



ESCACS

Cicle Superior de Desenvolupament d'Aplicacions Multiplataforma

Memòria del projecte de DAM

IES Abastos: Curs 2022/23. Grup 7J.

Tutor individual: Xavier Ibañez Catalá

Codi font a <https://github.com/VictoRPiles/escacs>

Víctor Piles Delgado

31 de Maig de 2023

Continguts

1	Identificació, justificació i objectius del projecte	4
1.1	Identificació	4
1.2	Justificació	4
1.3	Objectius	5
2	Disseny del projecte	6
2.1	Arquitectura	6
2.2	Tecnologies	6
2.2.1	Codi font	6
2.2.2	Base de dades	6
2.2.3	Entorn de desenvolupament	7
2.2.4	Aspectes secundaris	7
2.3	Requeriments	7
2.3.1	Sistema de registre	7
2.3.2	Sistema d'inici de sessió	7
2.3.3	Sistema d'emparellament	8
2.3.4	Implementació d'un "chess engine"	8
2.3.5	Implementació d'una interfície gràfica d'usuari	8
2.3.6	Persistència de dades rellevants a les partides	9
2.4	Esbós de la interfície gràfica	10
2.4.1	Formularis d'inici de sessió i registre	10
2.4.2	Pàgina principal	11
2.5	Base de dades	11
3	Desenvolupament del projecte	13
3.1	Conceptes que s'han de conèixer	13
3.1.1	Què és Spring Boot?	13
3.1.2	Programació reactiva: WebFlux, fluxos de dades i l'efecte Hollywood	14
3.2	Estructura i processos	15
3.2.1	Estructura de classes	15
3.2.2	Flux d'execució normal	17
3.3	Codificació	17
3.3.1	Sistema de registre	17
3.3.2	Sistema d'inici de sessió	18
3.3.3	Sistema d'emparellament	18
3.3.4	Implementació d'un "chess engine"	18
3.3.5	Implementació d'una interfície gràfica d'usuari	20
3.3.6	Persistència de dades rellevants a les partides	22
3.4	Proves	22

3.5	Implantació i configuració	23
3.5.1	Implantació del servidor: Maven Deploy i GitHub Packages	23
3.5.2	Implantació del client: Node Package Manager i Electron Forge	24
4	Resultat i conclusions finals	25
4.1	Avaluació i aprenentatge	25
4.1.1	Problemes i obstacles	25
4.1.2	Aprenentatge i conceptes nous	25
4.2	Proposta de millores	26

Figures

2.1	Arquitectura de tres capes	6
2.2	Tecnologies	7
2.3	Flux d'ús del programa	8
2.4	Esbós dels formularis d'inici de sessió i registre	10
2.5	Pàgina principal	11
2.6	Diagrama entitat relació	12
3.1	Programació reactiva	14
3.2	Diagrama de classes de l'API	16
3.3	Diagrama de classes del motor	16
3.4	Flux d'execució normal	17
3.5	Exemples de desplaçament vectorial i no vectorial	20
3.6	Captures dels formularis d'inici de sessió i registre	21
3.7	Captura de la pàgina principal	21
3.8	Captures dels taulers de jugadors i sol·licituds	22
3.9	Tests JUnit i Postman	23
3.10	Paquet del servidor al repositori de GitHub	24
3.11	Client empaquetat per a Debian i Fedora Linux	24
4.1	Escac i mat	26

1 Identificació, justificació i objectius del projecte

En aquest capítol s'identificaran els conceptes més importants del projecte i es justificarà l'existència i els objectius d'aquest.

1.1 Identificació

La intenció d'aquest projecte és dur a terme la creació d'una versió digital del joc d'escacs.

L'escacs és un joc d'estratègia que es juga en un tauler amb seixanta-quatre caselles disposades en una graella de vuit per vuit i es compon de setze peces per jugador.

Existeixen sis tipus de peces diferents, cadascuna amb moviments únics i propietats distintives, aquestes són: un rei, una dama, dues torres, dos alfils, dos cavallers i vuit peons. Es poden diferenciar les peces amigues i enemigues pel color d'aquestes.

