parameterNb="0">
            <![CDATA[
        SELECT * FROM miniatura
            ]]>
    </SQLStatement>
    <SQLStatement method="UPD_RECORD" parameterNb="6">
            <![CDATA[
    UPDATE miniatura set idmodelo = ?,idfaccion = ?,idrol = ?,pathjpeg = ? WHERE idminiatura = ?
            ]]>
    </SQLStatement>
```

TrackerMiniatura.java:

En esta clase de Java se almacenarán los nombres de los campos de la base de datos en un array de Strings, así como el valor correspondiente de cada uno de dichos campos en un objeto hashMap.

También se implementarán tanto getters como setters para cada campo, así como un método “global” con el que modificar cualquier campo, pasando dicho campo como parámetro.

Ilustración 20. TrackerMiniatura.java

```
public class TrackerMiniatura  implements DAOModel, java.io.Serializable{


    protected static final String FLD_IDMINIATURA = "idMiniatura";
    protected static final String FLD_IDMODELO = "idModelo";
    protected static final String FLD_IDFACCION = "idFaccion";
    protected static final String FLD_IDROL = "idRol";
    protected static final String FLD_PATHJPEG = "pathJpeg";
    private String[] fieldNames = { FLD_IDMINIATURA, FLD_IDMODELO, FLD_IDFACCION, FLD_IDROL, FLD_PATHJPEG};

    private HashMap fields;

    //Miniatura totalmente vacia
   public TrackerMiniatura() {
        fields = new HashMap();
   }
   //Miniatura vacia, solo con ID
   public TrackerMiniatura( Integer idMiniatura) {
        fields = new HashMap();

        this.setFieldValue(this.FLD_IDMINIATURA, idMiniatura);

  }
   //Miniatura con todos los datos
   public TrackerMiniatura( Integer idMiniatura,Integer idModelo, Integer idFaccion, Integer idRol, String pathJpeg) {
        fields = new HashMap();

        this.setFieldValue(this.FLD_IDMINIATURA, idMiniatura);
        this.setFieldValue(this.FLD_IDMODELO, idModelo);
        this.setFieldValue(this.FLD_IDFACCION, idFaccion);
        this.setFieldValue(this.FLD_IDROL, idRol);
        this.setFieldValue(this.FLD_PATHJPEG, pathJpeg);
   }
   //Devuelve un array Object[] con los distintos datos
   public Object[] getParameterValues(){
        Object[] values = new Object[fieldNames.length];
      for(int i = 0; i < fieldNames.length; i++){
            values[i] = fields.get(fieldNames[i]);
      }
      return values;
   }
```

7.2.2. Manager

La clase TrackerServiceManager se encargará de recibir las llamadas de la implementación de la interfaz síncrona (explicada en el siguiente punto) y servirá de intermediario entre las dos clases de datos que tendremos por cada tabla. Por ejemplo, en la ilustración que se muestra a continuación, trabaja tanto con TrackerMiniatura.class (implementada siguiendo el modelo DAO, con la que accedemos a la base de datos) como con Miniatura.class (implementada para ser pasada entre el servidor y el cliente, de forma que, por ejemplo, en vez de tener un atributo int para IDModelo, posee un atributo de la clase Modelo)

Ilustración 21. TrackerServiceManager.java

```
public List<Miniatura> getAllMiniaturas(Integer offset, Integer numElements) {
    List<Miniatura> rtn = new ArrayList<>();
    ResultMinis res = new ResultMinis();
    //Configuramos el RequestInfo, que se encargará de devolvernos los registros necesarios
    //en funcion de el número de elementos por página y el offset
    RequestInfo ri = ThreadRequestInfoManager.getRequestInfo();
        if (ri != null) {
            if (offset > 0) {
                offset = offset / numElements;
            }

