TecnoCampus
Escola Superior
Politècnica

Centre adscrit a la
upf. Universitat Pompeu Fabra Barcelona

Grau en Enginyeria Informàtica de Gestió i Sistemes d'Informació

Creador de llistes de Warhammer 40K

Memòria Intermèdia

Francesc-Xavier Boix Aburraad

TUTOR: Alfredo Rueda Unsain

5é Doble Grau Info-Videojocs

TecnoCampus
Mataró-Maresme

DEDICATIORIA

A l'amor de la meva vida, Marina Argimon, per romandre sempre al meu costat i tenir el seu suport incondicional.

Al meu estimat avi, Benet Boix, el meu referent a la vida.

A la meva estimada germana, Anna Boix, la meva família ara i sempre.

Als meus estimats pares, Benet i Mirza, espero que us sentiu orgullosos.

I en honor de la meva gosseta Negrita, sempre et portaré al meu cor.

Abstract

The project consists of developing a mobile application for creating Warhammer 40K rosters. For the front-end, React Native has been used to implement the application, with a strong emphasis on providing an intuitive and user-friendly interface. For the back-end, Java Spring Boot and JPA have been employed, following a well-organized hexagonal architecture that enables easy management and scalability of the system. This application structure provides a solid foundation for future maintenance and the possibility of adding new features.

Resum

El projecte consisteix en el desenvolupament d'una aplicació mòbil per a la creació de llistes de Warhammer 40K. Per al front-end, s'ha utilitzat React Native per a implementar l'aplicació, posant un gran èmfasi en proporcionar una interfície d'usuari intuïtiva i fàcil d'utilitzar. Per al backend, s'ha utilitzat Java Spring Boot i JPA per a la seva implementació, adoptant una arquitectura hexagonal ben organitzada, que permet una gestió senzilla i escalabilitat del sistema. Aquesta estructura de l'aplicació proporciona una base sòlida per al manteniment futur i la possibilitat d'afegir noves funcionalitats.

Resumen

El proyecto consiste en el desarrollo de una aplicación móvil para la creación de listas de Warhammer 40K. Para el front-end, se ha utilizado React Native para implementar la aplicación, poniendo un gran énfasis en proporcionar una interfaz de usuario intuitiva y fácil de usar. Para el back-end, se ha utilizado Java Spring Boot y JPA para su implementación, adoptando una arquitectura hexagonal bien organizada, que permite una gestión sencilla y escalabilidad del sistema. Esta estructura de la aplicación proporciona una base sólida para el mantenimiento futuro y la posibilidad de agregar nuevas funcionalidades.

Índex

1 - Objecte del TFG

1.1 - Antecedents

Warhammer40K és un joc de taula (Wargame) ambientat en un futur distòpic de temàtica de Ciència-Ficció. Dos jugadors munten, pinten i formen un exercit representat per miniatures. Cada exercit té les seves pròpies regles i habilitats, i amb aquests exercits, tots dos jugadors juguen una partida. Aquesta partida consisteix a realitzar objectius i missions, i a eliminar les miniatures de l'oponent, tot això utilitzant les habilitats especials de les miniatures de cada exercit. Les habilitats de les miniatures i la interacció entre jugadors (atacs, defenses etc.) es resol llançant daus i superant un valor en concret. En aquest Wargame es realitzen partides úniques o tornejos competitius, tant individuals com per equips i part fonamental en qualsevol cas és la creació de llistes d'exercit. Cada jugador ha de seleccionar la quantitat de miniatures que vulgui fer servir durant les partides. Cada tipus de miniatura té associat un valor en punts, i la llista d'exercit esta limitada a una quantitat que vària en funció de l'aparellament. Cada jugador haurà d'anotar la quantitat i tipus de cada miniatura triada del seu exercit, i amb elles jugarà la/s partida/s.

1.2 - Objectiu

L'objectiu és fer una aplicació per a realitzar Llistes de Exercit de Warhammer40K (la llista de miniatures que conformen un exercit i determinen que jugues en cada partida), que cada usuari pugui crear les seves llistes i guardar-les en la base de dades del núvol, i que pugui compartir-les. També que, amb els resultats dels tornejos mes importants (afegits en la base de dades o extretes d'una API), l'aplicació sigui capaç de generar-te llistes amb mes Winrate o recomanar-te segons quines unitats per sobre d'altres basant-se en els resultats dels tornejos competitius.

Es tracta de realitzar una aplicació per a mòbil/web amb React Native per al Frontend, i usar Springboot i JPA per al Backend (seguint arquitectura hexagonal) i allotjat tot en el núvol.

1.3 - Justificació

1.3.1 - Perquè Warhammer?

Games Workshop (l'empresa que posseeix el joc) és una de les empreses mes rendibles del món i hi ha una autentica oportunitat en el que a creació de llistes d'exercit es refereix, ja que no hi ha aplicacions que realment funcionin correctament. Games Workshop te una aplicació oficial per fer llistes però aquesta te una pèssima valoració entre els usuaris perquè no funciona correctament (bugs, rendiment baix etc). La única alternativa es una aplicació creada per la comunitat que també te les seves mancances (bugs, baix nivell d'actualitzacions, poc manteniment...).

1.3.2 - Perquè React Native, Springboot, JPA i arquitectura hexagonal?

React Native perquè és un Framework que esta a l'ordre del dia, serveix tant per a iOS, com Android.

Una aplicació web no es adient per al tipus de desempeny que s'espera del producte ja que per jugar a Warhammer cal traslladar-se físicament per jugar-hi i portar un portàtil/pc no es pràctic. Això implica que la aplicació mòbil sigui la mes adient per aquest escenari ja que el mòbil es molt mes pràctic per consultar i dur enlloc.

Springboot és un Framework molt potent per al Backend, es molt popular per al desenvolupament d'aplicacions, és ràpid i senzill, i ofereix moltes possibilitats (rest, secutity etc). I JPA per a la persistència, ja que facilita molt la creació, administració i ús de la base de dades.

Arquitectura hexagonal perquè separar el sistema en infraestructura, aplicació i domini fa que tot estigui menys acoblat, i que sigui molt mes fàcil realitzar, crear, canviar i entendre el codi. A més es molt escalable, ja que es pot canviar la manera d'introduir dades o la pròpia base de dades.

2 - Estudi previ

2.1 - Estudi sobre Warhammer 40K

Warhammer 40,000 és un joc de taula d'estratègia en temps real que es juga amb miniatures i daus. El joc es desenvolupa en un futur llunyà on diverses races i faccions lluiten pel control de l'univers. Els jugadors assumeixen el control d'un exèrcit i competeixen contra altres jugadors en batalles èpiques. Les parts fonamentals per jugar una partida de Warhammer son les següents:

- Preparació: Abans de començar a jugar, els jugadors han de triar una facció o raça i construir el seu exèrcit. Això inclou la selecció d'unitats, equips i personatges importants, així com l'assignació de punts. Es a dir, crear una llista d'exercit.

- Tauler i terreny: El joc es juga en un tauler representatiu d'una batalla. Els jugadors han d'establir el terreny en el tauler, incloent-hi edificis, obstacles i altres elements d'escenografia que afecten al joc.

- Torns: El joc es divideix en torns alterns, durant els quals cada jugador pren torns per a moure les seves unitats, disparar i combatre.

- Moviment: Durant el seu torn, els jugadors poden moure les seves unitats a través del terreny. Cada unitat té un rang de moviment i pot veure's afectada pel terreny i altres obstacles.

- Dispar: Durant el seu torn, els jugadors també poden fer que les seves unitats disparin contra les unitats enemigues. Cada unitat té un abast i una potència de foc únics, i els resultats dels trets es determinen a través de la tirada de daus.

- Combat: Si dues unitats entren en contacte, es lliurarà un combat cos a cos. Els resultats del combat es determinen a través de la tirada de daus i la comparació de les habilitats i estadístiques de les unitats involucrades.

- Comandant: Cada jugador també té un personatge important conegut com a Comandant, que és responsable de dirigir i motivar a l'exèrcit. Els Comandants tenen habilitats i estadístiques úniques que poden afectar el resultat de la batalla.

- Objectius: L'objectiu del joc és destruir l'exèrcit enemic o complir amb un objectiu específic, com capturar un objecte o controlar una àrea determinada del tauler.

Crear una llista d'exèrcit per a Warhammer 40,000 és un procés que requereix de planificació i coneixement sobre les diferents faccions i unitats disponibles en el joc. Els passos per crear una llista d'exercit son:

- Triar una facció o raça: Warhammer 40,000 inclou una àmplia varietat de faccions i races, cadascuna amb les seves pròpies fortaleses i febleses. Triar una facció o raça és el primer pas per a crear una llista d'exèrcit.

- Conèixer les regles: És important conèixer les regles i límits de cada facció o raça abans de crear una llista d'exèrcit. Per exemple, algunes faccions tenen límits en el nombre d'unitats d'un tipus específic que es poden incloure en una llista d'exèrcit.

- Determinar la estratègia de combat: Quina estratègia es vol adoptar en les batalles. Pot ser enfocar-se en la velocitat i la maniobrabilitat, o donar un enfocament més defensiu i de sosteniment. L'elecció de l'estratègia afectarà la forma en què es construeix la llista d'exèrcit.

- Seleccionar les unitats: Cal seleccionar les unitats del exercit que tant individualment com en conjunt, siguin capaces de dur a terme de la estratègia plantejada de la forma mes eficient possible.

- Complir amb els requisits de punts: Cada unitat en Warhammer 40,000 té un cost en punts assignat. És important assegurar-se de complir amb els requisits de punts per a garantir una competència justa amb altres jugadors.

- Prova i ajust: Una vegada que s'ha creat la llista d'exèrcit, cal jugar alguns jocs per a veure com funciona en la pràctica. Si alguna cosa no està funcionant, es poden fer ajustos i provar diferents combinacions fins trobar la llista d'exèrcit mes eficient.

2.2 - Estudi metodologies de treball

S'ha realitzat un estudi previ per escollir la metodologia mes adient per dur a terme el desenvolupament de la aplicació. Les dues possibles opcions son Kanban i Scrum.

Avantatges de Kanban:

- Flexibilitat: Kanban permet una gran flexibilitat en la forma en què els projectes es duen a terme. És possible canviar els requisits, la freqüència i el nombre de tasques segons les necessitats.

- Visualització clara: Kanban permet una representació visual clara de les tasques pendents, en progrés i completades, la qual cosa facilita la planificació i el seguiment dels projectes.

- Millora contínua: Kanban permet una millora contínua, ja que es poden fer canvis en el flux de treball en qualsevol moment i ajustar-se segons les necessitats canviants del projecte.

Desavantatges de Kanban:

- Pot ser confús: La representació visual de Kanban pot ser confusa per a algunes persones, especialment si no estan familiaritzades amb el mètode.

- Limitacions en l'escalabilitat: Kanban pot ser menys efectiu per a projectes grans i complexos, ja que pot ser difícil d'escalar i mantenir un seguiment adequat.

- Requereix una bona implementació: Kanban requereix una implementació adequada i una comprensió profunda de com funciona per a obtenir resultats òptims.

Avantatges de Scrum:

- Enfocament en el lliurament continu: Scrum s'enfoca en el lliurament continu de productes i valor als clients.

- Millora contínua: Scrum permet la millora contínua a través de retroalimentació regular i la capacitat de fer ajustos en el procés en qualsevol moment.

- Major col·laboració: Scrum promou una major col·laboració i comunicació entre els membres de l'equip, inclòs el client, la qual cosa pot resultar en una millor qualitat del producte final.

- Transparència: Scrum proporciona una transparència en el progrés i la planificació del projecte, la qual cosa pot millorar la confiança del client i la presa de decisions més informades.

Desavantatges de Scrum:

- Corba d'aprenentatge: Scrum pot requerir un període d'adaptació per als membres de l'equip, ja que pot ser un marc de treball complex amb molts components diferents.

- Rigidesa en la planificació: Encara que Scrum permet la millora contínua, pot ser rígid en la planificació i el compliment dels temps, la qual cosa pot ser un desafiament per a alguns equips.

- Requereix una gran dedicació: Scrum requereix una gran dedicació per part dels membres de l'equip, inclòs el paper del Scrum Màster, per a implementar-ho adequadament.

- Pot ser més costós: Scrum pot ser més costós que altres marcs de treball degut a la necessitat de reunions regulars i la documentació exhaustiva.

Conclusió i elecció:

S'ha escollit Kanban. La raó principal es perquè la metodologia Scrum es inherentment una metodologia de grup i requereix la figura del client, i no es l'escenari d'aquest projecte. A mes, les dimensions del projecte fan que Kanban sigui una opció idònia.

2.3 - Estudi Frontend

S'ha realitzat un estudi previ per escollir l'eina mes adient per dur a terme el Frontend de la aplicació. Les dues possibles opcions son React Native i Android Studio.

Avantatges de React Native:

- Desenvolupament mòbil multiplataforma: React Native permet desenvolupar aplicacions tant per iOS com per Android amb el mateix codi, reduint el temps i costos de desenvolupament

- Rendiment: React Native utilitza components natius en lloc de elements simulats, això implica un rendiment mes ràpid i fluid.

- Interfície d'usuari nativa: Les aplicacions desenvolupades amb React Native tenen una aparença similar a les aplicacions natives, això millora l'experiència d'usuari.

- Comunitat de desenvolupament: React Native te una gran comunitat de desenvolupadors i una gran quantitat de recursos i documentació disponible.

Desavantatges de React Native:

- Aprenentatge: El JavaScript fet servit per React Native pot resultar una barrera per nous desenvolupadors

- Personalització limitada: La personalització de la interfície d'usuari i la integració amb components natius es menys flexible que el desenvolupament natiu.

- Dificultat per manegar els errors: La depuració i el seguiment de errors es difícil en React Native degut a la complexitat del entorn de desenvolupament i la interacció entre components natius i no natius.

- Desactualització de components: Amb cada nova versió del sistema operatiu o del navegador, alguns component poden quedar obsolets.

Avantatges Android Studio

- Facilitat per manegar els errors: Android Studio te un conjunt de eines per depuració i proba d'aplicacions que permet als desenvolupadors identificar i solucionar problemes amb mes facilitat

- Interfície d'usuari intuïtiva: Android Studio ofereix una interfície d'usuari intuïtiva i personalitzable això permet treballar de forma eficient i còmoda als desenvolupadors

- Integració amb eines de tercers: Android Studio permet la integració amb una amplia varietat de eines de tercers com Firebase, Github etc

- Desenvolupament natiu: Android Studio esta dissenyat específicament pel desenvolupament de aplicacions per Android.

Desavantatges Android Studio

- Requisits de hardware: Android Studio pot ser una aplicació pesada i requerir un terminal amb especificacions mes altes per funcionar de manera eficient

- Corba d'aprenentatge: Android Studio es una aplicació complexa que pot requerir molt temps i esforç per aprendre les seves funcions

- Lentitud en la compilació: La compilació de les aplicacions en Android Studio poden ser lentes

- Incompatibilitat de versions: Poden sorgir problemes de compatibilitat amb versions mes antigues o noves de Android Studio, i això pot requerir actualitzacions o reinstal·lació del software.

Conclusió i elecció:

S'ha escollit React-Native. La raó principal es perquè limita la compatibilitat a terminals Android, i l'objectiu es arribar al màxim numero d'usuaris possibles.