El jugador que controla les peces blanques es mou primer, després els torns s'alternen.

Altre element important és el rellotge; cada jugador compta amb un màxim de deu minuts per realitzar els moviments, si el rellotge es queda sense temps, el jugador perdrà la partida.

L'objectiu del joc és fer escac i mat al rei de l'oponent, és a dir, el rei està sota un atac immediat o "escac" i no hi ha manera de que escape. També hi ha diverses maneres de què un joc acabe en empat.

1.2 Justificació

S'ha decidit dur aquest projecte a terme per diversos motius, alguns personals i altres acadèmics.

En primer lloc, la gran afició de l'autor pels escacs.

Per altra banda és un projecte molt recomanable per a millorar els coneixements en diferents paradigmes de software com la programació orientada a objectes (s'identifiquen clarament tècniques pròpies d'aquest, com herència o polimorfisme), la programació multifil (ja que dóna peu a l'implementació de dimonis, com el rellotge) o la computació distribuïda (pel seu element multijugador).

Per últim, en aquest projecte s'utilitzen diversos tèmics vistos a classe com programació en Java (mòdul Programació), bases de dades (mòduls Bases de dades i Accés a dades), control de versions, entorns de desenvolupament integrat, documentació (mòdul Entorns

de desenvolupament), creació d'una interfície gràfica d'usuari (mòdul Desenvolupament d'interfícies) i programació multifil i en xarxa (mòdul Programació de serveis i processos).

1.3 Objectius

Es tenen com a objectius:

- El desenvolupament un joc d'escacs funcional que segueixi les regles del joc, per tant s'ha de implementar un "Chess engine".
- L'implementació d'una interfície gràfica d'usuari atractiva i fàcil d'utilitzar. Aquesta comptarà amb un tauler i un historial de moviments, així com de pàgines d'inici de sessió, registre, llistat de jugadors, etc.
- La creació d'un sistema multijugador en xarxa, l'aplicació permetrà emparellar a jugadors des de màquines remotes i es connectarà amb un servidor que farà de "backend", aquest s'encarregarà de gestionar la partida i connectar-se amb la base de dades. El menú principal mostrarà els jugadors disponibles a l'espera de parella per a jugar i també, en cas de no voler jugar amb cap d'ells, una opció de posar-se a l'espera.
- Permetre la persistència de dades rellevants a les partides. Cada jugador al connectar-se podrà consultar les seves partides i els moviments passats.

2 Disseny del projecte

En aquest capítol es plantejaran diferents aspectes relacionats amb la planificació i els requeriments del projecte.

2.1 Arquitectura

Al projecte es pot observar l'arquitectura de tres capes, ja que compta amb un mòdul que fa de client o "frontend", un servidor o "backend" i una base de dades.

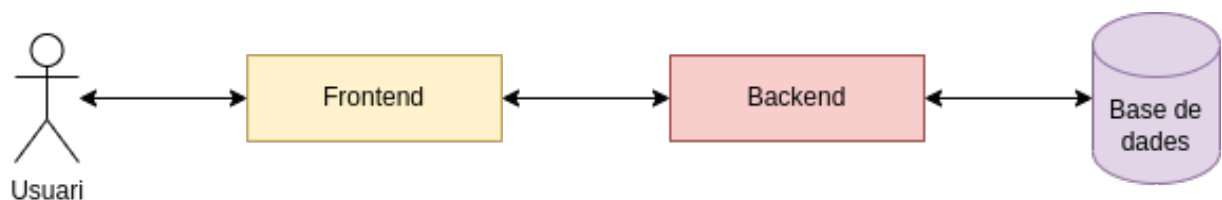


Figura 2.1: Arquitectura de tres capes

2.2 Tecnologies

2.2.1 Codi font

El frontend utilitza tecnologies web com HTML, CSS (Bootstrap), TypeScript o NodeJS, empaquetades amb Electron, i es comunica amb el backend mitjançant la llibreria Fetch i el protocol HTTP.