            ri.setPagBlockNumber(offset + 1);
            ri.setPagBlockSize(numElements);
        } else {
            ri = new RequestInfo();
            ri.setPagBlockNumber(offset + 1);
            ri.setPagBlockSize(numElements);
            ThreadRequestInfoManager.setRequestInfo(ri);
        }
    try {
        TrackerMiniatura[] records = factory.getTrackerMiniaturaDao(DB_SYSTEM).getAllRecords();
        for (int i = 0; i < records.length; i++) {
            Miniatura aux = new Miniatura();
            aux.setidMiniatura(records[i].getIdMiniatura());
            aux.setModelo(factory.getTrackerModeloDao().getRecord(records[i].getIdModelo()));
            aux.setFaccion(factory.getTrackerFaccionDao().getRecord(records[i].getIdFaccion()));
            aux.setRol(factory.getTrackerRolDao().getRecord(records[i].getIdRol()));
            aux.setpathJpeg(records[i].getPathJpeg());
            rtn.add(aux);
        }
        ri = ThreadRequestInfoManager.getRequestInfo();
            if (ri != null && ri.getPagTotalRows() != null) {
                //res será el objeto de la clase auxiliar con la lista y el numero total de elementos
                res.setNumTotElements(ri.getPagTotalRows());
            }
        res.getNumTotElements();
        res.setElements(rtn);
    } catch (Exception ex) {
        log.errorWithTrace("Error en el metodo getAllMiniaturas - ", ex);
        throw ex;
    }
    return rtn;
}
```

7.2.3. Interfaces

El sistema de interfaces se organizará en dos interfaces.

La primera, TrackerServiceInterface, que extenderá de RemoteService, donde declararemos los métodos que usaremos en la parte del servidor, los cuales serán implementados en la clase TrackerServiceInterfaceImpl.

Dichas implementaciones harán llamadas al Manager.

Ilustración 22. TrackerServiceInterface.java

```java
package com.apelufo.client;

import com.apelufo.client.gwt.data.tablespace.beans.Faccion;
import com.apelufo.client.gwt.data.tablespace.beans.Miniatura;
import com.apelufo.client.gwt.data.tablespace.beans.Modelo;
import com.google.gwt.user.client.rpc.RemoteService;
import java.util.List;

/**
 *
 * @author apelufo
 */
public interface TrackerServiceInterface extends RemoteService{
    public List<Miniatura> getAllMiniaturas(Integer offset, Integer numRegs);

    public Boolean saveMiniatura(Miniatura data);

    public Boolean deleteMiniatura(Miniatura data);

    public List<Modelo> getAllModelos(Integer offset, Integer numRegs);

    public Boolean savePieza(Modelo data);

    public Boolean deletePieza(Modelo data);

    public List<Faccion> getAllFacciones(Integer offset, Integer numRegs);

    public Boolean saveFaccion(Faccion data);

    public Boolean deleteFaccion(Faccion data);


}
```

Como podemos observar, se cuenta con un método por cada letra de CRUD, excepto el Update. Esto se debe a que se usará el mismo método que para guardar, con la diferencia que, en el Manager, se diferenciará a la hora de guardar teniendo en cuenta si el ID que vamos a guardar existe (Update) o no (Create). En contrapunto, se contará con la interfaz asíncrona,

TrackerServiceInterfaceAsync.

Dicha interfaz servirá de nexo entre el frontend (ya que los métodos declarados en esta interfaz serán los que se llamen desde la UI) y la lógica de negocio implementada previamente en la librería.

Esta interfaz contará con métodos void que reflejarán los métodos creados en la interfaz síncrona, con la diferencia que contarán con un parámetro Callback que hará de valor de retorno.

De esta manera, si en la interfaz síncrona tenemos un método de borrado que devuelve un valor booleano dependiendo de si el borrado se ha realizado con éxito, en la interfaz asíncrona dicho método contará con un parámetro CallBack<Boolean>.

Esto se ilustra más fácilmente en la siguiente imagen:

Ilustración 23. TrackerServiceInterfaceAsync

```
public interface TrackerServiceInterfaceAsync {

    public void getAllMiniaturas(Integer offset, Integer numRegs, AsyncCallback<List<Miniatura>> callback);

    public void saveMiniatura(Miniatura data, AsyncCallback<Boolean> callback);

    public void deleteMiniatura(Miniatura data, AsyncCallback<Boolean> callback);

    public void getAllModelos(Integer offset, Integer numRegs, AsyncCallback<List<Modelo>> callback);

    public void savePieza(Modelo data, AsyncCallback<Boolean> callback);

    public void deletePieza(Modelo data, AsyncCallback<Boolean> callback);

    public void getAllFacciones(Integer offset, Integer numRegs, AsyncCallback<List<Faccion>> callback);

    public void saveFaccion(Faccion data, AsyncCallback<Boolean> callback);

    public void deleteFaccion(Faccion data, AsyncCallback<Boolean> callback);