2.4 - Estudi Backend

S'ha realitzat un estudi previ per escollir l'eina mes adient per dur a terme el Backend de la aplicació i s'ha escollit Java Springboot. Spring Boot és un Framework d'aplicacions Java que s'utilitza per a desenvolupar aplicacions de manera ràpida i eficient, i s'enfoca en la simplificació del desenvolupament i la configuració d'aplicacions, proporcionant una sèrie d'eines i característiques que permeten als desenvolupadors crear aplicacions en poc temps. Les característiques mes rellevants son:

- Configuració fàcil: Spring Boot proporciona una configuració automàtica dels components de l'aplicació i una integració rapida amb altres tecnologies, això permet al desenvolupador treballar de forma mes eficient

- Alta productivitat: Spring Boot proporciona un conjunt d'eines i característiques que permeten al desenvolupador escriure i provar codi de manera rapida i senzilla

- Escalabilitat: Spring Boot es altament escalable i això permet al desenvolupador adaptar-se a la demanda dels clients

- Comunitat Java: Spring Boot te una comunitat de desenvolupadors activa i una amplia quantitat de recursos en línia, això permet al desenvolupador resoldre problemes i aprendre noves habilitats de manera eficient

2.5 - Estudi Persistència

2.5.1 - Base de Dades

S'ha realitzat un estudi previ per escollir l'eina mes adient per dur a terme el la persistència de la aplicació. Les dues possibles opcions son els dos tipus de base de dades, relacional (MariaDB) i no relacional (DynamoDB).

Avantatges de base de dades relacional:

- Integritat de les dades: Les BBDD relacionals permeten al desenvolupador garantitzar la integritat de les dades mitjançant la implementació de restriccions com claus primàries i foranies

- Flexibilitat: Les BBDD relacionals permeten al desenvolupador modelar i emmagatzemar dades de manera flexible i escalable

- Consultes potents: Les BBDD relacionals permeten al desenvolupador realitzar consultes complexes i personalitzades amb ajuda de llenguatges de consulta estandarditzats com SQL

- Integració fàcil: Les BBDD relacionals s'integren fàcilment amb altres sistemes i tecnologies

Desavantatges de base de dades relacional:

- Dificultat de maneg de dades no estructurades: Les BBDD relacionals tenen dificultats per manegar dades no estructurades com imatges o vídeos

- Dificultat per escalar horitzontalment: Les BBDD relacionals tenen una capacitat limitada per manegar gran quantitats de dades

Avantatges de base de dades no relacional:

- Maneg de dades no estructurades: Les BBDD no relacionals poden manegar dades no estructurades com imatges o vídeos

- Escalabilitat horitzontal: Les BBDD no relacionals son fàcils d'escalar en quantitat de dades

- Temps de resposta ràpids: Les BBDD no relacionals son capaces de manegar gran quantitat de dades amb temps de resposta ràpids

Desavantatges de base de dades no relacional:

- Integritat de dades: Les BBDD no relacionals no garantitzen la integritat de les dades

- Consultes complexes: Les BBDD no relacionals no son capaces de dur a terme consultes complexes o personalitzades de forma eficient

- Integració amb altres sistemes: Les BBDD no relacionals son difícils d'integrar amb altres sistemes i tecnologies

Conclusió i elecció:

S'ha escollit la base de dades relacional (MariaDB). Le principals raons son el grau d'experiència sobre aquesta tecnologia i que es valora la robustesa, flexibilitat i escalabilitat, i no tant els temps de resposta, la escalabilitat horitzontal o una gran quantitat de dades

2.5.2 - Mapeig Base de Dades

S'ha realitzat un estudi previ sobre l'eina Java Persistence API (JPA) per incorporar-la a la aplicació. JPA és una especificació de Java que proporciona una forma de mapejar objectes Java a taules de bases de dades i viceversa. Ofereix una gestió automàtica del cicle de vida de les entitats, una API portàtil, flexibilitat i compatibilitat amb altres marcs. Les característiques mes rellevants son:

- Mapatge objecte-relacional: JPA permet mapejar objectes Java a taules de bases de dades i viceversa. Això significa que els desenvolupadors poden interactuar amb les bases de dades usant objectes Java en lloc de sentències SQL.

- Integració amb el cicle de vida de l'entitat: JPA proporciona una gestió automàtica del cicle de vida de les entitats, cosa que significa que els desenvolupadors no han d'escriure codi manual per a realitzar operacions com la inserció, l'actualització o l'eliminació de dades.

- API portàtil: JPA és una API estandarditzada que es pot usar en diferents plataformes i entorns de desenvolupament. Això significa que els desenvolupadors poden moure les seves aplicacions a diferents plataformes sense haver de reescriure el codi.

- Flexibilitat: JPA ofereix moltes opcions i configuracions que permeten als desenvolupadors personalitzar i optimitzar la seva implementació per als seus requisits específics.

- Integració amb altres marcs: JPA és compatible amb altres marcs Java, com Spring, que proporcionen característiques addicionals com la injecció de dependències o la gestió de transaccions.

2.6 - Estudi Arquitectura del Software

S'ha realitzat un estudi previ per escollir l'arquitectura de software mes adient pel desenvolupament de la aplicació. S'ha escollit l'Arquitectura Hexagonal.

L’Arquitectura Hexagonal es un conjunt de patrons de software que es centra en la independència, la flexibilitat i la prova automatitzada. Permet als desenvolupadors crear sistemes robustos, adaptables i fàcilment probables que poden evolucionar amb el temps. Les característiques mes rellevants son:

- Centrat en el domini: L'arquitectura hexagonal es centra en el domini del sistema, és a dir, en el negoci o en el problema que s'està tractant de resoldre. L'objectiu és aïllar i protegir el domini del sistema de les influències externes, com les tecnologies o les plataformes subjacents.

- Interfície d'adaptadors: L'arquitectura hexagonal es basa en una interfície d'adaptadors, que actua com un pont entre el nucli del sistema i el seu entorn extern. Els adaptadors permeten que el nucli del sistema es comuniqui amb els components externs, com les bases de dades, les aplicacions externes o els usuaris, sense afectar la integritat del domini.

- Independència de tecnologia: L'arquitectura hexagonal permet als desenvolupadors canviar la tecnologia subjacent sense afectar el nucli del sistema. Això significa que el sistema pot evolucionar i adaptar-se a les noves tecnologies sense haver de reescriure tot el codi.

- Proves automatitzades: L'arquitectura hexagonal promou la implementació de proves automatitzades, que permeten als desenvolupadors provar el sistema de manera ràpida i de confiança. Això ajuda a millorar la qualitat i l'estabilitat del sistema a llarg termini.

- Flexibilitat: L'arquitectura hexagonal permet als desenvolupadors afegir noves funcionalitats i modificar les existents de manera senzilla i flexible. Això significa que el sistema pot evolucionar i adaptar-se als canvis en el negoci o en els requisits de l'usuari sense haver de reescriure tot el codi.

2.7 - Estudi Contenidors

S'ha realitzat un estudi previ per escollir l'eina mes adient per facilitar el manteniment dels servidors en producció i s'ha escollit Docker.

Docker és una plataforma de contenidors que permet desenvolupar, implementar i executar aplicacions en qualsevol entorn. Les característiques mes rellevants son:

- Portabilitat: Docker permet empaquetar aplicacions i les seves dependències en contenidors, la qual cosa facilita la implementació d'aplicacions en diferents entorns i sistemes operatius.

- Escalabilitat: Docker permet als desenvolupadors crear múltiples contenidors d'una mateixa aplicació i escalar-los de manera independent, la qual cosa facilita la gestió de l'escalabilitat.

- Isolació: Docker permet aïllar les aplicacions i les seves dependències en contenidors, la qual cosa millora la seguretat i evita conflictes entre diferents aplicacions.

- Automatització: Docker permet automatitzar la implementació i desplegament d'aplicacions, la qual cosa estalvia temps i redueix errors humans.

2.8 - Estudi Entorn de Desenvolupament

S'ha realitzat un estudi previ per escollir l'eina mes adient per l'entorn de desenvolupament de la aplicació. S'ha escollit Visual Studio Code per al Frontend i IntelliJ per al Backend.

IntelliJ és un entorn de desenvolupament integrat (IDE) de Java que ofereix una sèrie de característiques rellevants per als desenvolupadors Java.

- Autocompletat intel·ligent: IntelliJ ofereix un autocompletat intel·ligent que ajuda els desenvolupadors a escriure codi més ràpid i amb major precisió

- Depuració i perfilat: IntelliJ inclou eines avançades per a depurar aplicacions Java, la qual cosa permet als desenvolupadors trobar i solucionar errors més fàcilment

- Integració amb altres marcs de treball: IntelliJ ofereix una integració fàcil amb altres marcs de treball com Spring, la qual cosa permet als desenvolupadors treballar amb múltiples tecnologies i marcs de treball des d'una sola plataforma

- Refactorització: IntelliJ ofereix eines de refactorització avançades que ajuden els desenvolupadors a reorganitzar i millorar el seu codi de manera eficient

- Anàlisi de codi: IntelliJ ofereix una anàlisi de codi en temps real que detecta i corregeix errors i problemes en el codi mentre s'escriu.

Visual Studio Code (VSCode) és un editor de codi gratuït i multiplataforma desenvolupat per Microsoft. Les seves característiques rellevants son:

- Interfície d'usuari intuïtiva: VSCode ofereix una interfície d'usuari intuïtiva i fàcil d'usar que permet als desenvolupadors centrar-se en el seu codi sense distraccions

- Autocompletat: VSCode ofereix un auto completat intel·ligent que ajuda els desenvolupadors a escriure codi més ràpid i amb major precisió

- Integració amb Git: VSCode inclou integració nativa amb Git, la qual cosa permet als desenvolupadors treballar de manera col·laborativa i controlar les versions del seu codi

- Extensibilitat: VSCode és altament extensible, cosa que significa que els desenvolupadors poden agregar funcions addicionals mitjançant l'ús d'extensions desenvolupades per la comunitat

- Depuració: VSCode ofereix eines de depuració avançades que ajuden els desenvolupadors a trobar i solucionar errors en el seu codi

- Multiplataforma: VSCode és un editor de codi multiplataforma que s'executa en Windows, macOS i Linux

- Lleuger: VSCode és un editor de codi lleuger que s'executa ràpidament en computadors de baixos recursos

S'ha realitzat un estudi previ per poder simular la posada en producció de la plataforma i la eina que es fa servir es XAMPP. XAMPP és una distribució de programari lliure que permet als desenvolupadors crear i provar aplicacions web en un entorn local. Les seves característiques rellevants son:

- Entorn de desenvolupament integral: XAMPP inclou Apatxe, MySQL, PHP i Perl en un paquet fàcil d'usar, la qual cosa permet als desenvolupadors crear i provar aplicacions web en el seu propi equip

- Instal·lació fàcil: XAMPP és fàcil d'instal·lar i configurar, cosa que significa que els desenvolupadors poden tenir un entorn de desenvolupament funcionant en poc temps

- Lliure: XAMPP és un programari lliure i de codi obert, cosa que significa que els desenvolupadors poden usar-ho i modificar-ho sense restriccions

- Eines d'administració: XAMPP inclou una sèrie d'eines d'administració fàcils d'usar que permeten als desenvolupadors administrar i mantenir el seu entorn de desenvolupament

- Gratuït: XAMPP és gratuït i no requereix la compra de llicències o el pagament de quotes mensuals.

2.9 - Estudi Control de Versions

S'ha realitzat un estudi previ per escollir l'eina mes adient per el control de versions i s'ha escollit GitHub. GitHub és una plataforma integral que permet als desenvolupadors col·laborar i millorar el seu codi de manera eficient. Les principals característiques de GitHub son les següents:

- Allotjament de codi: GitHub és una plataforma d'allotjament de codi en línia que permet als desenvolupadors allotjar, controlar i compartir el seu codi amb uns altres

- Control de versions: GitHub ofereix control de versions basat en Git, la qual cosa permet als desenvolupadors rastrejar i revertir canvis en el seu codi

- Eines de seguiment de problemes: GitHub inclou eines de seguiment de problemes que permeten als desenvolupadors rastrejar i solucionar problemes en el seu codi

- Integració amb altres plataformes: GitHub s'integra amb una àmplia varietat de plataformes i serveis, la qual cosa permet als desenvolupadors unir el seu codi amb altres eines que utilitzen

- Gratuït: GitHub ofereix un compte gratuït que permet als desenvolupadors allotjar i col·laborar en projectes públics sense cap cost

2.10 - Estudi Plataformes de Producció

S'ha realitzat un estudi previ per escollir la plataforma de producció mes adient. Les dues possibles opcions son Amazon Web Services i Azure. Les raons son les següents:

Avantatges de AWS:

- Escalabilitat: AWS permet als usuaris escalar ràpidament els seus recursos en línia segons les seves necessitats

- Flexibilitat: AWS ofereix una àmplia gamma de serveis en línia, cosa que significa que els usuaris poden seleccionar i usar només els serveis que necessiten

- Disponibilitat: AWS és una plataforma en el núvol altament disponible, cosa que significa que els usuaris poden accedir als seus serveis en línia en qualsevol moment i des de qualsevol lloc.

- Seguretat: AWS ofereix una àmplia gamma d'opcions de seguretat per a protegir les dades i els recursos en línia dels usuaris.

- Cost: AWS és generalment més assequible que els sistemes tradicionals de IT, ja que només paguen pels serveis que usen.

Desavantatges de AWS:

- Complexitat: AWS pot ser complex d'usar, especialment per a aquells que no tenen experiència prèvia amb el núvol.
- Càrrega de treball: AWS requereix una càrrega de treball addicional per a la configuració i el manteniment dels recursos en línia.
- Compatibilitat: Alguns serveis o aplicacions poden no ser compatibles amb AWS.
- Costos ocults: Alguns costos addicionals poden ser incorreguts per l'ús de serveis addicionals o la transmissió de dades.

Avantatges Azure

- Integració: Azure és compatible amb una àmplia gamma de tecnologies i plataformes, la qual cosa permet als usuaris integrar fàcilment les seves aplicacions i serveis amb altres sistemes.
- Escalabilitat: Azure permet als usuaris escalar ràpidament els seus recursos en línia segons les seves necessitats.
- Fiabilitat: Azure és una plataforma altament de confiança que garanteix la disponibilitat dels serveis en línia per als usuaris.
- Seguretat: Azure ofereix una àmplia gamma d'opcions de seguretat per a protegir les dades i els recursos en línia dels usuaris.
- Eines de desenvolupament: Azure ofereix una àmplia gamma d'eines de desenvolupament i plataformes per a ajudar els desenvolupadors a crear i implementar les seves aplicacions i serveis en línia.

Desavantatges de Azure:

- Cost: Azure pot ser més costós que algunes altres plataformes en el núvol.
- Complexitat: Azure pot ser complex d'usar, especialment per a aquells que no tenen experiència prèvia amb el núvol.

- Càrrega de treball: Azure requereix una càrrega de treball addicional per a la configuració i el manteniment dels recursos en línia.

- Integració limitada: Alguns serveis o aplicacions poden no ser compatibles amb Azure.

Conclusió i elecció:

S'ha escollit AWS per la flexibilitat, escalabilitat i coneixement de la plataforma. A mes, el cost es un punt determinant per rebutjar Azure.

3 - Objectius i abast

L'objectiu principal es crear una aplicació per a realitzar Llistes de Exercit de Warhammer per a mòbil/web amb React Native per al Frontend, Springboot i JPA per al Backend (seguint arquitectura hexagonal) i allotjat tot en el núvol.

Els clients objectius/finals de la plataforma son tots els jugadors del joc Warhammer 40K, ja que tant si son jugadors novells o competitius, necessiten una eina per crear llistes d'exercit.