El backend es divideix en dos mòduls diferenciats:

1. S'ha implementat una API Rest amb Java i el framework Spring Boot, aquest rep les peticions HTTP que el client envia als endpoints de la API i s'encarrega de processar-les i d'actuar com a intermediari amb la base de dades. D'aquesta forma mai es violarà l'arquitectura de tres capes.
2. S'ha desenvolupat un mòdul que s'encarrega de la lògica del motor d'escacs i defineix el comportament i les normes del joc.

2.2.2 Base de dades

La base de dades utilitza el motor H2 incrustat ("embedded") i es comunica amb Spring Boot mitjançant la implementació del ORM Hibernate a Spring Data JPA.

2.2.3 Entorn de desenvolupament

Com a entorn integrat de desenvolupament s'utilitzarà IntelliJ IDEA.

Per al control de versions s'utilitzaran Git i GitHub, a més dels GitHub Workflows per a la integració, escalabilitat i testing del codi.

Per a facilitar el desenvolupament de la API, es realitzaran les peticions HTTP mitjançant Postman.

2.2.4 Aspectes secundaris

Es farà ús de GIMP i Inkscape per a l'edició d'imatges.

Els diagrames i esbossos es dissenyen amb la ferramenta draw.io

Per a escriure la documentació (aquest document) s'utilitzarà L^AT_EX (LaTeX) i Overleaf.

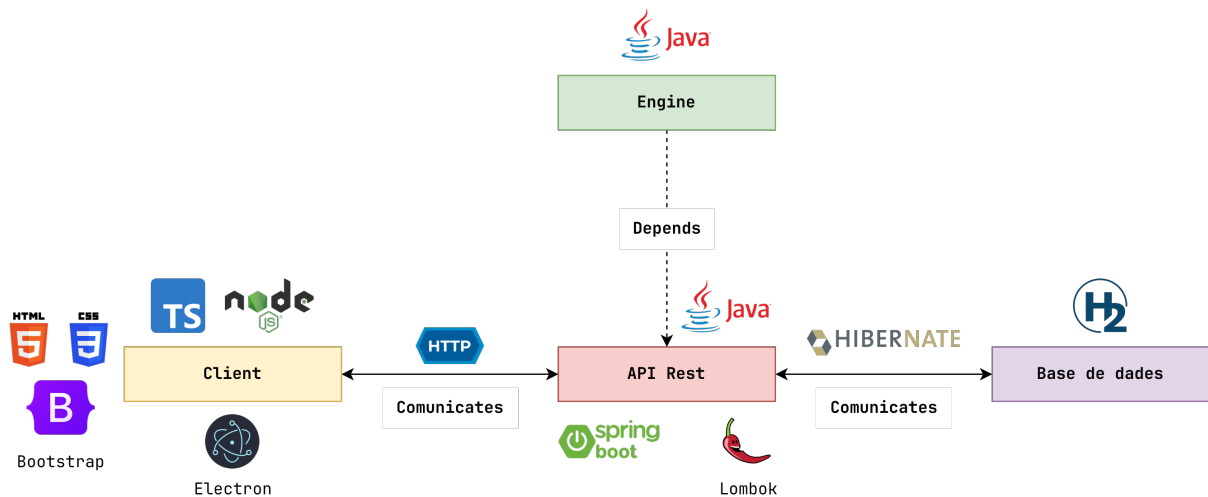


Figura 2.2: Tecnologies

2.3 Requeriments

2.3.1 Sistema de registre

Per a poder accedir a la aplicació els usuaris hauran de crear-se un compte introduint correu i contrasenya i assignar-se un nom d'usuari, les dades es validaran utilitzant Spring Validation i les credencials seran guardades de forma segura en la base de dades.

2.3.2 Sistema d'inici de sessió

Una vegada registrats, els usuaris hauran d'autenticar-se amb correu i contrasenya, aquestes dades es contrastaran amb les existents en la base de dades i se'ls permetrà o no l'accés.