}
```

7.2.4. Carga de datos en el servicio

En cuanto a la implementación del servicio, únicamente se explicará de forma detallada la manera en la que dicho servicio se comunica con el backend, ya que es la única parte realmente "compleja", teniendo en cuenta que la implementación de los componentes de interfaz son poco más que extender los modelos suministrados por la librería GWTMaterial (con algunas pequeñas modificaciones en los tipos de datos que gestionan estos modelos).

En este caso se mostrará cómo se realiza la carga de datos, siendo la operación más "compleja", ya que es necesaria la implementación de un datasource que suministre los datos de forma ordenada y bien dividida para la realización de una correcta paginación.

MinisDataSource:

Esta clase dispone únicamente del método Load; se encarga de hacer la llamada al método de la interfaz asíncrona que retorna los registros necesarios (getAllMiniaturas), así como de gestionar el offset actual y el número de elementos totales recuperados de la consulta.

Ilustración 24. MinisDatasource.java

```
public class MinisDataSource extends ListDataSource<Miniatura>{
    private final Pager pager;

    // Pager pager;
    public MinisDataSource(Pager pager) {
        super();
        this.pager = pager;
    }

    /*
    Método de carga de los registros de miniaturas, recibe como parámetro un objeto LoadConfig que contiene la configuración para cargar los
    correspondientes y el objeto LoadCallback<Miniatura> que hemos creado en la clase GridMiniatura
    */
    @Override
    public void load(LoadConfig<Miniatura> loadConfig, LoadCallback<Miniatura> load) {
        //Se llama al método que se encarga de consultar los registros, pasando los datos necesarios para que el RequestInfo seleccione los r
        SERVICE_PROXY.getServiceProxy().getAllMiniaturas( loadConfig.getOffset(), loadConfig.getLimit(), new AsyncCallback<ResultMinis>() {
            @Override
            public void onFailure(Throwable caught) {
                try {
                    System.out.println("Error");
                } catch (Exception ex) {
                    GWT.log("Error en GridMiniaturas.createColumns - onSuccess del MiniDataSource", ex);
                }
            }

            @Override
            public void onSuccess(ResultMinis result) {
                pager.getActionsPanel().getActionLabel().setText(((pager.getOffset() + 1)
                        + "-" + (pager.getOffset() + pager.getLimit()))
                        + " de " + result.getNumTotElements());
                LoadResult loadResult = new LoadResult(result.getElements(), loadConfig.getOffset(), result.getNumTotElements());
                load.onSuccess(loadResult);
            }
        });
    }

}
```

GridMiniaturas:

En la clase GridMiniaturas, extensión de la clase Grid de GWTMaterial, se implementan los métodos que realizan acciones sobre la misma. En este caso, se analizará más detalladamente el método loadTableData.

Como se puede observar, es necesaria la creación de un Pager, el cual necesita como parámetro el datasource que se ha creado previamente, para su posterior adhesión al objeto GridMiniaturas.

Para la carga, se ha creado un objeto LoadCallBack<Miniatura> el cual, al realizar la carga con éxito, añadirá al datasource los datos necesarios para la correcta paginación, así como un objeto LoadConfig en el que añadiremos también los datos de paginación.

Para finalizar el método, se llamará al método load del datasource, pasándole como parámetros estos dos objetos creados previamente.

Ilustración 25. GridMiniaturas.java

```
public void loadTableData() {

    pager = new Pager(this, dataSource); //Se inicializa el pager, con la propia tabla y el Datasource como parámetros

    this.add(pager); //Se añade el pager a la tabla

    pager.setEnabled(true);
    pager.setLimitOptions(5, 10, 15, 20, 25, 30);
    pager.setOffset(0); //Seteamos el offset a 0
    pager.setLimit(10);

    pager.setVisible(true);
    pager.setTable(this);

    dataSource = new MinisDataSource(pager); //Inicializamos el datasource, pasándole el Pager
    pager.setDataSource(dataSource); //Añadimos el datasource al pager
    //Creamos un objeto LoadCallback de Miniatura,
    //y sobreescribimos el metodo OnSuccess para que pase al datasource el offset y la lista de Miniaturas
    load = new LoadCallback<Miniatura>() {
        @Override
        public void onSuccess(LoadResult<Miniatura> loadResult) {
            try {
                int offset = loadResult.getOffset();
                List<Miniatura> hm = loadResult.getData();
                dataSource.add(offset, hm);
                getView().setVisibleRange(loadResult.getOffset(), pager.getLimit());
                getView().setLoadMask(false);
                loaded(offset, hm);
            } catch (Exception ex) {
                ConsoleLog.showError("Error en onSuccess del TableDataSource", ex);
            }
        }