El públic potencial son:

- Les organitzacions que munten tornejos de Warhammer40K que, veient la utilitat de la plataforma, exigeixin als seus participants a fer-la servir.
- La pròpia empresa Game Workshop que li pot interessar la plataforma per al seu joc i pels seus altres productes (Age of Sigmar, jocs especialistes etc)
- Els jugadors, organitzacions o empreses de altres Wargames que segueixen un model similar en la creació de llistes

Objectius del producte:

- Oferir una creació de llistes d'exercit senzilla
- Oferir una correcció de llistes acurada
- Oferir totes les regles del exercit de forma centralitzada
- Oferir una base de dades actualitzada i fiable
- Oferir totes les llistes d'exercit creades en una sola aplicació
- Oferir material de consulta sobre la llista i l'exercit durant la partida

Objectius del Client:

- Englobar en una sola aplicació totes les llistes d'exercit d'una o mes faccions

- Facilitar la compartició de llistes amb organitzacions o amics/clubs de joc
- Reduir la barrera d'accés al joc encapsulant totes les regles i canvis de punts actualitzats del joc en un únic gestor de llistes
- Simplificar la creació de llistes d'exercit
- Millorar el nivell competitiu de les llistes d'exercit
- Evitar llistes d'exercit errònies.

L'abast de la aplicació es limitarà a la creació de llistes del joc Warhammer 40K, i només de les 3 faccions mes importants del joc (Space Marines, Eldar i Chaos Space Marines). Aquesta limitació es deguda a que pel temps i recursos disponibles per a la realització del TFG, no es assequible afegir mes faccions ni ampliar els jocs disponibles.

4 - Metodologia

L'estratègia per intentar aconseguir el desenvolupament d'una aplicació i TFG exitós segueix el següent patró d'actuació:

- Cerca d'informació: Degut a que l'aplicació es basa en un Wargame i el públic objectiu es un nínxol, no hi ha prou bases de dades acadèmiques i s'han estudiat diverses fonts com publicacions oficials de la companyia propietària del Wargame (Games Workshop), publicacions de blogs de la comunitat del joc, feedback de jugadors i clubs de joc, i l'experiència pròpia com a jugador, client de la futura plataforma i client de les alternatives fins el moment.

- Anàlisi de la informació: Aquesta informació recopilada s'ha avaluat segons la seva rellevància i s'ha descartat el contingut purament subjectiu.

- Síntesi de la informació, avaluació i tria d'alternatives: D'aquesta informació s'ha realitzat una síntesi que ha permès determinar els requeriments funcionals de la aplicació.

Tant pel desenvolupament de la aplicació com del TFG en general, s'ha escollit una metodologia àgil, concretament Kanban.

Kanban es una metodologia de gestió de projectes que es basa en visualitzar el flux de treball i limitar el numero de tasques en progres per mantenir una cadència constant de feina i reduir el temps d'espera entre desenvolupaments de les funcionalitats.

Per dur a terme el correcte seguiment de les tasques, s'utilitza el següent:

- Tauler Kanban: On es representa visualment el flux de feina amb diferents columnes en funció del diferent estat de la tasca. Aquets estats son "To-Do" per les tasques pendents de fer, "Work in Progress" per les tasques que s'estan desenvolupant en aquell moment i "Done" per les tasques ja finalitzades.

- Limitació del Work in Progress: S'estableix un límit en la quantitat de tasques que es permeten en cada columna per evitar la sobrecarrega i mantenir un flux de feina constant.

- Flux de Pull: Les tasques nomes es mouen a la següent columna quan s'han completat tots els requisits per començar a desenvolupar-les o si s'han completat tots els requeriments per donar-la per completada. Això serveix per centrar-se en completar les tasques abans d'acceptar de noves.

- Retroalimentació i millora continua: S'ha de revisar regularment els canvis en l'entorn i les demandes del mercat per poder avaluar i identificar obstacles i oportunitats de millora, i realitzar ajustaments en conseqüència basats en les dades i l'observació.

5 - Definició de requeriments funcionals i tecnològics

5.1 - Requeriments funcionals

- Buscar per nom una llista en particular entre les llistes creades
- Crear una llista d'exercit
- Seleccionar facció i subfacció de l'exercit
- Seleccionar tipus de destacament principal i secundaris
- Posar nom a la llista
- Establir el límit de punts i el límit de punts de comando
- Seleccionar mida de la partida i rebre els beneficis en punts de comando pertinents
- Afegir unitats en funció del seu rol de batalla (HQ, TROOPS, ELITES, FAST ATTACK, HEAVY SUPPORT, FLYER, DEDICATED TRANSPORT, LORD OF WAR, FORTIFICATION, SUPREME COMMANDER)
- Modificar el numero, l'equip i les habilitats de les unitats amb opcions modificables.
- Notificar quants punts i punts de comando manquen per arribar al màxim i si aquest s'excedeix
- Notificar els possibles errors en la creació de la llista (habilitats úniques o equip únic duplicat, excedit el màxim d'una peça de equip per unitat/membres d'unitat...)
- Visualitzar la quantitat de punts i punts de comando de la llista
- Visualitzar beneficis i regles especials de la facció/subfacció.
- Visualitzar el cost de punts i punts de comando que te cada destacament, estratagema, unitat, habilitat etc
- Visualitzar nom, atributs, habilitats i punts de cada unitat/arma

- Visualitzar regles de facció i subfacció
- Determinar un percentatge límit de Winrate i recomanar unitats amb un Winrate superior

5.2 - Requeriments tecnològics

- Sistema de gestió de dades per a emmagatzemar i cercar llistes
- Base de dades de llistes, regles de faccions i subfaccions, unitats, destacaments, habilitats i equips etc
- Lògica de càlcul per a determinar el límit de punts i punts de comando, i per a notificar si s'excedeix
- Validació de dades per a detectar errors en la creació de la llista
- Interfície d'usuari per a la creació i selecció de llistes, així com per a la selecció de facció i subfacció, destacaments, unitats, equipament, habilitats etc
- Interfície d'usuari per a visualitzar la informació de la llista, com el cost de punts i punts de comando, el nom, els atributs, les habilitats de les unitats etc
- Sistema d'anàlisi de Winrate per a determinar un límit de Winrate i recomanar unitats amb un Winrate superior

6 - Planificació i control de versions

6.1 - Kanban

La planificació del desenvolupament s'ha realitzat mitjançant un taulell Kanban i seguint la ruta de la planificació de treball establerta.

S'ha creat un Trello per representar el Taulell Kanban i s'han afegit els To-Do a mesura que hi havia un requeriment al qual donar solució.

Com a limitació de tasques en Work in Progress s'ha establert un màxim de cinc tasques concurrents, però s'han acceptat excepcions quan aquestes tasques requerien d'altres funcionalitats no desenvolupades però necessàries per al seu funcionament o testeig.

Durant el transcurs del desenvolupament, s'ha seguit el Flux de Pull correctament ja que no s'ha mogut una tasca de columna fins completar els requeriments per començar a desenvolupar-la o completar els requeriments establerts per donar-la per acabada.

Quan s'ha determinat una actualització o reestructuració d'una funcionalitat, s'ha mogut aquesta tasca de nou a la columna de To-Do.

Per diferenciar ràpidament el tipus de tasca, s'ha establert el següent conjunt d'etiquetes (Fig. 6.1.1). Aquestes etiquetes segueixen un codi de colors per facilitar la seva lectura i comprensió, i s'agrupen per categories.

- Entorn:
    - Frontend: Tasques de desenvolupament del Frontend de l'aplicació.
    - Backend: Tasques de desenvolupament del Backend de l'aplicació.
    - Memory: Tasques de la redacció de la memòria del projecte.
- Tipus de funcionalitat:
    - Enhance: Tasques per millorar l'aplicació. Afegir funcionalitats.
    - Bug: Tasques per corregir errors d'altres tasques completades.
    - Update: Tasques per actualitzar les tasques completades per satisfer demandes del mercat i oportunitats de millora.
- Preparació:

- Research: Tasques per investigar com desenvolupar altres tasques.
- Design: Tasques per determinar el disseny de l'aplicació i l'UX.

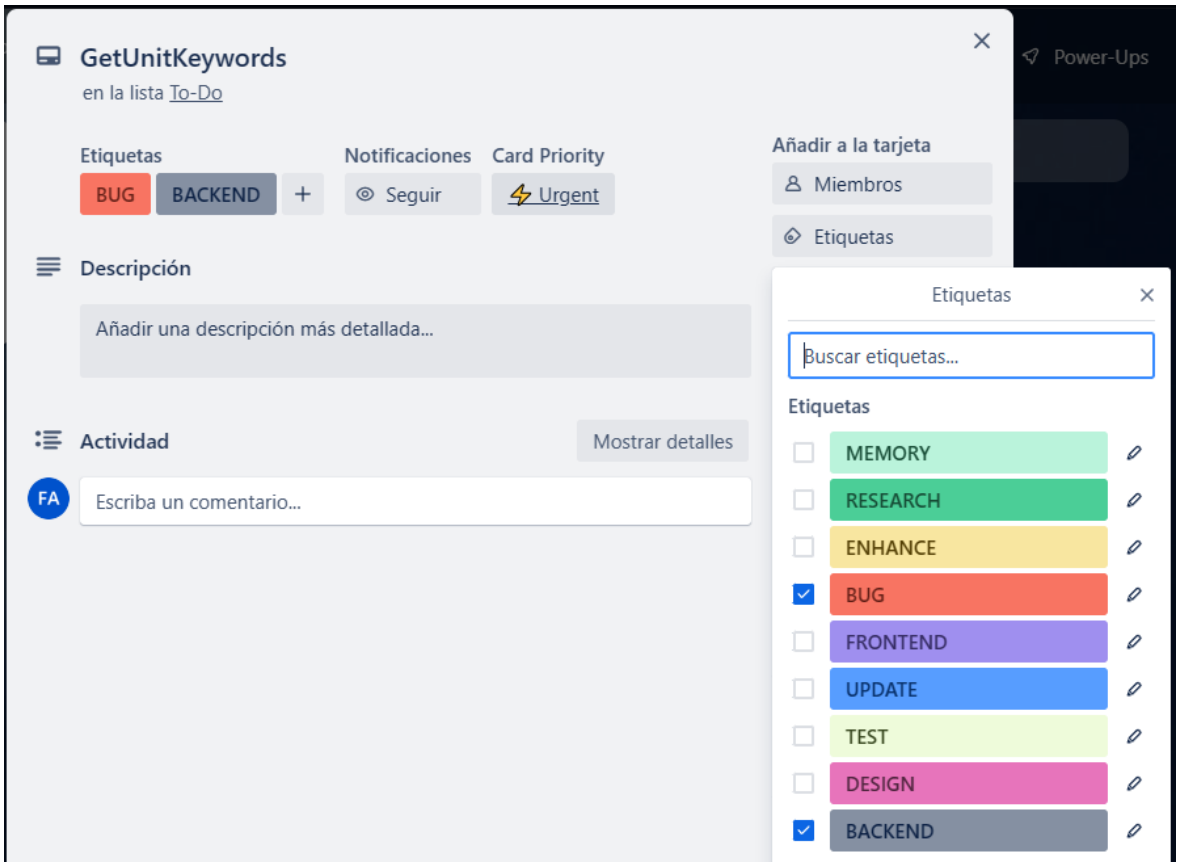

Fig. 6.1.1. Etiquetes.

Per determinar la prioritat en l'ordre de selecció de tasca, s'ha establert un codi d'icones (Fig.6.1.2.). Aquests icones representen la urgència per desenvolupar la tasca en els termes següents:

- Critical: Les tasques critiques son crucials pel èxit del projecte i s'han de completar immediatament. Un retràs en aquestes tasques pot representar un impacte negatiu en el desenvolupament i producte final. Implica dedicar recursos i temps elevat.
- Urgent: Les tasques urgents s'han de completar lo abans possible. Un retràs en aquestes tasques pot suposar problemes al projecte. Implica dedicar recursos i temps considerable.

- Important: Les tasques importants s'han de realitzar en un temps raonable. Aquestes tasques no son tan prioritàries com les altres però contribueixen al èxit general del projecte.
- Low: Les tasques de prioritat baixa es poden posposar sense afectar negativament al projecte i s'han de realitzar si no hi ha tasques mes prioritàries al davant.

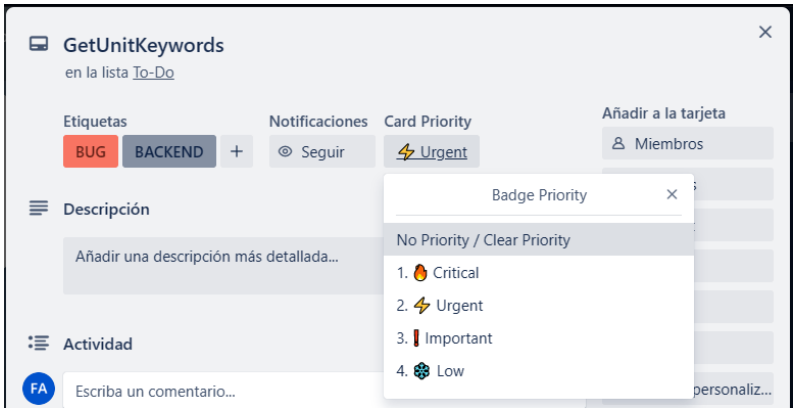

Fig.6.1.2.Prioritat.

Durant tot el transcurs del projecte, s'han revisat regularment les tasques desenvolupades i la tendència del mercat per proposar millores, corregir Bugs, afegir funcionalitats o superar incidències.

6.2 - Github

S'ha creat un repositori a GitHub per portar el control de versions de l'aplicació. El protocol seguit per desenvolupar cada tasca relacionada amb el codi es el següent:

- Crear una branca amb el nom de la tasca amb l'etiqueta de tipus al principi. Ex: *bug_Get-Unit-Keywords.*
- Desenvolupar la tasca realitzant *Commits* amb regularitat i amb un nom clar i explicatiu.
- Un cop finalitzada la tasca, fer Merge de la branca a Main.

7 – Desenvolupament

7.1 – Model de domini i base de dades

S'ha elaborat un diagrama UML (Fig.7.1.1.) amb les classes necessàries i les seves relacions per donar solució als requeriments funcionals de l'aplicació i poder desenvolupar la base de dades.