2.3.3 Sistema d'emparellament

Una vegada autenticats, els usuaris tindran accés a la llista de jugadors i podran enviar i rebre sol·licituds de joc.

Quan un usuari vol jugar amb altre, li envia una sol·licitud al destinatari, aquest podrà acceptar-la o rebutjar-la. Si decideix acceptar-la, els jugadors s'emparellaran a una nova partida.

Una vegada emparellats, els jugadors intercanviaran els missatges corresponents amb el servidor per a dur a terme la partida, fins la finalització d'aquesta.

2.3.4 Implementació d'un "chess engine"

El motor d'escacs funciona com una "caixa negra" amb la que el servidor pot interactuar. El servidor executarà en aquest les jugades que rep dels clients, el motor les processarà i tornarà un estatus com a resposta, indicant el resultat d'aquesta execució.

Al joc trobem sis tipus de peces diferents que hereten d'una classe superior. La principal diferència entre aquestes es troba als tipus de moviments vàlids que poden realitzar, tenint en compte les normes del joc, s'han d'implementar aquests comportaments de forma individual per a cada peça.

Els moviments són la forma d'interactuar amb el motor, s'entén per moviment cadascuna de les jugades que s'executen al tauler. Cal destacar que si un jugador realitza un moviment invàlid, no perdrà el torn, sinó que se li permetrà seguir fins que faci un vàlid. En aquest moment el torn passarà a l'oponent o s'acabarà la partida (si el moviment ocasiona l'estancament de la partida o l'escac i mat).

Els moviments estàndard o comuns a totes les peces són els de desplaçament i atac, encara que hi han altres moviments especials que només poden realitzar uns tipus de peces determinats, així com els moviments "matar al pas", el bot inicial o la promoció al peó o l'enroc per al rei i la torre.

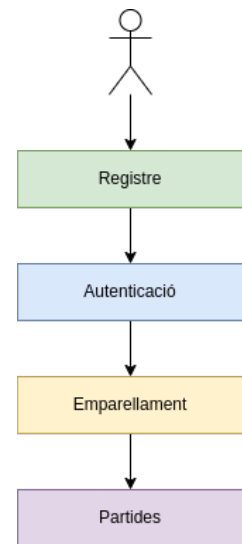


Figura 2.3: Flux d'ús del programa

2.3.5 Implementació d'una interfície gràfica d'usuari

Per a codificar aquest apartat s'han utilitzat tecnologies web com HTML, CSS (Bootstrap), TypeScript o NodeJS, empaquetades amb Electron.

S'ha utilitzat HTML5 per a l'organització i el disseny estructural de les pàgines. HTML és un llenguatge de marcatge utilitzat per a crear i estructurar contingut en pàgines web. És l'estàndard per al desenvolupament web i proporciona una estructura bàsica per a organitzar elements en una pàgina.

HTML utilitza etiquetes o tags per a marcar els diferents elements d'una pàgina web. Cada etiqueta té una funció específica i defineix com s'ha de mostrar o interpretar el contingut. Les etiquetes permeten crear encapçalaments, paràgrafs, llistes, taules, formularis,

imatges, enllaços i molts altres elements que es poden trobar en una pàgina web. Aquestes etiquetes s'estructuren jeràrquicament, formant elements anidats que defineixen la relació i l'organització del contingut. A més, HTML també permet l'ús d'atributs en les etiquetes per a proporcionar informació addicional o personalitzar el comportament dels elements.

En resum, HTML és el fonament de la web, ja que permet als desenvolupadors descriure i estructurar el contingut perquè els navegadors web puguin interpretar i mostrar les pàgines adequadament.