        @Override
        public void onFailure(Throwable caught) {
            ConsoleLog.showError("Error al cargar datos del grid: ", caught);
        }
    };
    //Creamos un objeto LoadConfig con todos los datos necesarios para que el datasource cargue los registros correspondientes
    LoadConfig config = new LoadConfig(pager.getOffset(), pager.getLimit(), null, null);

    //Hacemos la llamada al load del Datasource
    dataSource.load(config, load);

}
```

8. Aplicación Final

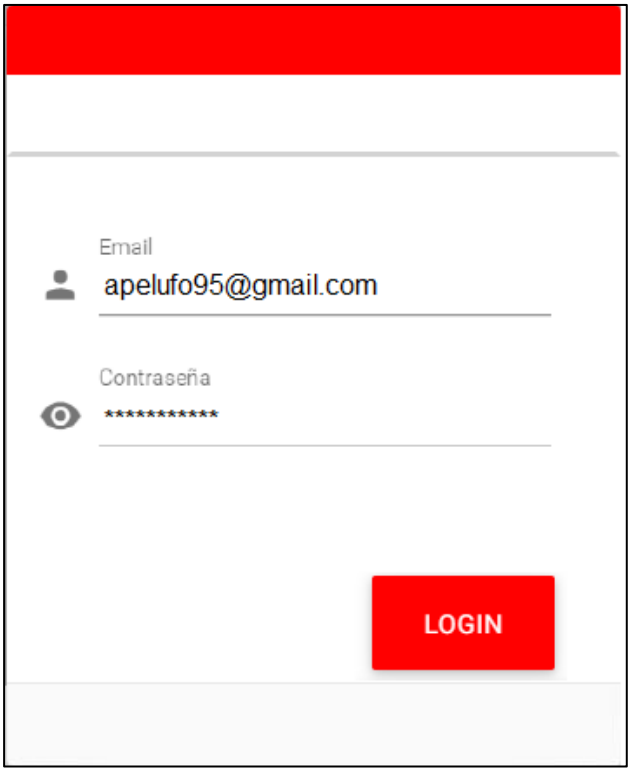
Para la muestra en este documento, se ha ejecutado la aplicación sobre un Tomcat cedido por una empresa externa, de manera que el acceso a la plataforma se realizará mediante la conexión a dicho Tomcat mediante WinSCP y la configuración de túneles del protocolo SSH.

8.1. Login

El primer paso para acceder a la herramienta consistirá en acceder a la misma mediante la autenticación del usuario.

El usuario debe de introducir su correo electrónico y contraseña correctamente, para seguidamente hacer clic en el botón LOGIN.

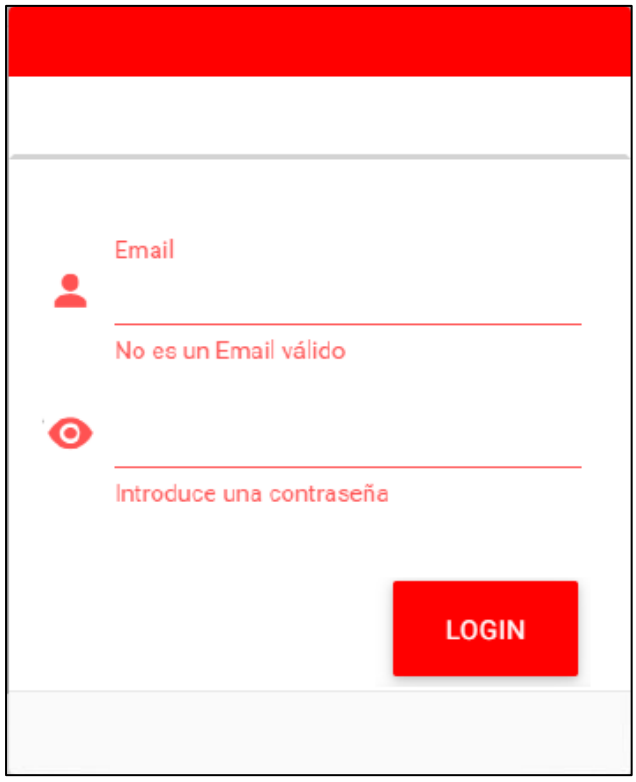
Ilustración 26. Login de la aplicación final



Si la dupla email-contraseña es correcta, el sistema redirigirá al usuario a la vista principal, siendo esta la vista de Miniaturas.

En caso contrario, el formulario tornará sus campos de color rojo y avisará al usuario de que los datos introducidos no son correctos.

Ilustración 27. Fallo de login en la aplicación final