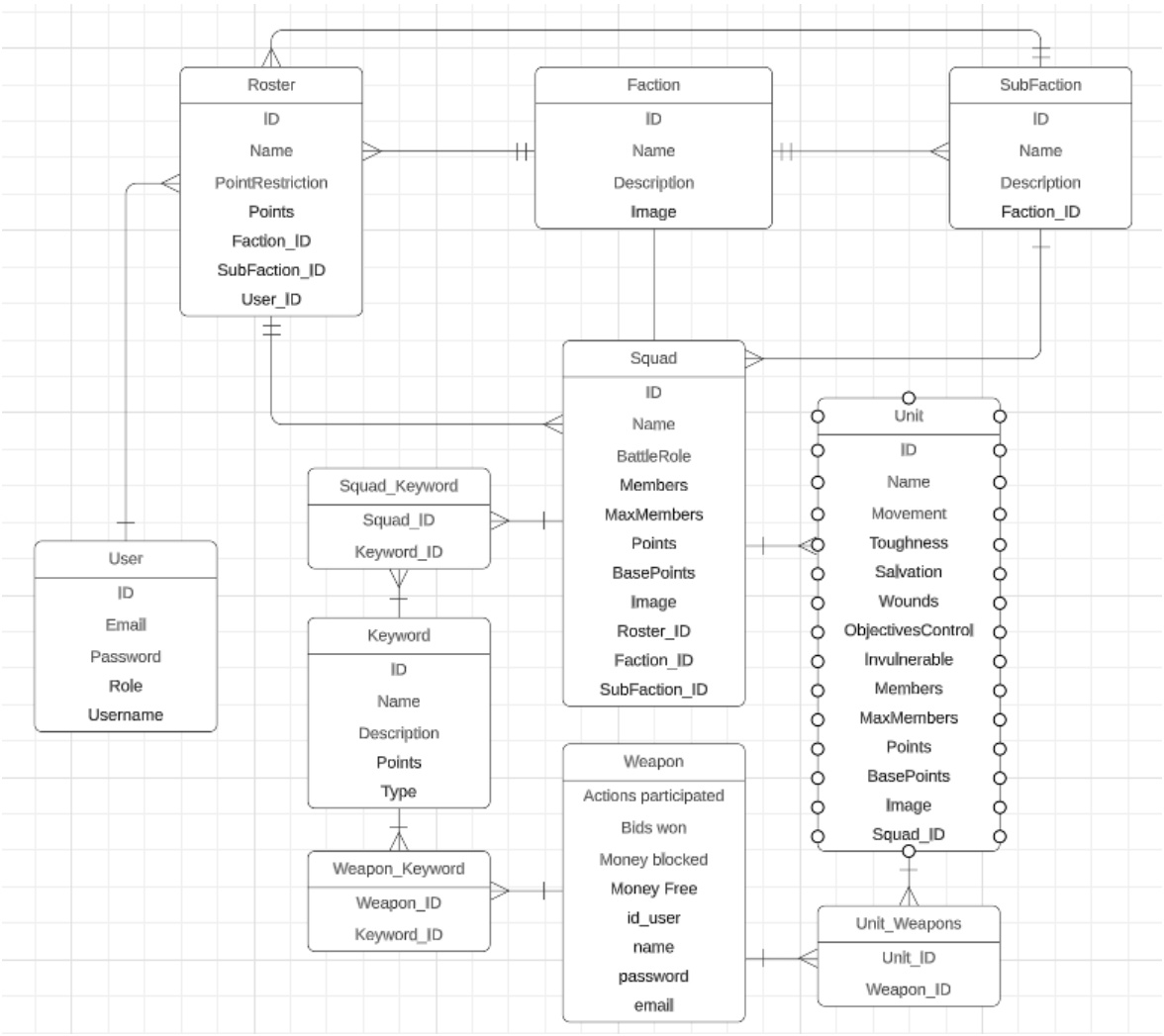

Fig. 7.1.1. Diagrama UML

Les classes del diagrama UML implementades a la base de dades del Backend (amb JPA) son les següents:

- Keywords: Classe que implementa les diferents paraules clau del reglament. Aquestes paraules clau representen les habilitats especials de les unitats, armes i equip, o faccions.
    - Atributs: ID, NAME, DESCRIPTION,TYPE, POINTS. .
- Units: Classe que implementa les diferents unitats de cada unitat i les seves característiques.
    - Atributs: ID, INVULNERABLE, LEADERSHIP, MEMBERS, MAX_MEMBERS, MOVEMENT, NAME, OBJECTIVES_CONTROL, POINTS, BASE_POINTS, SALVATION, TOUGHNESS, WOUNDS, SQUAD_ID, IMAGE
- Squads: Classe que implementa les diferents esquadres de cada exercit i les seves característiques.
    - Atributs: ID, BATTLE_ROLE, MEMBERS, MAX_MEMBERS, NAME, POINTS, ID_FACTION, ID_SUBFACTION, BASE_POINTS, IMAGE.
- Weapons: Classe que implementa el perfil de les diferents armes que tenen les unitats de cada exercit.
    - Atributs: ID, ARMOR_PENETRATION, ATTACKS, DAMAGE, NAME, HIT_ROLL, POINTS, RANGE, FIGHT_SKILL, STRENGTH, WEAPON_TYPE.
- Factions: Classe que implementa les diferents faccions disponibles per exercit.
    - Atributs: ID, NAME, DESCRIPTION, IMAGE.
- Sub-factions: Classe que implementa les diferents sub-faccions disponibles de cada facció.
    - Atributs: ID, NAME, DESCRIPTION, FACTION_ID, IMAGE.
- Rosters: Classe que implementa el punt principal de l'aplicació i que engloba la resta de classes. Les llistes de exercit.
    - Atributs: ID, NAME, POINT_RESTRICTION, POINTS, .ID_FACTION, ID_SUB_FACTION, ID_USER
- Users: Classe que implementa els diferents usuaris de l'aplicació, per poder fer Login i crear, modificar, guardar i eliminar les seves llistes.
    - Atributs: ID, EMAIL, USERNAME, PASSWORD, ROLE_ID.

Per realitzar les relacions de N..M s'han creat les següents classes d'unió:

- Squads_keywords: Classe que emparella cada esquadra amb les seves keywords.
    - Atributs: ID_SQUAD, ID_KEYWORD.
- Units_weapons: Classe que emparella cada unitat amb les seves armes.
    - Atributs: ID_UNIT, ID_WEAPON.
- Weapons_keywords: Classe que emparella cada arma amb les seves keywords.
    - Atributs: ID_KEYWORD, ID_WEAPON.

Tots els atributs aquí nombrats referents a aspectes del joc Warhammer40K s'han explicat degudament al glossari.

Els atributs Image de les entitats Faction, SubFaction, Squad i Unit haurien de tenir la URL en la base de dades, com no esta la aplicació en producció i encara no es te un contenidor d'imatges, aquestes seran "hardcodejades" al front end.

7.2 – Backend

Per a la realització del Backend s'ha fet servir Java amb Spring Boot i JPA per crear una REST API. S'ha elaborat la distribució (Fig.7.2.1) de components seguint els principis de l'arquitectura hexagonal i emprant el patrons Command/Query i ViewModel, a mes totes les dependències s'autoinjecten a cada classe mitjançant les anotacions pertinents.

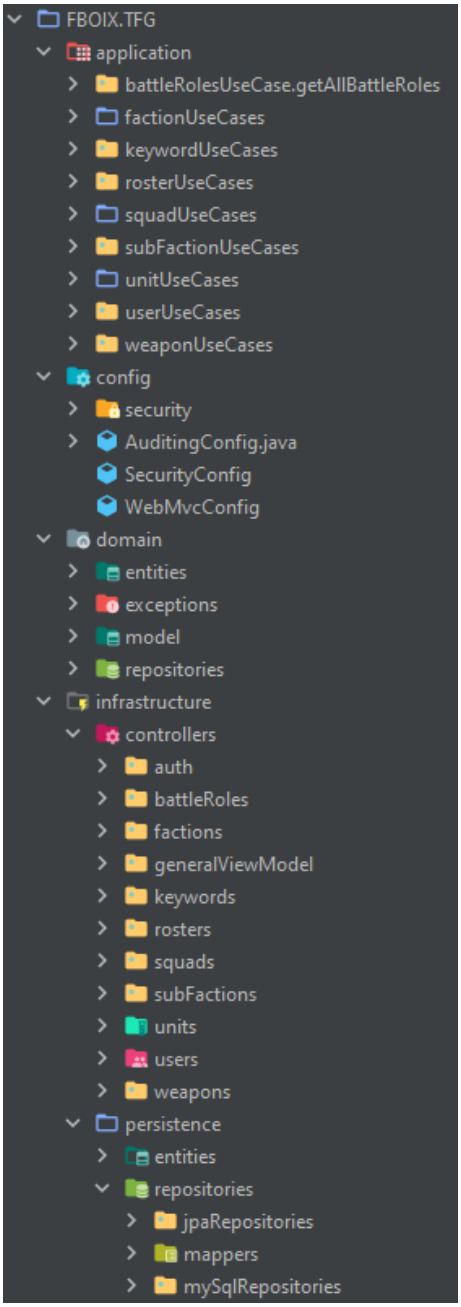

Fig. 7.2.1. Projecte.

El Backend es divideix en dues lògiques. L'ús dels usuaris convencionals i l'ús del administrador.

L'usuari administrador pot crear, modificar, eliminar i emmagatzemar els elements que posteriorment es fan servir com a model de creació de les llistes del usuari. Es creen les faccions, subfaccions, squads, unitats, keywords i Weapons que actuen com a base (Fig.7.2.2) i que quan l'usuari crea les seves, serveixen de model i es copien.

```
Francesc
@Override
public void execute(CreateSquadCommand command) {

    List<Keyword> keywords = findKeywords(command.keywordIds());

    Squad squad = new Squad(
            UUID.randomUUID().toString(),
            command.name(),
            command.maxMembers(),
            command.battleRole(),
            getKeywordPoints(keywords),
            getKeywordPoints(keywords),
            keywords,
            findFaction(command.factionId()),
            findSubFaction(command.subFactionId()),
            command.rosterId(),
            command.image());
    squadRepository.save(squad);
}
```

Fig.7.2.2. Creació d'una Squad pel Administrador

L'usuari convencional pot crear, modificar, eliminar i emmagatzemar el seu perfil i les seves llistes d'exercit (amb les seves Squads i unitats). Per fer-ho, el Backend fa duplicats (Fig.7.2.3) de les Squads i unitats creades pel administrador i se les assigna al usuari i un cop esta assignat al usuari, sempre es treballa amb aquelles entitats.

Fig.7.2.3. Clonació d'una Squad i assignació a la Roster d'un usuari.

7.2.1 – Arquitectura del software

Per al desenvolupament del Backend de la aplicació, s'ha emprat l'arquitectura hexagonal per tal de separar les responsabilitats amb l'objectiu de facilitar la gestió i manteniment del codi. El projecte s'ha estructurat seguint la següent distribució:

- Application: Es la capa que s'encarrega de la gestió dels casos d'us.
    - UseCase/UseCaseHandler: Els casos d'ús. Es segueix el patró Command/Query (Query quan es vol obtenir i Command quan es vol modificar) per rebre la instrucció que ha d'executar el UseCaseHandler.
- Config: Es la capa que conté la configuració i autenticació de l'aplicació.
    - Security: Conté la configuració per xifrar les connexions dels usuaris amb JWT.
- Domain: Es la capa central de l'aplicació on es defineixen la lògica de negoci i les regles de domini.
    - Exceptions: Es defineixen les excepcions adients.
    - Domain Objects: Es defineixen les entitats i objectes de domini.
- Infrastructure: Aquesta capa proporciona els mitjans per que l'aplicació es comuniqui amb l'exterior.

- Controllers: Es defineixen els controladors de l'aplicació. S'encarreguen de processar les sol·licituds dels usuaris i enviar les respostes adients.
    - ViewModels: Objectes que filtren el contingut dels objectes que es mostren al usuari.
- Persistence: Es la capa que permet a la aplicació comunicar-se amb la base de dades.
    - Entities: Son les entitats que representen les taules de la base de dades.
    - Mappers: Son els objectes que s'encarreguen de convertir els objectes de domini en entitats de persistència, i viceversa.
    - Repositories: Son els objectes que s'encarreguen de la lectura i escriptura de la base de dades.

7.2.2 – Infraestructura

A la carpeta *Infraestructura* s'han creat tots els elements relacionats amb els INPUTS i OUTPUTS del projecte. Aquets inclouen:

- Entitats JPA: Son els objectes principals de la aplicació i cada classe representa una taula que JPA s'encarregarà de crear, i cada objecte representa un registre de base de dades on JPA s'encarregarà de inserir en la taula corresponen.
- Repositoris JPA: Son les classes que permeten fer consultes a la base de dades. JPA traduirà les comandes i consultes al llenguatge de base de dades que es requereixi, en aquest cas MySQL.
- Repositori MySQL: Son les classes que mitjançant funcions permeten a la aplicació executar les consultes als repositori JPA. S'anomena MySQL per facilitar la comprensió de les classes emprades al projecte.
- Mapejadors: Son les classes que tradueixen els objectes obtinguts del Repositori JPA a objectes de domini.
- Controladors: Son les classes que permeten fer crides a l'aplicació mitjançant Endpoints de REST.

7.2.2.1 – Entitats JPA

Les entitats JPA son classes a les que s'ha especificat, mitjançant anotacions, les diferents característiques de les taules com son el tipus i nom dels atributs i les relacions (Fig.7.2.2.1.1). Aquestes anotacions son:

- @Entity: S'utilitza per marcar una classe com una entitat persistent.
- @Table: S'utilitza juntament amb l'anotació @Entity per especificar el nom de la taula a la qual està mapejada l'entitat a la base de dades.
- @Id: S'utilitza per marcar una propietat d'una entitat com a clau primària de la taula. Cada entitat ha de tenir una clau primària per identificar de manera única cada fila a la taula.
- @Column: S'utilitza per mapejar una propietat d'una entitat a una columna específica de la taula. Aquesta anotació permet especificar el nom de la columna, el seu tipus de dades, restriccions de longitud, si és nullable, etc.
- @OneToMany: S'utilitza per establir una relació d'un a molts entre dues entitats.
- @ManyToMany: S'utilitza per establir una relació de molts a molts entre dues entitats.
- @ManyToOne: S'utilitza per establir una relació de molts a un entre dues entitats
- @JoinColumn: S'utilitza per especificar el nom de la columna de junta en una relació entre entitats.

Les classes User, Roster, Squad, Unit, Weapon i Keyword estan relacionades de tal forma que cada una depèn de l'anterior, i si una instancia de major jerarquia es eliminada, les que depenen d'ella s'eliminen en cascada,

```java
@Entity
@Table(name = "factions")
public class FactionJpaEntity {
    @Id
    private String id;
    4 usages
    @Column(unique = true, nullable = false)
    private String name;
    4 usages
    @Column(nullable = false)
    private String description;
    4 usages
    @OneToMany(mappedBy = "faction", cascade=CascadeType.REMOVE, orphanRemoval=true)
    private Set<SubFactionJpaEntity> subFactions;
    no usages
    @OneToMany(mappedBy = "faction", cascade = CascadeType.REMOVE, orphanRemoval=true)
    private Set<SquadJpaEntity> squads;
    no usages
    @OneToMany(mappedBy = "faction", cascade = CascadeType.REMOVE, orphanRemoval=true)
    private Set<RosterJpaEntity> rosters;
    4 usages
    @Column(nullable = false)
    private String image;
```

Fig.7.2.2.1.1. FactionJpaEntity

7.2.2.2 – Repositori JPA

Els repositoris JPA (Fig.7.2.2.2.1) son interfícies que defineixen mètodes per accedir i manipular les dades d'una entitat a la base de dades. Utilitza les capacitats de JPA per realitzar operacions de persistència, com crear, llegir, actualitzar i eliminar registres a la base de dades. Aquests repositoris proporcionen una abstracció sobre la capa d'accés a dades i permeten interactuar amb les entitats de manera senzilla. S'ha fet servir "Native Query" per realitzar consultes natives principalment per un motiu de millora en eficiència, ja que es coneix el tipus de base de dades i la sintaxis, i així no cal traduir les crides. S'ha especificat a cada repositori el tipus de clau primària i la entitat JPA, i s'han emprat les següents anotacions:

- @Repository: S'utilitza en la classe per indicar que és un repositori.
- @Query: S'utilitza en els mètodes dels repositori per indicar una consulta personalitzada. Permet definir una consulta en llenguatge específic (SQL) per recuperar les dades desitjades de la base de dades. Aquesta anotació permet una

flexibilitat més gran per realitzar consultes complexes que no es poden expressar fàcilment amb els mètodes de consulta estàndard proporcionats per JPA.

- @Param: S'utilitza en un mètode amb l'anotació @Query per especificar els paràmetres de la consulta. S'utilitza per enllaçar els paràmetres de la consulta amb els paràmetres del mètode de Java. Això permet passar valors dinàmics als paràmetres de la consulta i realitzar consultes més flexibles i personalitzades.

```
3 usages  Francesc
@Repository
public interface FactionJpaRepository extends JpaRepository<FactionJpaEntity, String> {
    Francesc
    @Query(value =
            " SELECT " +
                    "    CASE WHEN EXISTS " +
                    "(SELECT * FROM factions f WHERE f.name = :name)" +
                    "    THEN 'TRUE'" +
                    "    ELSE 'FALSE'" +
                    "END", nativeQuery = true)
    boolean existsByName(@Param("name") String name);
    1 usage  Francesc
    @Query(value ="SELECT * from factions WHERE name LIKE :factionName ", nativeQuery = true)
    FactionJpaEntity findByName(@Param("factionName") String factionName);
}
```

Fig.7.2.2.2.1. FactionJpaRepository

7.2.2.3 – Repositori MySql

Aquesta classe (Fig.7.2.2.3.1) implementa les funcions del repositori de domini. Fa les crides al repositori JPA i s'encarrega de cridar als mapejadors per traduir els objectes bidireccionalment.