En quant al format i l'estil visual, Bootstrap ha segut un component imprescindible. Bootstrap és una biblioteca de codi obert que proporciona un conjunt de classes i estils predefinitos per ajudar a desenvolupar interfícies web responsives de manera ràpida i senzilla. Aquesta biblioteca utilitza CSS per definir els estils i la presentació dels elements HTML, permetent als desenvolupadors estilitzar i estructurar els components de la seva pàgina web de manera coherent i consistent. Amb Bootstrap CSS, és possible aplicar estils comuns, com a mides de text, colors, marges, alineacions i respostes a diferents dispositius, amb facilitat. Això permet crear un disseny visualment atractiu i professional, amb una experiència de l'usuari optimitzada en diferents plataformes i resolucions de pantalla. No obstant, s'ha escrit codi CSS addicional quan era necessari.

S'ha optat per utilitzar el llenguatge TypeScript per a la part lògica i interactiva del projecte. TypeScript és un llenguatge de programació de codi obert que amplia la sintaxi de JavaScript. Aquest llenguatge proporciona funcionalitats addicionals com ara tipat estàtic, orientació a objectes i suport per a característiques modernes de JavaScript.

El tipat estàtic és una característica clau de TypeScript que permet declarar i assignar tipus de dades a les variables, paràmetres de funcions i retorn de valors, entre d'altres. Això ajuda a detectar errors de tipus durant el desenvolupament, millorant la robustesa i la escalabilitat del codi.

A més, TypeScript facilita la programació orientada a objectes, permetent la creació de classes, herència, interfícies i altres conceptes propis de la programació orientada a objectes. Aquesta organització estructurada del codi ajuda a millorar la seva claredat i reutilització.

A més TypeScript també proporciona suport per a les funcionalitats més modernes de JavaScript, com ara els mòduls ES6, les funcions de fletxa i altres característiques sintàctiques avançades.

Per a realitzar i rebre peticions HTTP s'ha utilitzat l'API Fetch. És una interfície proporcionada per JavaScript que permet realitzar peticions HTTP asíncrones a un servidor i obtenir les respostes en format de Promises. L'API Fetch és compatible amb els formats de dades comuns, com ara JSON, i permet realitzar operacions addicionals, com ara afegir interceptors per manipular les peticions i respostes abans que siguin processades o afegir opcions personalitzades, com capçaleres i dades en el cos de la petició.

2.3.6 Persistència de dades rellevants a les partides

Les credencials dels usuaris i la informació de les partides es guarden a una base de dades implementada amb H2. S'ha triat aquest sistema gestor de base de dades per la seua senzilla integració amb Spring Boot, ja que no requereix cap configuració per a fer-lo funcionar.

Cada partida ha de tindre un identificador únic i estar relacionada amb dos jugadors, enllaçats

a aquesta s'agruparan els moviments realitzats amb format "Portable Game Notation".

Per a l'emmagatzemament de dades, s'ha utilitzat la implementació del ORM Hibernate proporcionada per Spring Data JPA. Spring Data JPA té com a objectiu simplificar l'accés i la interacció amb bases de dades en aplicacions Java. Proporciona una capa d'abstracció d'alt nivell sobre l'API de Persistència de Java (JPA), que és una especificació per a frameworks d'ORM (Mapeig Objecte-Relacional) en Java.

Spring Data JPA combina el poder de Spring Boot i JPA per oferir als desenvolupadors una manera simplificada de treballar amb bases de dades relacionals. Elimina la necessitat d'escriure codi boilerplate per a operacions comunes de bases de dades, com ara operacions CRUD (Crear, Llegir, Actualitzar, Esborrar), creació de consultes i gestió de transaccions.

Amb Spring Data JPA, pots definir repositoris que encapsulen les operacions de bases de dades en entitats. Seguint determinades convencions de nomenclatura o utilitzant consultes personalitzades, pots generar automàticament consultes SQL per a operacions comunes sense haver d'escriure declaracions SQL explícites. Això redueix considerablement la quantitat de codi que cal escriure i millora la productivitat de desenvolupament.