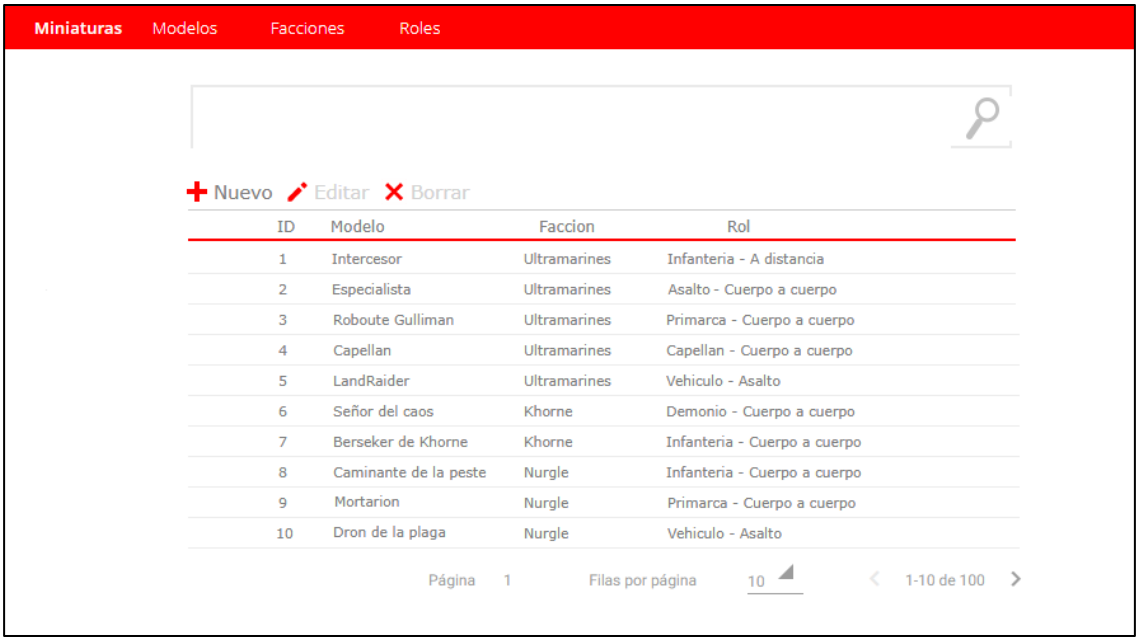


8.2. Vistas

Al introducir correctamente las credenciales, el sistema redirigirá a la vista de miniaturas.

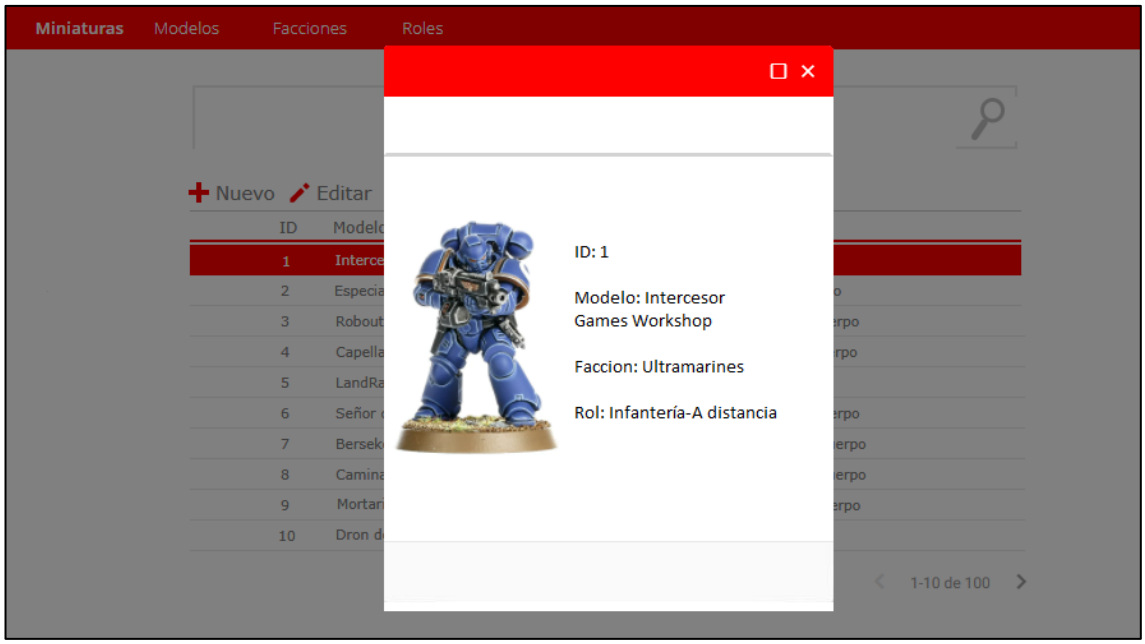
Desde esta vista, el usuario podrá moverse entre las otras vistas de la aplicación, así como visualizar las entradas de la tabla Miniaturas. Esta vista también dispone de un buscador, mediante el cual se puede filtrar usando cualquiera de los campos de la miniatura, ya que este buscador mostrará las miniaturas que contengan el texto introducido en alguno de los campos.

Ilustración 28. Vista principal de miniaturas



También se puede ver con detalle cualquiera de las miniaturas haciendo doble clic en ella. De esta manera se abrirá una ventana donde poder ver de que fabricante es el modelo usado para la miniatura, por ejemplo.

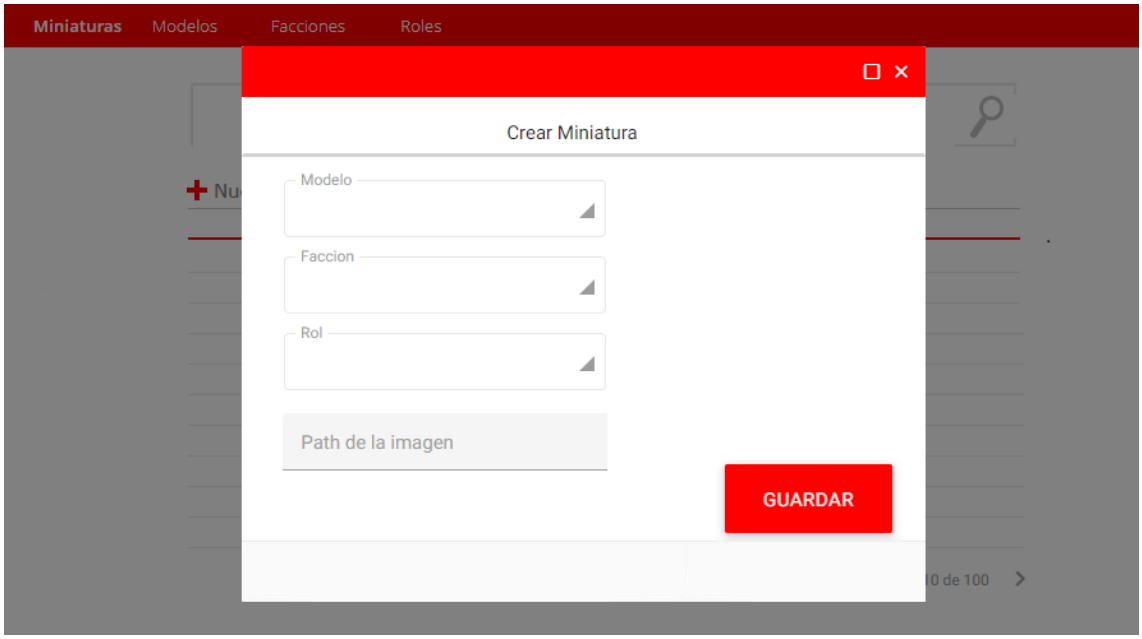
Ilustración 29. Vista en detalle de una miniatura



8.3. Creado, edición y borrado

Desde la vista de Miniaturas, al hacer clic en Crear el sistema mostrará el formulario de creación de Miniaturas. En dicho formulario, se encuentran tres desplegables en los que poder escribir para realizar una búsqueda más eficaz de los datos de la miniatura, así como una caja de texto que permite introducir el path de la imagen alojada en el servidor. Con el botón de guardar, se guardará la miniatura.

Ilustración 30. Formulario de creación de una miniatura

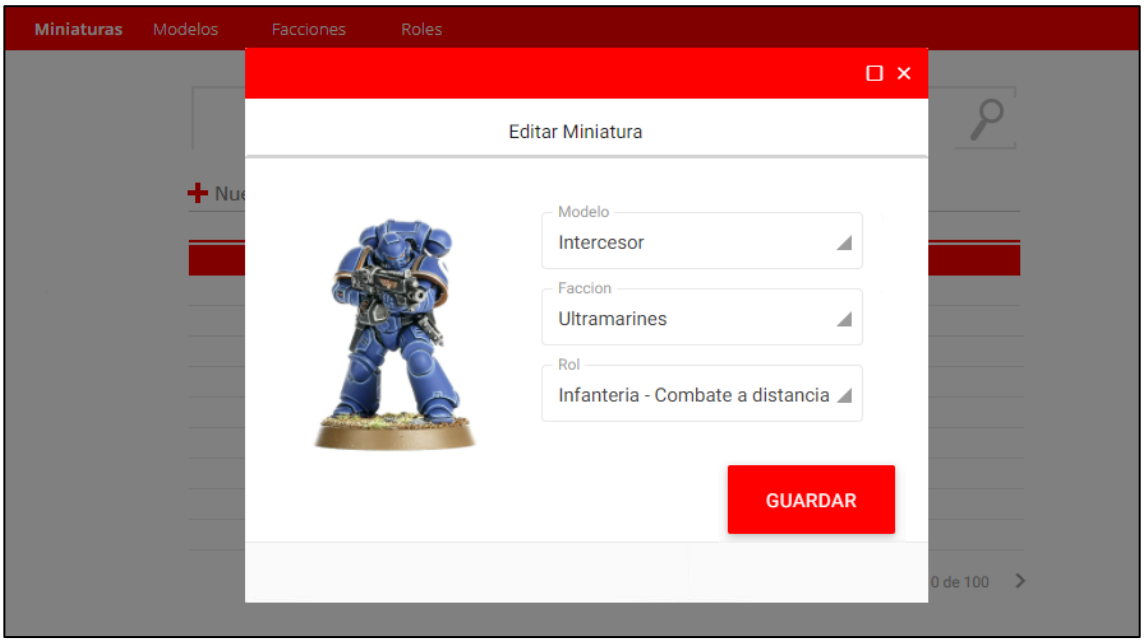