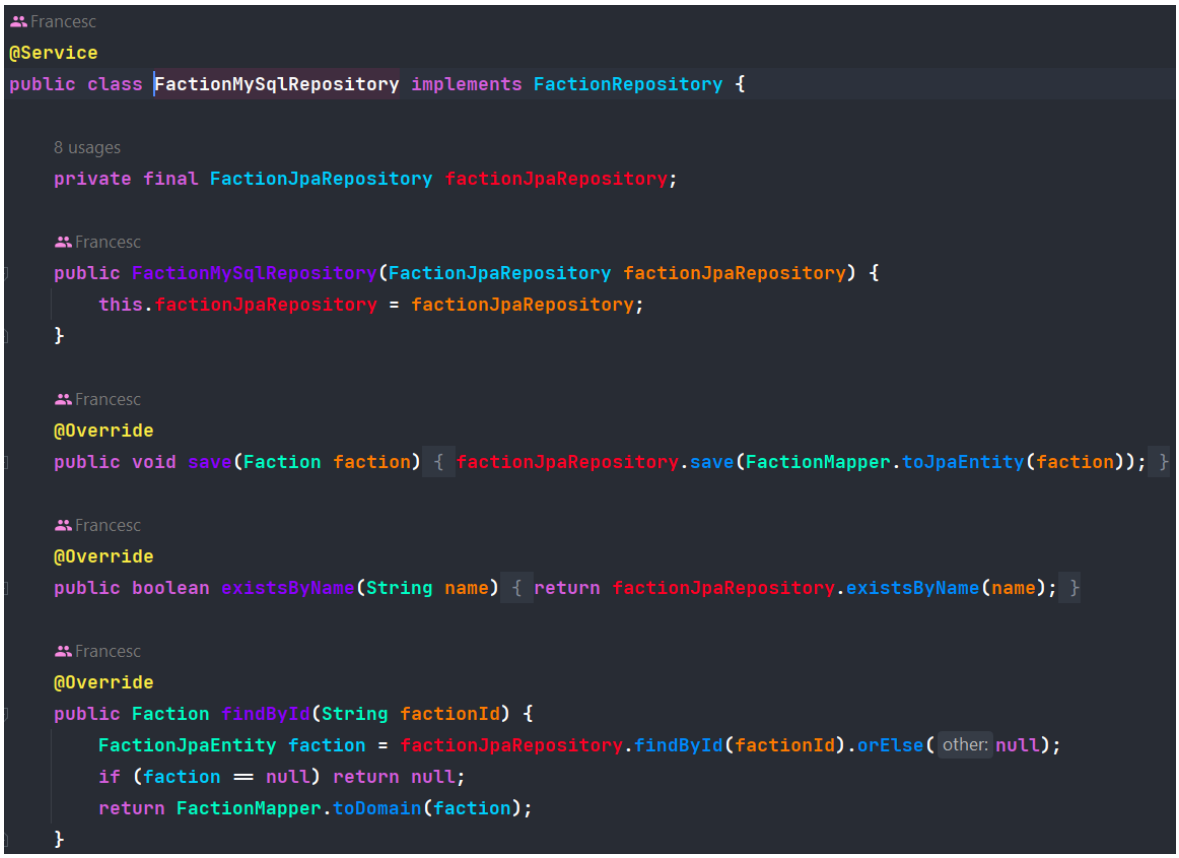

Fig.7.2.2.3.1. FactionMySqlRepository

7.2.2.4 – Mapejadors

S'utilitzen mapejadors (Fig.7.2.2.4.1) de entitats per convertir objectes de Java en estructures de dades que puguin ser emmagatzemades a la base de dades i viceversa.

```
11 usages  Francesc *
public class FactionMapper {
    Francesc
    public static Faction toDomain(FactionJpaEntity factionJpaEntity) {
        if (factionJpaEntity == null) return null;
        return new Faction(
                factionJpaEntity.getId(),
                factionJpaEntity.getName(),
                factionJpaEntity.getDescription(),
                SubFactionMapper.toDomain(factionJpaEntity.getSubFactions()),
                factionJpaEntity.getImage()
        );
    }

    Francesc
    public static FactionJpaEntity toJpaEntity(Faction faction) {
        if (faction == null) return null;
        return new FactionJpaEntity(
                faction.getId(),
                faction.getName(),
                faction.getDescription(),
                SubFactionMapper.toJpaEntity(faction.getSubFactions()),
                faction.getImage());
    }
```

Fig.7.2.2.4.1. FactionMapper

7.2.2.5 – Controladors

Els controladors de l'aplicació (Fig.7.2.2.4.1) són els components que gestionen les sol·licituds HTTP i actuen com a punt d'entrada per a les peticions dels clients. En el context d'aquesta aplicació REST, els controladors reben les sol·licituds, les executen utilitzant els casos d'us i retornen les respostes adequades en el format desitjat. En els controladors s'especifica el tipus de Endpoint, com es rep la petició i els objectes o paràmetres que te, i la resposta que retornarà, amb el següent:

- @RestController: S’utilitza al controllador per indicar que és un controlador REST. Un controlador REST és responsable de gestionar les sol·licituds HTTP i generar les respostes adequades en un servei web basat en l'arquitectura REST.
- @PostMapping("/api/weapons/create"): S’utilitzen anotacions en funció de les peticions i la direcció del Endpoint es personalitzada per cridar a cada classe. Les peticions poden ser:
    - POST: Crea un o mes objectes d’una de les entitats.
    - GET: Demana un o mes objectes d’una de les entitats.
    - PUT: Modifica un o mes objectes d’una de les entitats.
    - DELETE: Elimina un o mes objectes d’una de les entitats.
- @RequestBody: S'utilitza com a paràmetre del mètode d'un controlador per indicar que el cos de la sol·licitud HTTP es un objecte, i es mappeja a la classe indicada (en aquest cas una Command o una query).
- @RequestParam: S'utilitza com a paràmetre del mètode d'un controlador per indicar que el valor ha de ser extret dels paràmetres de la sol·licitud HTTP amb un format semblant a aquest exemple: “/delete?user-name=xxxx”.
- @PathVariable: S'utilitza com a paràmetre del mètode d'un controlador per indicar que el valor ha de ser extret de part de la URL de la sol·licitud HTTP que estigui envoltat amb “{}”. Si hi ha una ruta "/api/users/{id}", l'anotació @PathVariable s'utilitza per capturar el valor de "id" de la URL i assignar-lo al paràmetre anotat.
- ViewModel: Quan alguna crida requereix objectes de domini, s’utilitzen seguint per encapsular les dades del objecte de domini i així poder retornar lo estrictament necessari sense comprometre tot l’objecte (seguint la idea de l'arquitectura Model-View-Controller (MVC))
- ResponseEntity: Quan alguna crida no requereix objectes, es retorna una resposta HTTP personalitzada en funció de la crida, aquestes respostes poden ser:
    - Status HTTP 201: L’estatus HTTP 201 CREATED indica que s’ha creat correctament el recurs.
    - Status HTTP 204: L’estatus HTTP 204 NO_CONTENT indica que s’ha eliminat correctament el recurs.
    - Status HTTP 200: L’estatus HTTP 200 OK indica que la petició s’ha executat correctament (es l’estatus per defecte).

Fig.7.2.2.4.1. CreateFactionController

7.2.3 – Aplicació

Aquesta capa és responsable de gestionar la comunicació amb els actors externs (controladors) seguint el patró Command/Query, l'aplicació rep les sol·licituds dels controladors, processa les dades i coordina la interacció entre els casos d'ús i el domini.

7.2.3.1 – Command/Query

S'ha emprat el patró Command/Query (Fig.7.2.3.1.1) per separar les operacions de lectura (queries) i les operacions de modificació (commands) de l'aplicació.

- Query: Una query és una operació que consulta i recupera dades de l'aplicació, però no modifica l'estat o les dades del sistema. Les queries no tenen efectes secundaris en el sistema.
- Command: Un command és una operació que realitza una modificació o una acció en el sistema. Aquest tipus d'operacions poden incloure la creació, l'actualització o l'eliminació de dades, així com altres accions que canvien l'estat del sistema. Els commands són responsables de modificar i actualitzar les dades del sistema.

La principal idea darrere del patró Command/Query és evitar que les operacions de lectura modifiquin l'estat del sistema i que les operacions de modificació realitzin lectures

innecessàries. Aquesta separació millora la claredat i la cohesió del codi, ja que cada part de l'aplicació té una responsabilitat clara.

A cada Command/Query es valida el format de les dades i es llença una excepció personalitzada en el cas de que el format o valor sigui erroni.

```
4 usages  Francesc
public record CreateFactionCommand(String name, String description, String image) {

    no usages  Francesc
    public CreateFactionCommand {
        if (name == null || name.isBlank())
            throw new InvalidCommandQueryParametersException("CreateFactionCommand: Name can't be null or blank.");
        if (description == null || description.isBlank())
            throw new InvalidCommandQueryParametersException("CreateFactionCommand: Description can't be null or blank.");
        if (image == null || image.isBlank())
            throw new InvalidCommandQueryParametersException("CreateFactionCommand: Image can't be null or blank.");
    }

    Francesc
    @Override
    public String toString() {
        return "CreateFactionCommand{" +
                "name='" + name + '\'' +
                ", description='" + description + '\'' +
                ", image='" + image + '\'' +
                '}';
    }
}
```

Fig.7.2.3.1.1. CreateFactionCommand

7.2.3.2 – Casos d'ús

Els casos d'ús (Fig.7.2.3.2.1) representen els diversos escenaris o funcionalitats que el sistema ha de proporcionar als seus usuaris. Cada cas d'ús defineix una tasca o objectiu específic que es pot dur a terme en el sistema. En aquest context, els casos d'ús defineixen la lògica de negoci i interactuen amb el domini per satisfer els requisits de l'usuari.

En els casos d'us es realitzen validacions dels recursos, es llencen excepcions en cas de no superar la validació i s'executa la lògica del cas d'ús.

```java
Francesc
@Service
public class CreateFactionUseCaseHandler implements CreateFactionUseCase {
    3 usages
    private final FactionRepository factionRepository;

    Francesc
    public CreateFactionUseCaseHandler(FactionRepository factionRepository) {
        this.factionRepository = factionRepository;
    }

    Francesc
    @Override
    public void execute(CreateFactionCommand command) {

        if (factionRepository.existsByName(command.name()))
            throw new ResourceAlreadyExistsException("Faction [" + command.name() + "] already exists.");

        Faction faction = new Faction(
                UUID.randomUUID().toString(),
                command.name(),
                command.description(),
                command.image());
        factionRepository.save(faction);
    }
}
```

Fig.7.2.3.2.1. CreateFactionUseCaseHandler

7.2.4 – Domini

El domini és el nucli de l'arquitectura hexagonal. Aquesta capa conté les regles de negoci, les entitats i la lògica que defineixen el comportament essencial del sistema. És la part més independent de l'arquitectura i no depèn de les implementacions o detalls tècnics. En aquesta capa, s'implementen les entitats, així com altres components com els repositoris, models i excepcions.

7.2.4.1 – Entitats

Les entitats (Fig.7.2.4.1.1) són els objectes que representen les entitats JPA al domini. Aquestes entitats encapsulen les dades i el comportament relacionat amb el desenvolupament de l'aplicació. Les entitats són una representació de la informació del domini i contenen els atributs i mètodes necessaris per manipular aquestes dades.

```java
Francesc
public class Faction {
    5 usages
    private String id;
    5 usages
    private String name;
    5 usages
    private String description;
    4 usages
    private List<SubFaction> subFactions;
    5 usages
    private String image;

    Francesc
    public Faction(String id, String name, String description, List<SubFaction> subFactions, String image) {
        this.id = id;
        this.name = name;
        this.description = description;
        this.subFactions = subFactions;
        this.image = image;
    }
```

Fig.7.2.4.1.1. Faction

7.2.4.2 – Repositoris

Els repositoris (Fig.7.2.4.2.1) son les interfícies que declaren el tipus d'acció que requereix cada entitat i que son implementats pels repositoris MySql.

```java
1 implementation  Francesc
public interface FactionRepository {
    1 implementation  Francesc
    void save(Faction faction);

    1 implementation  Francesc
    boolean existsByName(String name);

    1 implementation  Francesc
    Faction findById(String factionId);

    1 usage  1 implementation  Francesc
    boolean factionExistsById(String factionId);

    1 implementation  Francesc
    void delete(Faction faction);

    1 usage  1 implementation  Francesc
    List<Faction> findAll();

    1 usage  1 implementation  Francesc
    Faction findByName(String factionName);
}
```

Fig.7.2.4.2.1. FactionRepository

7.2.4.3 – Models

Els enum (Fig.7.2.4.3.1) representen valors fixos o constants relacionats amb el domini. Aquestes enumeracions s'utilitzen per representar opcions predefinides o valors permesos en el model del domini. S'han emprat enums per als BattleRole, FightSkill, KeywordType, UpdateOperation, UserRole i WeaponType.

```
Francesc
public enum BattleRole {
    no usages
    HQ,
    no usages
    TROOPS,
    no usages
    ELITE,
    no usages
    FAST_ATTACK,
    no usages
    HEAVY_SUPPORT,
    no usages
    FLYER,
    no usages
    DEDICATED_TRANSPORT,
    no usages
    LORD_OF_WAR,
    no usages
    FORTIFICATION,
    no usages
    SUPREME_COMMANDER
}
```

Fig.7.2.4.3.1. Enum BattleRole

7.2.4.4 – Excepcions

Es fan servir excepcions personalitzades (Fig.7.2.4.4.1) en funció del error aparegut durant l'execució de l'aplicació. Aquestes excepcions s'han creat al domini i quan hi ha un error en temps d'execució, aquestes son recollides pel GlobalExceptionHandler (Fig.7.2.4.4.2). Aquest servei retorna un Status relacionat amb l'excepció per notificar correctament l'origen i motiu de la excepció i que el front End pugui mostrar-ho al usuari.

Fig.7.2.4.4.1. InvalidCommandQueryParametersException

Fig.7.2.4.4.2. GlobalExceptionHandler

7.2.5 – Configuració

S'ha aplicat una lògica de seguretat per al Log In de l'aplicació (Fig.7.2.5.1). Aquest lògica consisteix en crear un Jason Web Token amb el username i la contrasenya del usuari i, en funció d'uns paràmetres (Fig.7.2.5.2), permetre nomes peticions que incloguin aquest Bearer Token, si no s'inclou el JWT es llença una excepció prohibint l'accés a aquell recurs.

```
Francesc *
@Bean
AuthenticationManager usersAuthenticationManager() {
    return authentication → {
        UserDetails userPrincipal = customUserDetailsService.loadUserByUsername(authentication.getName());
        if (!passwordEncoder().matches(authentication.getCredentials().toString(), userPrincipal.getPassword()))
            throw new BadCredentialsException("Invalid password");
        return new UsernamePasswordAuthenticationToken(
                userPrincipal,
                authentication.getCredentials(),
                userPrincipal.getAuthorities());
    };
}
```

Fig.7.2.5.1. Authentication

```
Francesc
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
            .cors().and().csrf().disable() HttpSecurity
            .exceptionHandling().authenticationEntryPoint(unauthorizedHandler) ExceptionHandlingConfigurer<HttpSecurity>
            .and() HttpSecurity
            .sessionManagement() SessionManagementConfigurer<HttpSecurity>
            .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and() HttpSecurity
            .logout() LogoutConfigurer<HttpSecurity>
            .logoutUrl("/api")
            .and() HttpSecurity
            .authorizeHttpRequests() AuthorizationManagerRequestMat...
            .requestMatchers("/", "index", "/css/*", "/js/*").permitAll()
            .requestMatchers("/v2/**", "/webjars/**").permitAll()
            .requestMatchers("/api/auth/**").permitAll()
            .requestMatchers("/api/users/create").permitAll()
            .anyRequest().authenticated();

    http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
    return http.build();
}
```

Fig.7.2.5.2. Filters

7.2.6 – Implementació dels requeriments funcionals

Seguint la planificació s'han implementat els requeriments funcionals establerts.