Utilitzant Spring Data JPA, pots aprofitar els avantatges de JPA i els conceptes d'ORM, alhora que beneficis de les funcionalitats i convencions proporcionades per Spring Data. Simplifica l'accés a bases de dades, fomenta la reutilització de codi i ajuda els desenvolupadors a centrar-se en la lògica de negoci en lloc de les interaccions de baix nivell amb la base de dades.

2.4 Esbós de la interfície gràfica

2.4.1 Formularis d'inici de sessió i registre

El diagrama mostra dos formularis side-by-side. El formulari 'Iniciar sessió' a l'esquerra té dos camps d'entrada: 'Email' amb un icona '@' i 'Contrasenya' amb un icona '*'. A sota hi ha un botó 'Enviar' de color verd i un enllaç 'Registrar-se' de color blau. El formulari 'Registrar-se' a la dreta té tres camps d'entrada: 'Nom d'usuari' amb un icona 'O', 'Email' amb un icona '@' i 'Contrasenya' amb un icona '*'. A sota hi ha un botó 'Enviar' de color verd i un enllaç 'Iniciar sessió' de color blau. Ambdós formularis tenen una barreja de color rosa a la part inferior amb el text 'Missatge d'error!'.

Figura 2.4: Esbós dels formularis d'inici de sessió i registre

2.4.2 Pàgina principal

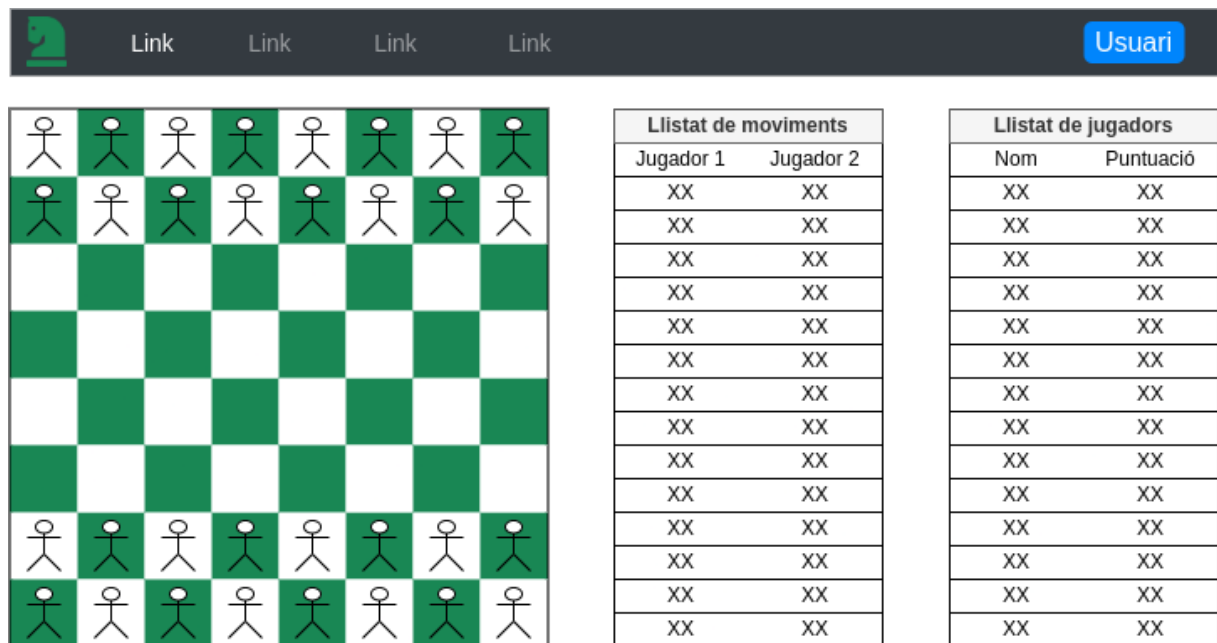


Figura 2.5: Pàgina principal

2.5 Base de dades

La base de dades contindrà quatre entitats: jugadors, moviments, partides i sol·licituds de joc.