De igual manera, haciendo un clic sobre cualquier entrada, se habilitarán las opciones de edición y borrado de la miniatura.

Al seleccionar editar, se mostrará el siguiente formulario donde modificar tanto el modelo, como la Facción o el Rol, haciendo clic en los campos de texto (los cuales filtran entre ellos mismos a medida que se seleccionen). Permitiendo así una selección más fácil e intuitiva.

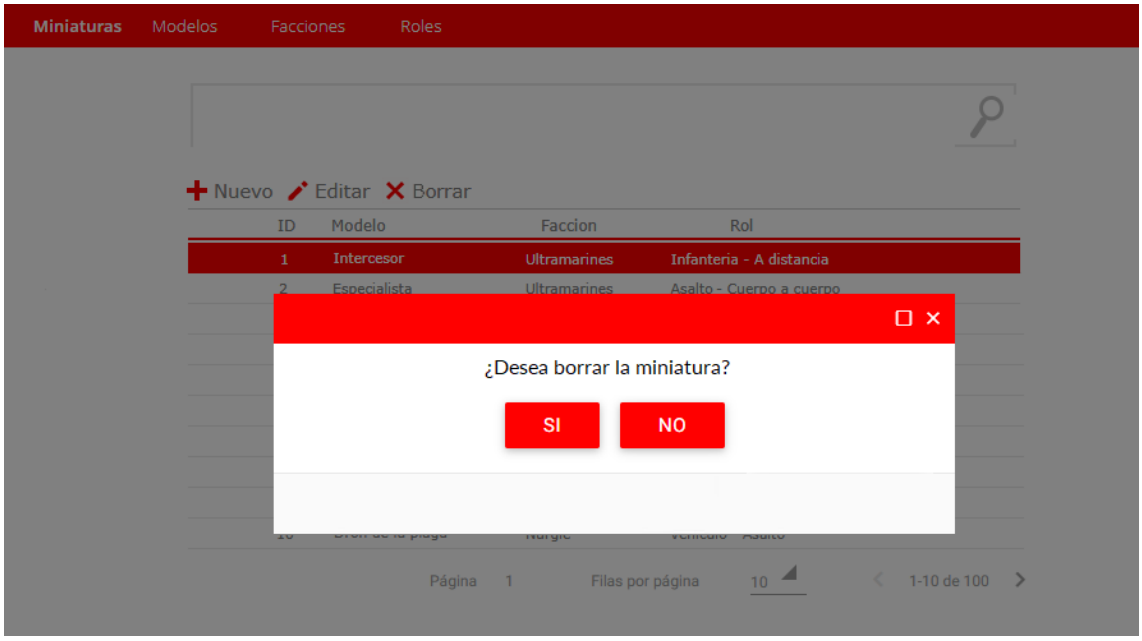
Para finalizar la edición, se hará clic en el botón Guardar.

Ilustración 31. ***Formulario de edición de una miniatura***



Al seleccionar borrar, el sistema mostrará una ventana básica para la confirmación, con el fin de evitar borrados indeseados en la base de datos.

Ilustración 32. Ventana de confirmación de borrado



9. Conclusiones

(Nota personal). Con la realización de este proyecto, he podido percibir cuán importante es la planificación a la hora de realizar cualquier desarrollo con un mínimo de complejidad.

Es de gran importancia ser consciente de la dirección que el proyecto toma, ya que tener en mente los posibles cambios y problemas que pueden surgir durante la realización de este nos puede ayudar a que la solución de estos sea mucho más fácil de implementar.

También me he podido percatar de cómo de importante es la elección de las tecnologías, ya que, por ejemplo, GWT ofrece muchas facilidades a la hora de implementar una interfaz que se adapte a las necesidades del producto, pero, en cambio, para realizar una interfaz visualmente