S'han creat els diferents Controllers a la infraestructura de la aplicació que permeten crear, modificar i eliminar els elements principals (USER, WEAPON, KEYWORD, ROSTER, UNIT, SQUAD, FACTION, SUB-FACTION) així com assignar els diferents elements a les ROSTER i poder fer LOG IN amb l'usuari.

Per implementar cada funcionalitat, s'ha seguit el següent ordre d'actuació (es fa servir la funcionalitat "*Example*" com a exemple):

Es crea el Controller (Fig.7.2.6.1.1.) amb un Endpoint.

Es crea un ViewModel (Fig.7.2.6.1.2.) o una ResponseEntity.

Es crea la Command o la Query (Fig.7.2.6.1.3.).

Es crea el UseCase (Fig.7.2.6.1.4. i Fig.7.2.6.1.5.).

Es crea el Domain Object (Fig.7.2.6.1.6.).

Es crea el Repository de Domain (Fig.7.2.6.1.7.).

Es crea el Repository MySQL (Fig.7.2.6.1.8.).

Es crea el Repository JPA (Fig.7.2.6.1.9.).

Es crea la JpaEntity (Fig.7.2.6.1.10.).

Es crea el Mapper (Fig.7.2.6.1.11.).

```
package FBOIX.TFG.infrastructure.controllers;
import ...
Francesc *
@RestController
public class ExampleController {
    2 usages
    private final ExampleUseCase exampleUseCase;
    new *
    public ExampleController(ExampleUseCase exampleUseCase) { this.exampleUseCase = exampleUseCase; }
    new *
    @GetMapping("api/test")
    public ExampleViewModel example (@RequestParam(value = "example-value") String exampleValue){

        return ExampleViewModel.ToViewModel(exampleUseCase.execute(new ExampleQuery(exampleValue)));
    }
}
```

Fig. 7.2.6.1.1. Controller.

```java
package FBOIX.TFG.infrastructure.controllers.viewModel;
import FBOIX.TFG.domain.Example;
5 usages   new *
public class ExampleViewModel {
    3 usages
    private final String resultQuery;
    1 usage   new *
    public ExampleViewModel(String resultQuery) { this.resultQuery = resultQuery; }
    1 usage   new *
    public static ExampleViewModel ToViewModel(Example example) {
        return new ExampleViewModel(example.getExampleValue());
    }
    no usages   new *
    public String getResultQuery() { return resultQuery; }
    new *
    @Override
    public String toString() {
        return "ExampleViewModel{" +
                "resultQuery='" + resultQuery + '\'' +
                '}';
    }
}
```

Fig. 7.2.6.1.2. ViewModel.

```java
package FBOIX.TFG.application.Example;
4 usages   new *
public class ExampleQuery {
    3 usages
    private String exampleValue;
    1 usage   new *
    public ExampleQuery(String exampleValue) {
        this.exampleValue = exampleValue;
    }
    1 usage   new *
    public String getExampleValue() {
        return exampleValue;
    }
    no usages   new *
    public void setExampleValue(String exampleValue) {
        this.exampleValue = exampleValue;
    }
}
```

Fig. 7.2.6.1.3. Query.

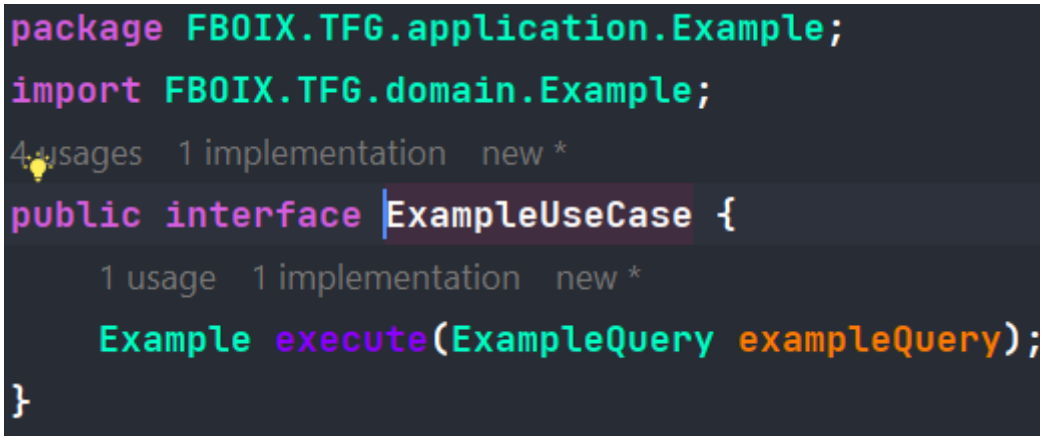

Fig. 7.2.6.1.4. UseCase.

Fig. 7.2.6.1.5. UseCaseHandler.

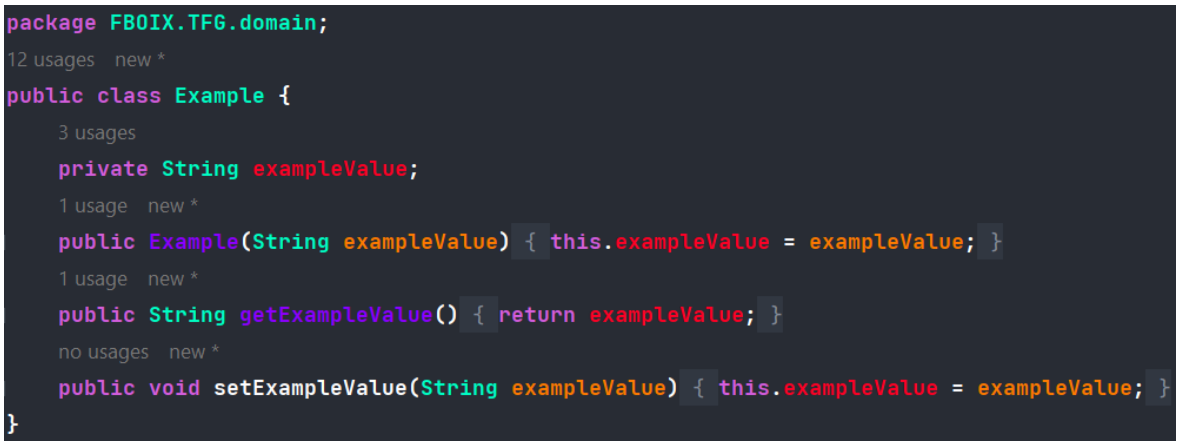

Fig. 7.2.6.1.6. Domain Object.

```java
package FBOIX.TFG.domain;

5 usages   1 implementation   new *
public interface ExampleRepository {

    1 usage   1 implementation   new *
    Example exampleJob(String exampleValue);
}
```

Fig. 7.2.6.1.7. Domain Repository.

```java
package FBOIX.TFG.infrastructure.persistence.repositories;

import FBOIX.TFG.domain.Example;
import FBOIX.TFG.domain.ExampleRepository;
import FBOIX.TFG.infrastructure.persistence.mappers.ExampleMapper;
import org.springframework.stereotype.Service;

new *
@Service
public class ExampleMySqlRepository implements ExampleRepository {

    2 usages
    private final ExampleJpaRepository exampleJpaRepository;

    new *
    public ExampleMySqlRepository(ExampleJpaRepository exampleJpaRepository) {
        this.exampleJpaRepository = exampleJpaRepository;
    }

    1 usage   new *
    @Override
    public Example exampleJob(String exampleValue) {
        return ExampleMapper.toDomain(exampleJpaRepository.exampleJpaJob(exampleValue));
    }
}
```

Fig. 7.2.6.1.8. MySQL Repository.

```java
package FBOIX.TFG.infrastructure.persistence.repositories;
import FBOIX.TFG.infrastructure.persistence.entities.ExampleJpaEntity;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.CrudRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;
2 usages  new *
@Repository
public interface ExampleJpaRepository extends CrudRepository<ExampleJpaEntity, String> {
    1 usage  new *
    @Query(value = "SELECT * FROM example WHERE id = :exampleValue", nativeQuery = true)
    ExampleJpaEntity exampleJpaJob(@Param("exampleValue")String exampleValue);
}
```

Fig. 7.2.6.1.9. JPA Repository.

```java
package FBOIX.TFG.infrastructure.persistence.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import org.hibernate.annotations.DynamicUpdate;
7 usages  new *
@Entity
@Table(name = "example")
@DynamicUpdate
public class ExampleJpaEntity {
    @Id
    private String id;
    1 usage  new *
    public ExampleJpaEntity(String id) { this.id = id; }
    new *
    public ExampleJpaEntity() {    }
    new *
    public String getId() { return id; }
    new *
    public void setId(String id) { this.id = id; }
}
```

Fig. 7.2.6.1.10. JPA Entity.

```java
package FBOIX.TFG.infrastructure.persistence.mappers;

import FBOIX.TFG.domain.Example;
import FBOIX.TFG.infrastructure.persistence.entities.ExampleJpaEntity;
2 usages   new *
public class ExampleMapper {
    1 usage   new *
    public static Example toDomain(ExampleJpaEntity exampleJpaEntity) {
        return new Example(exampleJpaEntity.getId());
    }
}
```

Fig. 7.2.6.1.11. Mapper

7.3 – Frontend

El Frontend es la part client que interactua amb l'usuari i li permet fer servir les funcionalitats de la plataforma. El Frontend de l'aplicació s'ha desenvolupat amb React Native (Fig.7.3.1) i consisteix en una aplicació mòbil on els usuaris poden crear, editar i eliminar les diferents llistes d'exercit i les seves esquadres i unitats, així com visualitzar tot el conjunt de entitats (armes, paraules clau, perfils d'unitats etc).

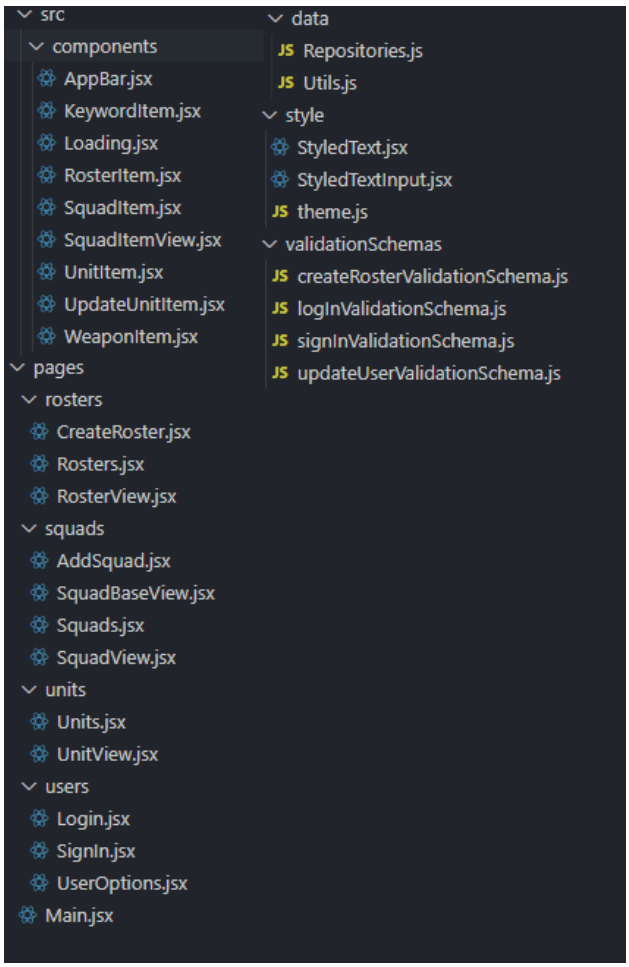

Fig.7.3.1. Organitzacio del FrontEnd.

Per visualitzar l'aplicació aneu als annexes.

7.3.1 – Data

Per fer les crides al Backend es fa servir funcions asíncrones (Fig7.3.1.1) per realitzar crides HTTP al Endpoint corresponent. En aquestes peticions, s'ha d'incorporar el JWT que s'aconsegueix cada cop que un usuari fa Log In. Juntament amb el token, s'incorporen a la

url els paràmetres pertinents per a fer correctament la consulta, ja sigui un body JSON o variables.

```
import { getHeaders } from './Utils.js';

const API_BASE_URL = 'http://192.168.0.3:8080/api';


export async function login(loginRequest) {
    const url = API_BASE_URL + "/auth/logIn";
    const headers = await getHeaders('POST');

    try {
        const response = await fetch(url, Object.assign({}, headers, { body: JSON.stringify(loginRequest) }));
```

Fig7.3.1.1. Crida HTTP de la funció LogIn

Un cop es realitza la petició HTTP, el server respon amb la resposta personalitzada del endopoint. A cada funció, es comprova si la resposta conté el status HTTP correcte i si no es així, llença un error personalitzat (Fig.7.3.1.2).

```
try {
    const response = await fetch(url, Object.assign({}, headers, { body: JSON.stringify(loginRequest) }));
    if (response.status !== 200) {
        if (response.status === 401)
            throw new Error("The user or password are not correct, try again please.");
        else {
            if (response.status === 404)
                throw new Error("The user doesn't exist, try again please.");
            throw new Error("Internal server error.");
        }
    }
    return await response.json();
} catch (error) {
    throw new Error(error.message);
}
```

Fig.7.3.1.2. Excepcions personalitzades de la funció LogIn.

Per guardar en la sessió tant el usuari com el seu token, i per afegir aquest token a cada petició (crear els headers) es fan servir una sèrie de funcions especifiques (Fig.7.3.1.3). Per guardar l'usuari i el token es fa servir la llibreria SecureStorage.

```
import * as SecureStore from 'expo-secure-store';

async function getValueFor(key) {
    let result = await SecureStore.getItemAsync(key);
    if (result) {
        return result
    } else {
        return ''
    }
}
export async function getHeaders(method) {
    const headers = new Headers({ 'Content-Type': 'application/json', })
    let token = await getValueFor('access_token')
    if (token !== '') {
        headers.append('Authorization', "Bearer " + token)
    }
    return Object.assign({}, { headers: headers }, { method });
}
export async function userIsAuthenticated() {
    let token = await getValueFor('access_token')
    if (token !== '') {
        return true;
    } else return false
}
export async function getCurrentUserStored() {
    let username = await getValueFor('user_name')
    let role = await getValueFor('user_role')
    return { "userName": username, "userRole": role }
}
```

Fig.7.3.1.3. Utils.js

7.3.2 – Pantalles

El projecte esta dividit en pantalles i components. Les pantalles son totes aquelles classes que son una pantalla principal de l'aplicació i estan dividides en categories en funció de cada entitat.

El main de l'aplicació (Fig.7.3.2.1) incorpora totes les rutes i l'element que renderitza.

```jsx
const Main = () => {
    return (
        <View style={{ flex: 1, backgroundColor: '#FFFFFF' }}>
            <AppBar />
            <Routes>
                <Route path='/' element={<LogInPage />} />
                <Route path='/logIn' element={<LogInPage />} />
                <Route path='/signIn' element={<SignIn />} />
                <Route path='/userOptions' element={<UserOptions />} />

                <Route path='/rosters' element={<Rosters />} />
                <Route path='/createRoster' element={<CreateRoster />} />
                <Route path='/rosterView' element={<RosterView />} />

                <Route path='/squads' element={<Squads />} />
                <Route path='/addSquad' element={<AddSquad />} />
                <Route path='/squadBaseView' element={<SquadBaseView />} />
                <Route path='/squadView' element={<SquadView />} />

                <Route path='/units' element={<Units />} />
                <Route path='/unitView' element={<UnitView />} />

            </Routes>
        </View>
    )
}

export default Main
```

Fig.7.3.2.1. Main.jsx

7.3.2.1 – Login

La pantalla Log In permet al usuari accedir amb les seves credencials a la aplicació.