Dels jugadors volem guardar el seu nom d'usuari, el seu correu, la seua contrasenya encriptada i la seua puntuació. El nom i la puntuació seran visibles per a la resta d'usuaris. El correu i la contrasenya s'utilitzaran per a autenticar-se.

En quant als moviments és important saber el valor d'aquests, que s'almacenarà amb format "Portable Game Notation", així com quin ha segut el seu efecte o estatus a la partida. A més hem de relacionar-los amb el jugador que l'ha realitzat i la partida on s'ha executat.

A la taula partides es guardarà la sol·licitud de joc amb la que han accedit els usuaris (d'aquesta manera es poden relacionar els usuaris participants amb la partida), també guardarem qui ha segut el guanyador i si la partida ha acabat o escara està sent jugada.

Les sol·licituds de joc són un element més bé temporal, ja que deixen de ser vàlides una vegada s'accepten, es rebutgen o expiren. En aquesta tabla es interessant almacenar l'hora de creació i d'expiració, també l'estat de la sol·licitud i els usuaris involucrats en aquesta.

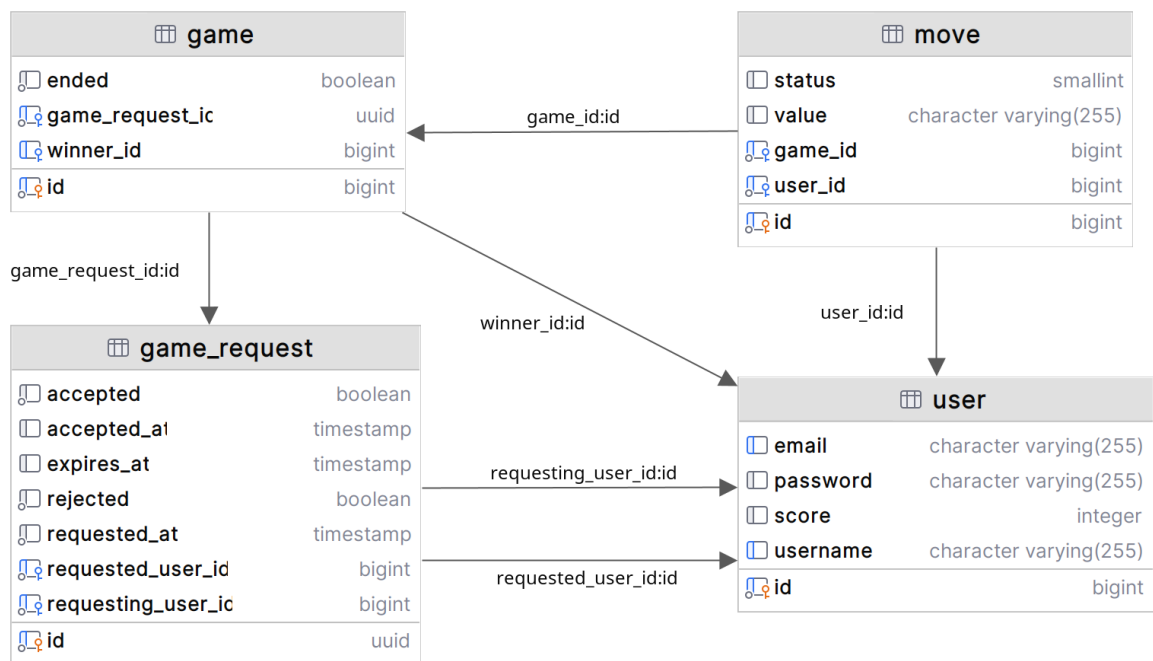


Figura 2.6: Diagrama entitat relació

3 Desenvolupament del projecte

En aquest capítol s'explicarà el funcionament del programa, es mostrarà com s'ha donat solució a cada un dels requeriments plantejats a l'apartat de disseny, es descriurà el procés de proves que s'ha realitzat per assegurar la qualitat del programari i la forma d'exportar el projecte a fitxers executables per l'usuari final.