agradable puede ser hasta cierto punto tedioso, ya que sólo puede editarse mediante el código, teniendo que realizar subidas de código al Tomcat constantes para cada modificación, o, como alternativa, modificar el código fuente de la página al ser ejecutada, para luego traspasar dichas modificaciones al código local.

De la misma manera, la realización de este proyecto me ha ayudado a comprender como en etapas tempranas del desarrollo, si se enfoca de la debida manera, se puede empezar a trazar la línea que defina la estructura de una herramienta de manera que dicha herramienta sea mucho más adaptable a otras nuevas funcionalidades que se tenga en vista para su implementación en un futuro.

10. Bibliografía

Economiatic, 2021. El lienzo del modelo de negocio. Recuperado el día 12 de julio de https://economiatic.com/business-model-canvas/

Games Workshop, 2021. Games Workshop. Recuperado el día 11 de julio de https://www.games-workshop.com/Home?_requestid=2531313

GWT Material Design, 2021. GWT Material. Recuperado el día 17 de agosto de https://gwtmaterialdesign.github.io/gwt-material-demo/#about

IBM, 2021. Informix Servers. Recuperado el día 20 de agosto de https://www.ibm.com/docs/es/informix-servers/12.10?topic=started-what-is-jdbc

Ionos, 2019. CRUD, la base de la gestión de datos. Recuperado el día 24 de julio de https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/