Es un formulari validat mitjançant la llibreria Formik (Fig. 7.3.2.1.1) on, un cop introduïdes les credencials i el format ha sigut validat, es fa una petició al server on es comprovarà si existeix l'usuari i si la contrasenya es correcte.

```
<Formik
    validationSchema={logInValidationSchema}
    initialValues={initialValues}
    onSubmit={values => logIn(values)}
    innerRef={formRef}>
    {({ }) => {
        return (
            <View style={styles.form}>
                <StyledText fontSize="subheading" bold>Username</StyledText>
                <FormikInputValue
                    name='username'
                    placeholder='Username'
                />
                <StyledText fontSize="subheading" bold>Password</StyledText>
                <FormikInputValue
                    name='password'
                    placeholder='Password'
                    secureTextEntry
                />
            </View>
        )
    }}
</Formik>
```

Fig. 7.3.2.1.1. Formulari Formik

Amb una resposta positiva, el server retorna un JWT que es guarda en el SecureStorage. Un cop es té el JWT, es fa una altre petició al server per demanar el nom d'usuari y el seu rol. Aquets valors es fan servir en altres pantalles i es guarden també en el SecureStorage (Fig. 7.3.2.1.2). Quan tot esta guardat, es fa us del useNavigate per anar a la pantalla de Rosters.

```
const logIn = (values) => {
    login(values).then(response => {
        saveSecureItem('access_token', response.accessToken).then(() => {
            getCurrentUser().then(response => {
                saveSecureItem('user_name', response.username)
                saveSecureItem('user_role', response.authorities[0].authority)
                navigate("/rosters")
            }).catch(error => { alert(error); });
        })
    }).catch(error => { alert(error); });
}
```

Fig. 7.3.2.1.2. Log In i SecureStorage

Si un usuari entra a la aplicació, es fa servir el hook useEffect per executar una funció (Fig. 7.3.2.1.3) que comprova si el usuari ja esta autenticat. En cas positiu es fa us del useNavigate per anar a la pantalla de Rosters.

Fig. 7.3.2.1.3. UseEffect

7.3.2.2 – Sign In

La pantalla Sign In permet al client de l'aplicació crear un nou usuari per accedir a la plataforma.

Es un formulari validat mitjançant la llibreria Formik on, un cop introduïdes les credencials i el format ha sigut validat, es fa una petició al server on es comprova si ja existeix un usuari amb el mateix username, si existeix es mostra un missatge d'error, i en cas contrari es crea l'usuari al server, es mostra un missatge de confirmació i amb l'useNavigate retorna a la pagina Log In.

7.3.2.3 – UserOptions

La pantalla UserOptions permet al usuari modificar les seves dades o eliminar el seu usuari definitivament.

Al renderitzar aquest component, es crida al hook useEffect per executar una funció que busca el les dades del usuari actual al server (mitjançant les dades del SecureStorage) per posar-les al formulari com a placeholder.

Es un formulari validat mitjançant la llibreria Formik on, un cop introduït mínim un canvi en les dades, es valida el format i es fa una petició al server on es realitzen els canvis. En el cas de que es que es vulgui canviar de username es comprova si ja existeix un usuari amb

el mateix username, en el cas de que si existeix es mostra un missatge d'error, i en cas contrari es modifiquen les dades de l'usuari al server. Es mostra un missatge de confirmació i amb l'useNavigate retorna a la pagina Rosters.

En el cas de que l'usuari vulgui eliminar el seu compte, es demana una validació amb una alerta. En cas afirmatiu, es fa una petició per eliminar l'usuari i es borren les dades emmagatzemades en el SecureStorage (Fig.7.3.2.3.1)

```
const deleteUser = () => {
    Alert.alert('Delete User', 'Are you sure you want to delete your user?', [
        {
            text: 'Cancel',
            style: 'cancel',
        },
        {
            text: 'Yes', onPress: () => deleteUserByUsername(originalName).then(() => {
                saveSecureItem('access_token', '')
                    .then(() => saveSecureItem('user_name', '')
                        .then(() => saveSecureItem('user_role', '').then(() => navigate("/"))))
            }).catch(error => { alert(error) })
        },
    ]);

}
```

Fig.7.3.2.3.1. Delete User

Tant si es modifiquen les dades, com si s'elimina l'usuari, es tanca la sessió i mitjançant l'useNavigate es retorna al Log In.

7.3.2.4 – Rosters

La pantalla rosters serveix per visualitzar totes les llistes d'exercit creades i actua com un menú principal.

Quan es renderitza aquesta pantalla, amb l'useEffect s'executa una funció que recupera les dades del usuari del SecureStorage i es fa una petició al server per recuperar totes les rosters que ha creat aquest usuari.

Aquestes rosters es guarden mitjançant el hook useState i es renderitzant mitjançant un component FlatList (que renderitza un component RosterItem per cada roster), que permet tenir totes les Rosters en una llista i poder fer scroll si hi han moltes.

Des de aquesta pantalla i mitjançant el hook useNavigate es pot:

- Anar al creador de noves llistes.
- Anar al editor del teu usuari.
- Fer Log Off de la aplicació i anar al Login.

Amb la renderització del Roster:

- Anar a la edició de les llistes actuals.
- Eliminar una llista (es fa un refresh d'aquesta pantalla)
- Anar al visualitzador detallat de una llista.

7.3.2.5 – CreateRoster

Aquesta pantalla permet al usuari crear una Roster amb un formulari validat mitjançant la llibreria Formik.

En el formulari es demana:

- Nom de la llista
- Restricció de punts
- Facció (modal)
- Subfacció (modal que depèn de la elecció de la Facció)

Quan es renderitza la pantalla, amb el hook useEffect s'executa una funció que demana al server totes les Faccions que es guarden mitjançant el hook useState i que es mostraran en un modal al prémer un boto.

Un cop seleccionada la facció es fa una crida al server buscant totes les subfaccions de la facció seleccionada, i es guardaran en un useState.

En el cas de que es faci un canvi en la facció quan ja s'ha seleccionat una subfacció, aquesta es borra i es demana escollir una nova subfacció de la facció.

Si s'intenta escollir una subfacció abans que una facció, es mostra un missatge d'error.

Quan es validen les dades de la Roster, s'envia una petició al server amb un body amb totes les dades. Si no hi ha cap problema, es guarda la roster a la base de dades i es torna a la pantalla de Rosters mitjançant el useNavigate.

7.3.2.6 – RosterView

Aquesta pantalla serveix per visualitzar amb detall tots els components de la Roster.

Quan es renderitza aquesta pantalla, es crida al useEffect per que executi una funció que demana al server la entitat Roster passant-li per paràmetre la ID de la roster que es vol visualitzar. Aquesta ID s'obté del hook useLocation (des de el RosterItem es passa al estat del location la rosterId que es necessita).

Un cop es te l'objecte, es mapeja els atributs per presentar-los per pantalla. Es fa una distinció entre els atributs de la pròpia Roster i la llista de Squads que te assignades. Els atributs de la Roster es guarda mitjançant el hook useState i es renderitzen directament en aquesta pantalla (la facció i la subfacció es renderitzen mitjançant una taula (Fig.7.3.2.6.1)), i la llista de Squads es passa a una FlatList que renderitzarà components SquadItemView per mostrar els atributs de les Squads.

```
<View style={{ marginRight: 10 }}>
    {faction && <ScrollView horizontal={true} style={styles.scrollView}>
        <Table borderStyle={styles.tableStatsBorder}>
            {faction.map((rowData, index) => (
                <Row
                    key={index}
                    data={rowData}
                    widthArr={[80, 180]}
                    style={styles.tableStatsSection}
                    textStyle={styles.tableStatsText}
                />
            ))}
        </Table>
    </ScrollView>}
    {subFaction && <ScrollView horizontal={true} style={styles.scrollView}>
        <Table borderStyle={styles.tableStatsBorder}>
            {subFaction.map((rowData, index) => (
                <Row
                    key={index}
                    data={rowData}
                    widthArr={[80, 180]}
                    style={styles.tableStatsSection}
                    textStyle={styles.tableStatsText}
                />
            ))}
        </Table>
    </ScrollView>}
</View>
```

Fig.7.3.2.6.1. Taula Facció i Subfacció

7.3.2.7 – Squads

A aquesta pantalla s'accedeix quan es vol editar una Roster (des de el RosterItem es passa al estat del location la rosterId i rosterName que es necessita). Mostra totes les Squads de la roster separades en categories (BattleRoles) i permet afegir, modificar, eliminar o visualitzar les squads.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al server una llista amb tots els rols de batalla (per crear les categories) i un altre funció que demana totes les squads associades a la Roster.

Els battleroles i les squads es renderitzen mitjançant una SectionList (Fig.7.3.2.7.1) on els battleroles son la capçalera (Fig.7.3.2.7.1) i les squads el contingut.

```
<SectionList
    sections={squads}
    keyExtractor={(item, index) => item + index}
    renderItem={({ item }) => (
        <View style={styles.container}>
            <View style={styles.squadList}>
                <View style={styles.imageContainer} >
                    <Image style={styles.image} source={{ uri: "https://wh40k.lexicanum.com/mediawiki/images/f/f7/Eldar.png" }} /
                </View>
                <View style={{ flex: 1, justifyContent: 'center' }}>
                    <StyledText style={styles.section} fontSize="subheading" bold underline color='secondary'>{item.name}</Styled
                    <StyledText style={styles.section}>{"Members: " + item.members + "/" + item.maxMembers}</StyledText>
                    <StyledText style={styles.section}>{"Points: " + item.points}</StyledText>
                </View>
                <View style={styles.buttonsContainer}>
                    <View>
                        <TouchableOpacity onPress={() => handleView(item.id)} style={styles.button}>
                            <StyledText fontSize="subheading" bold>
                                <Icon name="eye" style={styles.buttonSymbol} />
                            </StyledText>
                        </TouchableOpacity>
                        <TouchableOpacity onPress={() => handleEdit(item.id, item.name)} style={styles.button}>
                            <StyledText fontSize="subheading" bold>
                                <Icon name="pencil" style={styles.buttonSymbol} />
                            </StyledText>
                        </TouchableOpacity>
                        <TouchableOpacity onPress={() => handleDelete(item.id)} style={styles.button}>
                            <StyledText fontSize="subheading" bold>
                                <Icon name="trash" style={styles.buttonSymbol} />
                            </StyledText>
                        </TouchableOpacity>
                    </View>
                </View>
            </View>
        </View>
    )}
```

Fig.7.3.2.7.1. SectionList de Squads i Battleroles

```
renderSectionHeader={({ section: { title } }) => (
    <View style={styles.sections}>
        <StyledText fontSize="subheading" bold color='secondary' style={{ marginLeft: 10 }}>{title}</StyledText>
        <TouchableOpacity onPress={() => addSquad(title)} style={styles.barButton}>
            <StyledText fontSize="subheading" bold>
                <Icon name="plus" style={styles.buttonSymbol} />
            </StyledText>
        </TouchableOpacity>
    </View>
)}
```

Fig.7.3.2.7.1. Capçalera de la SectionList

7.3.2.8 – AddSquad

A aquesta pantalla s'accedeix si l'usuari vol afegir squads a una Roster (des de el Squads es passa al estat del location el battleRole, la rosterId que es necessita). Mostra totes les squads disponibles pel rol de batalla que ha escollit.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al server una llista amb totes les squads disponibles d'aquell rol de batalla. Aquestes Squads es mostren amb una FlatList (que renderitza un SquadItem per cada element) i l'usuari pot afegir-ne una o visualitzar els atributs basics de la Squad.

7.3.2.9 – SquadsBaseView

A aquesta pantalla es pot visualitzar els atributs basics d'una Squad.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al server els atributs basics de la squad en qüestió i les Keywords assignades (les renderitza en una flatlist, que renderitza KeywordItem per cada una).

7.3.2.10– SquadView

A aquesta pantalla es pot visualitzar tots els atributs d'una Squad, incloent les seves unitats i les armes i keywords d'aquestes unitats.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al sever l'objecte sencer de la squad en qüestió. Son els mateixos que els atributs base, però afegeix una llista de les ID de les seves unitats. Aquets Id's es passen a una Flatlist on mitjançant UnitItems, s'encarregarà de renderitzar cada unitat.

7.3.2.11 – Units

A aquesta pantalla s'accedeix quan es vol editar una Squad (desde la llista de Squads, es passa al estat del location la rosterId, rosterName, la squadId, i la squadName que es necessita). Mostra totes les Unitats disponibles de la squad i permet afegir o treure membres de cada unitat, i visualitzar cada Unit.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al server una llista amb totes les unitats disponibles d'aquella Squad i el numero de membres de cada unitat.

Cada una d'aquestes unitats es renderitza en una FlatList mitjançant el component UpdateUnitItem.

Un cop s'ha afegit o tret els membres desitjats de cada unit, es guarda les unitats fent una crida al server. Un cop guardada les unitats a la squad, es torna a la pantalla squads mitjançant el hook useNavigate.

7.3.2.12 – UnitView

A aquesta pantalla es pot visualitzar tots els atributs d'una Unit, incloent les seves armes i les keywords de les armes.

Quan es renderitza s'executa el useEffect cridant a una funció que demana al sever l'objecte sencer de la Unit en qüestió. Es mostren els atributs de les unitats mitjançant taules (Fig.7.3.2.12.1) i les llistes d'armes es passen a una una Flatlist on mitjançant el components WeaponItem, s'encarregarà de renderitzar cada Weapon.

```
{stats && <ScrollView horizontal={true} style={{ margin: 5 }}>
    <Table borderStyle={styles.tableStatsBorder}>
        {stats.map((rowData, index) => (
            <Row
                key={index}
                data={rowData}
                widthArr={[30, 30, 30, 30, 30, 30, 30]}
                style={styles.tableStatsSection}
                textStyle={styles.tableStatsText}
            />
        ))}
    </Table>
</ScrollView>}
```

Fig.7.3.2.12.1. Taula Units Stats

Els atributs de les unitats es renderitzen amb un scroll horitzontal per poder visualitzar correctament en pantalla totes les dades.

7.3.3 – Components

Els components son aquelles components JSX que serveixen per complementar lo que es renderitza a les pantalles.

7.3.3.1- AppBar

Aquest component crea una barra en la part superior del mòbil utilitzant les constants de Expo (el editor de React Native).

7.3.3.2 – RosterItem

Aquest component es el encarregat de renderitzar cada Roster a la FlatList de la pantalla Rosters. Te dos botons que permeten, mitjançant el hook useNavigate visualitzar la Roster (ens porta a la pantalla RosterView), editar la Roster (ens porta a la pantalla Squads), o eliminar aquella Roster. Per eliminar-la es fa una petició al server, i si s'elimina correctament es crida a una funció que se li ha passat per props per refrescar la pantalla Rosters i així poder veure que la roster eliminada ja no hi es.

7.3.3.3 SquadItem

Aquest component es el encarregat de renderitzar cada Squad a la FlatList de la pantalla AddSquad. Te dos botons que permeten, el primer, mitjançant el hook useNavigate, permet visualitzar la els atributs base de la squad (ens porta a la pantalla SquadBaseView), i l'altre boto ens permet afegir aquella squad a la Roster. Per afegir-la es fa una petició al server i si aquest respon favorablement ens retorna, mitjançant el hook useNavigate, a la pantalla Squads.

7.3.3.5 – SquadItemView

Es el component que renderitza cada squad a la pantalla SquadView. Quan es renderitza el component, mitjançant el hook useEffect s'executa una funció que fa una crida a al server per obtenir aquella squad. Es renderitzen els atributs d'aquella squad les Unitats que te aquesta squad es passen a una Flatlist que renderitza cada una mitjançant el component UnitItem.

7.3.3.6 – KeywordItem, WeaponItem, UnitItem

Son els components encarregats de renderitzar les dades de les Keywords, Weapons i Units. Aquests objectes estan enllaçats i segueixen el mateix patró.

Quan es renderitzen, fan servir el hook useEffect per executar una funció que demana al server el objecte en qüestió. Aquest objecte es renderitza fent servir una Taula per mostrar

les dades i cada un te una llista de Id del següent, que el passa a una FlatList i aquesta renderitza cada un seguint el mateix procés.

7.3.3.7 – UpdateUnitItem

Es el component encarregat de renderitzar les dades de la unitat que es vol modificar (se li ha passat les dades a aquest component des de la FlatList e la pantalla Units). Ofereix tres botons, un per afegir membres a la unitat, un per restar membres a la unitat i un per visualitzar la unitat.

7.3.3.8 – Loading

Aquest component afegeix una roda animada amb el text “Loading...” i esborrona el fons a cada pantalla mentre s’està accedint obtenint dades necessàries per mostrar contingut en aquella pantalla.

7.3.4 – Estil

S’ha creat un theme (Fig.7.3.4.1) per fer servir a tota la aplicació com a base visual. En aquest theme s’especifica el codi de colors, la mida de la lletra i tota la resta de valors estètics que farà servir tota la aplicació per mantenir un estil conseqüent i comú.

```
import { Platform } from "react-native"

const theme = {
    colors: {
        textPrimary: '#24292e',
        textSecondary: '#fefefe',
        primary: '#16FF00',
        secondary: '#24292e',
        white: '#fefefe',
        error: '#BF202F',
    },
    fontSizes: {
        body: 14,
        subheading: 16,
        heading: 25,
        little: 12,
        diminute: 8
    },
    fonts: {
        main: Platform.select({
            ios: 'Roboto',
            android: 'System',
            default: 'Roboto'
        })
    },
    fontWeigths: {
        normal: '400',
        bold: '700'
    }
}
export default theme
```

Fig.7.3.4.1. Theme.js

S'ha creat un StyledText (Fig.7.3.4.2) y un StyledTextInput(Fig.7.3.4.3) per facilitar la personalització dels components Text i millorar l'aspecte visual.

```
})
export default function StyledText({ children, color, fontSize, bold,
    style, underline, italic, ...restOfProps }) {
    const textStyles = [
        styles.text,
        color === 'primary' && styles.colorPrimary,
        color === 'secondary' && styles.colorSecondary,
        fontSize === 'subheading' && styles.subheading,
        fontSize === 'heading' && styles.heading,
        fontSize === 'body' && styles.body,
        fontSize === 'little' && styles.little,
        bold && styles.bold,
        underline && styles.underline,
        italic && styles.italic,
        style
    ]
    return (
        <Text style={textStyles} {...restOfProps}>
            {children}
        </Text>
    )
}
```

Fig.7.3.4.2. StyledText

```
const styles = StyleSheet.create({
    textInput: {
        borderRadius: 5,
        borderWidth: 1,
        borderColor: theme.colors.secondary,
        paddingHorizontal: 20,
        paddingVertical: 10,
        marginBottom: 10
    },
    error: {
        borderColor: theme.colors.error,
    },
})

const StyledTextInput = ({ style = {}, error, ...props }) => {
    const inputStyle = [
        styles.textInput,
        style,
        error && styles.error
    ]
    return <TextInput style={inputStyle} {...props} />
}
export default StyledTextInput
```

Fig.7.3.4.3. StyledTextInput

7.3.5 – Esquemes de validació

S'ha creat diferents esquemes de validació mitjançant la llibreria Yup (Fig.7.3.5.1) per validar els inputs que es fan servir als formularis amb la llibreria Formik. Aquets schemas son personalitzats per cada apartat.

```
import * as yup from 'yup'
export const createRosterValidationSchema = yup.object().shape({
    rosterName: yup
        .string()
        .min(3, 'Roster name must have 3 char or more.')
        .required('Roster name is required.'),
    pointRestriction: yup
        .number().typeError("Point restriction must be a number.")
        .moreThan(0, 'Point restriction must be more than 0.')
        .lessThan(40000, 'Point restriction can not be more than 40.000.')
        .required('Point restriction is required.'),
    faction: yup.
        string()
        .required('Faction is required.'),
    subFaction: yup.
        string()
        .required('SubFaction is required.'),


})
```

Fig.7.3.5.1. Schema de validació CreateRoster

7.4 – Incidències

Durant el transcurs del projecte, s'han donat una sèrie d'incidències que han provocat canvis en la planificació i desenvolupament de l'aplicació. Aquestes incidències son:

- Nova edició de Warhammer 40K: Durant el desenvolupament de l'aplicació, la edició del Wargame sobre la que s'ha treballat ha quedat obsoleta i s'ha realitzat feina extra per tal d'actualitzar tot el contingut i forma del Backend a la nova edició, per tal de garantir el recorregut de la aplicació a llarg termini.
- Dificultat per limitar les tasques Work in Progress: Moltes de les funcionalitats noves que s'afegeixen en la columna "To-Do" s'han proposat mentre es realitzava un altre tasca, i normalment es perquè es necessita aquesta funcionalitat nova per prosseguir amb el desenvolupament. Això comporta dificultat de mantenir el límit de 5 funcionalitats en "Work in Progress".
- Incompatibilitat de llibreries: Per poder afegir funcionalitats referents a la seguretat, tant de la connexió web com de la autenticació del usuari, han hagut incompatibilitats entre la versió de Java, del IDE IntelliJ i de les diferents llibreries

que fa servir Maven. S'ha realitzat les modificacions pertinents per tal de que les dependències funcionin correctament i l'aplicació sigui segura.

- Carrega de feina mes elevada del previst: Durant del desenvolupament, la quantitat de funcionalitats i el fet de treballar amb una tecnologia totalment nova (React Native) i sense cap experiència en ella, ha provocat un augment molt significatiu del temps de desenvolupament i això comporta un endarreriment de les fites i una disminució de la qualitat.
- Tasques de planificació no assolides: Les incidències anteriors han suposat un impediment a l'hora d'afegir funcionalitats referents al Winrate (generació de llistes en funció del Winrate de les unitats en el panorama competitiu del joc) ja que amb el canvi d'edició comporta una obsolescència del Metajoc actual i de les regles. Per tot això, s'ha donat prioritat a resoldre les altres incidències per sobre d'aquesta funcionalitat.

8 - Anàlisi de resultats, conclusions i possibles ampliacions

L'anàlisi de resultats de l'aplicació mostra que és totalment funcional i compleix amb èxit els objectius establerts, que eren crear una aplicació mòbil per a la gestió de llistes de Warhammer i permetre la seva visualització. S'ha aconseguit desenvolupar un backend net amb una arquitectura hexagonal ben organitzada.

En primer lloc, s'ha realitzat una separació clara de les diferents capes de l'aplicació, incloent controladors, repositoris, casos d'ús, patró command query, viewmodels, mapejadors i entitats. Aquesta estructura permet una millor organització i facilita el manteniment i la implementació de futures funcionalitats.

Pel que fa al frontend, s'ha optat per utilitzar React Native per a la seva implementació. Aquesta elecció ha permès crear una aplicació que és intuïtiva i fàcil d'utilitzar, proporcionant una bona experiència d'usuari. S'ha posat èmfasi en la facilitat d'ús, perquè els usuaris puguin gestionar les seves llistes de Warhammer sense complicacions.

A més de l'anàlisi de resultats i les conclusions prèvies, hi ha algunes possibles millores que es podrien considerar per a l'aplicació:

- Afegir noves funcionalitats: Les funcionalitats presentades al inici del projecte sobre afegir informació sobre el winrate de les diferents unitats de Warhammer i la possibilitat de implementar l'autocreació de llistes d'exèrcit basades en les unitats amb millors winrates son possibles ampliacions per la plataforma. Això permetria als usuaris prendre decisions més informades a l'hora de crear les seves llistes d'exèrcit.
- Afegir una pantalla d'administració per al usuari Admin que faciliti afegir entitats model a la base de dades des de la mateixa aplicació i no haver de fer-ho mitjançant PostMan.
- Millorar la part del domini: Una àrea de millora podria ser evitar tenir un domini anèmic, és a dir, un domini que només conté les entitats sense lògica de negoci. Es podria considerar moure la lògica de les commands/query i dels casos d'ús al domini. Això ajudaria a consolidar la lògica de negoci en una sola capa i a mantenir una millor separació de responsabilitats.

- Utilitzar patrons addicionals: Una millora específica en la part del domini podria ser l'ús del patró Finder, que permet buscar i recuperar dades de manera més eficient. L'aplicació podria beneficiar-se d'aquest patró per a operacions de recerca o filtratge de dades per tal de no repetir codi i millorar el rendiment de la plataforma.
- Afegir Testing: El testing és molt important per garantir la qualitat i confiabilitat. En el backend, les proves avaluen la funcionalitat, la seguretat i l'eficiència en el processament de dades. En el frontend, les proves asseguren una experiència d'usuari fluida, detectant errors visuals i d'interacció. Ambdós tipus de proves ajudarien a proporcionar una aplicació robusta i satisfactòria per als usuaris.

Tenint en compte aquestes millores, l'aplicació mòbil de gestió de llistes de Warhammer podria evolucionar per oferir funcionalitats addicionals i millorar l'arquitectura i la lògica de negoci. Això proporcionaria als usuaris una experiència més completa i enriquida, alhora que asseguraria una millor escalabilitat i mantenibilitat de l'aplicació.

En conclusió, l'aplicació mòbil ha aconseguit amb èxit el seu objectiu principal de permetre als usuaris crear i visualitzar llistes de Warhammer. El backend està ben organitzat i utilitza una arquitectura hexagonal que facilita el desenvolupament i el manteniment. El frontend, realitzat amb React Native, ofereix una interfície intuïtiva i fàcil d'utilitzar. Amb algunes millores i la consideració de feedback dels usuaris, l'aplicació pot continuar creixent i millorant per satisfer encara millor les necessitats dels seus usuaris.

9 - Glossari

4

40K

40.000 ..... 0, 6, 21, 22, 27, 30, 31

A

Age of Sigmar

Un wargame semblant a Warhammer 40K pero amb tematica medieval. 21

API

Interfície de programació d'aplicacions 4, 14

App

Aplicació 31, 35

ARMOR_PENETRATION

Valor que empitjora la tirada de salvació de la unitat objectiu. 41

ATTACKS

Atacs que es poden realitzar amb l'arma. 41

AWS

Amazon Web Services 19, 20, 21, 26, 27, 29, 30, 33, 34

B

Backend

Software encarregat de la logica que fa funcionar l'aplicació 4, 5, 11, 16, 21, 26, 29

BATTLE_ROLE

Rols de batalla de les diferents unitats. Poden ser HQ, TROOPS, ELITE, FAST_ATTACK, HEAVY_SUPPORT, FLYER, DEDICATED_TRANSPORT, LORD_OF_WAR, FORTIFICATION, SUPREME_COMMANDER. 40

BBDD

Bases de Dades 12, 13, 26, 27

Bug

És un error en un programa de programari que causa un comportament no desitjat o inesperat. 37

Ch

Chaos Space Marines

Exèrcit de soldats humans que s'han deixat corrompre per les forces malignes del caos 22

C

Command/Query

Command Query Responsibility Segregation (CQRS) és un patró d'arquitectura de programari que separa la part de l'aplicació que llegeix dades (consultes) de la part que modifica dades (comandos). 42

Commits

D

E

F

I

J

K

L

M

O

P

R

RANGE

Distància màxima que arriba l'arma i que es mesura amb una cinta mètrica. 41

RGPD

Reglament General de Protecció de Dades Europeu 35

Rosters

Llistes de exèrcit. 41

Royalties

Pagaments que cal fer, per a tenir la llicència d'algun dret o producte 29, 33, 35

S

SALVATION

Valor del dau de 6 cares que s'ha d'aconseguir per evitar una ferida. 40

Space Marines

Exèrcit de soldats humans modificats genèticament per ser superiors física i mentalment 22

SQL

Structured Query Language 12, 14

STRENGTH

Valor de força de l'arma que es compara amb la resistència de la unitat objectiu per veure si és capaç de ferir-la. 41

T

TFG

Treball de fi de grau 4, 22, 23, 26, 27, 29, 34

TOUGHNESS

Resistencia de la unitat vers els atacs enemics. 40

U

UML

Unified Modeling Language 40

UX

La UX (User Experience) es refereix a l'experiència que té un usuari en interactuar amb l'aplicació. 37

W

Wargame

Joc de taula per dos o mes jugadors que representa una batalla. Cada jugador dirigeix un exercit de miniatures i ha de vèncer al contrari completant missions o eliminant l'exercit enemic *4*, *22*, *31*, *36*

WEAPON_TYPE

Tipus de l'arma. Pot ser RANGED, MELEE (a distància, o cos a cos). 41

10 – Bibliografia

[1] "Warhammer 40000," [Online]. Available: https://warhammer40000.com/es/. [Accessed: 10-Feb-2023].

[2] "Tozudos 40k," [Online]. Available: https://tozudos40k.blogspot.com/. [Accessed: 10-Feb-2023].

[3] "Docker Guide," [Online]. Available: https://www.baeldung.com/ops/docker-guide. [Accessed: 10-Feb-2023].

[4] "Spring Boot, JPA, Hibernate, and PostgreSQL Restful CRUD API Example," [Online]. Available: https://www.callicoder.com/spring-boot-jpa-hibernate-postgresql-restful-crud-api-example/. [Accessed: 10-Feb-2023].

[5] "Hexagonal Architecture," [Online]. Available: https://alistair.cockburn.us/hexagonal-architecture/. [Accessed: 10-Feb-2023].

[6] "Learn JPA & Hibernate," [Online]. Available: https://www.baeldung.com/learn-jpa-hibernate. [Accessed: 10-Feb-2023].

[7] "Building a RESTful Web Service," [Online]. Available: https://spring.io/guides/gs/rest-service/. [Accessed: 10-Feb-2023].

[8] "Spring Boot," [Online]. Available: https://www.baeldung.com/spring-boot. [Accessed: 10-Feb-2023].

[9] "The Clean Architecture," [Online]. Available: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. [Accessed: 10-Feb-2023].

[10] "Battlescribe," [Online]. Available: https://battlescribe.net/?tab=news. [Accessed: 10-Feb-2023].

[11] "Warhammer 40,000," [Online]. Available: https://play.google.com/store/apps/details?id=com.gamesworkshop.warhammer40k. [Accessed: 10-Feb-2023].

[12] "Amazon Web Services (AWS)," [Online]. Available: https://aws.amazon.com/es/. [Accessed: 10-Feb-2023].

[13] "Aigües de Barcelona," [Online]. Available: https://www.aiguesdebarcelona.cat/. [Accessed: 10-Feb-2023].

[14] "Naturgy," [Online]. Available: https://www.naturgy.es/negocios_y_autonomos?origen=AC&vn=907008019. [Accessed: 10-Feb-2023].

[15] "Visual Studio," [Online]. Available: https://visualstudio.microsoft.com/es/vs/pricing/?tab=business. [Accessed: 10-Feb-2023].

[16] "Windows 11 Pro," [Online]. Available: https://www.microsoft.com/es-es/d/windows-11

[17] “Warhammer 40K New Edition”, [Online]. Avaliable: https://www.warhammer-community.com/es/2023/03/23/a-mindblowing-new-edition-of-warhammer-40000-is-coming/. [Accessed: 21-Apr-2023].