Medium, 2018. Requerimientos funcionales y no funcionales, ejemplos y tipos. Recuperado el día 22 de julio de https://medium.com/@requeridosblog/requerimientos-funcionales-y-no-funcionales-ejemplos-y-tips-aa31cb59b22a

Microsoft, 2017. Bases de datos. Recuperado el día 20 de agosto de https://docs.microsoft.com/es-es/sql/relational-databases/databases/databases?view=sql-server-ver15

OPTC Data Base, 2021. OPTC Data Base. Recuperado el día 12 de julio de https://optc-db.github.io/

Oscar Blancarte, 2018. Data Access Object DAO Pattern. Recuperado el día 22 de agosto de https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/

Perforce, 2019. Mercurial vs GIT. Recuperado el día 10 de agosto de https://www.perforce.com/blog/vcs/mercurial-vs-git-how-are-they-different

Wikipedia, 2021. Desarrollo Iterativo y creciente. Recuperado el día 19 de julio de https://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente#Debilidades_de_este_modelo_de_desarrollo

Wikipedia, 2021. Google Web Toolkit. Recuperado el día 12 de agosto de https://es.wikipedia.org/wiki/Google_Web_Toolkit

Wikipedia, 2021. SonarQube. Recuperado el día 16 de agosto de https://es.wikipedia.org/wiki/SonarQube

Wikipedia, 2021. SQL. Recuperado el día 16 de julio de https://es.wikipedia.org/wiki/SQL#Caracter%C3%ADsticas_generales_de_SQL

Wikipedia, 2021. Diseño Software. Recuperado el día 25 de agosto de https://es.wikipedia.org/wiki/Dise%C3%B1o_de_software

Wikipedia, 2021. Diseño Software. Recuperado el día 26 de agosto de https://es.wikipedia.org/wiki/Dise%C3%B1o_de_software