3.1 Conceptes que s'han de conèixer

3.1.1 Què és Spring Boot?

Per al desenvolupament del backend, s'ha utilitzat el framework Spring Boot. Spring Boot és una eina de desenvolupament de programari lliure basada en Java. Facilita la creació d'aplicacions Java de qualitat de forma senzilla i ràpida. Aprofita el poder de l'ecosistema Spring però redueix la necessitat de configuració i personalització.

Spring Boot s'utilitza per construir aplicacions Java autònomes, és a dir, aplicacions que no requereixen un servidor web separat per a funcionar. Això és possible gràcies al seu suport per a contenidors integrats. Aquesta característica permet que l'aplicació s'executi com un fitxer JAR amb un servidor web incorporat, com ara Tomcat o Jetty.

Una altra característica clau de Spring Boot són els "starters". Els "starters" són conjunts de dependències que simplifiquen la configuració inicial d'una aplicació. Proporcionen configuracions predeterminades per a casos d'ús comuns, com ara aplicacions web o accés a bases de dades. Amb l'ús dels "starters", es pot començar a desenvolupar una aplicació amb les dependències adequades amb poques línies de codi.

Spring Boot també ofereix una funcionalitat d'autoconfiguració. Utilitza tècniques de programació reflexiva per analitzar les llibreries i classes disponibles en l'aplicació i configura automàticament els components segons les necessitats detectades. Això evita haver de realitzar configuracions manuals i permet als desenvolupadors centrar-se en la lògica de l'aplicació en lloc de preocupar-se per la configuració tècnica.

Amb Spring Boot, és fàcil desenvolupar aplicacions RESTful, gestionar peticions web, interactuar amb bases de dades, implementar seguretat, programar tasques i integrar-se amb sistemes de missatgeria.

En resum, Spring Boot és una eina que simplifica el desenvolupament d'aplicacions Java, permetent als desenvolupadors crear aplicacions de qualitat de forma senzilla i ràpida. Amb el seu suport per a contenidors integrats, els "starters" i la funcionalitat d'autoconfiguració, Spring Boot redueix la complexitat i la necessitat de configuració, permetent als desenvolupadors centrar-se en la creació de lògica d'aplicacions.

3.1.2 Programació reactiva: WebFlux, fluxos de dades i l'efecte Hollywood

La programació d'API Reactiva és un enfocament per a construir APIs que abraça els principis reactius. La programació reactiva és un paradigma de programació que es centra en la programació asíncrona i basada en esdeveniments, permetent que els sistemes siguin més responsius, escalables i resistents. En el context de les APIs, la programació reactiva permet gestionar un gran nombre de sol·licituds concurrents amb un nombre reduït de fils d'execució, millorant així l'aprofitament dels recursos i reduint les operacions de bloqueig.

Spring WebFlux és un framework proporcionat per l'ecosistema de Spring. Està dissenyat per a donar suport a models de programació reactius, incloent fluxos de dades (o streaming) i entrada/eixida no bloquejant. Spring WebFlux permet als desenvolupadors construir endpoints d'API reactius aprofitant la llibreria [Reactor](#). Aquests endpoints són notificats i invocats pel framework quan arriben sol·licituds, seguint un model basat en esdeveniments. Aquesta inversió de control és similar a l'"efecte Hollywood", on el framework assumeix el comandament i crida els controladors adequats quan siga necessari.

Ara, aborem la relació entre l'"efecte Hollywood" i la programació d'API Reactiva amb Spring WebFlux. Encara que no hi ha una connexió directa entre els dos conceptes, podem fer una analogia per a entendre'ls millor. L'efecte Hollywood, conegut també com "No ens crides, ja te cridarem nosaltres", és un principi que s'aplica en la programació reactiva i es refereix a invertir la direccionalitat de les sol·licituds. Segons aquest principi, el client no farà trucades periòdiques al servidor per a saber si ha ocorregut o no un esdeveniment, sinó que es subscriurà a un endpoint i en el que el servidor anirà publicant dades quan corresponga.

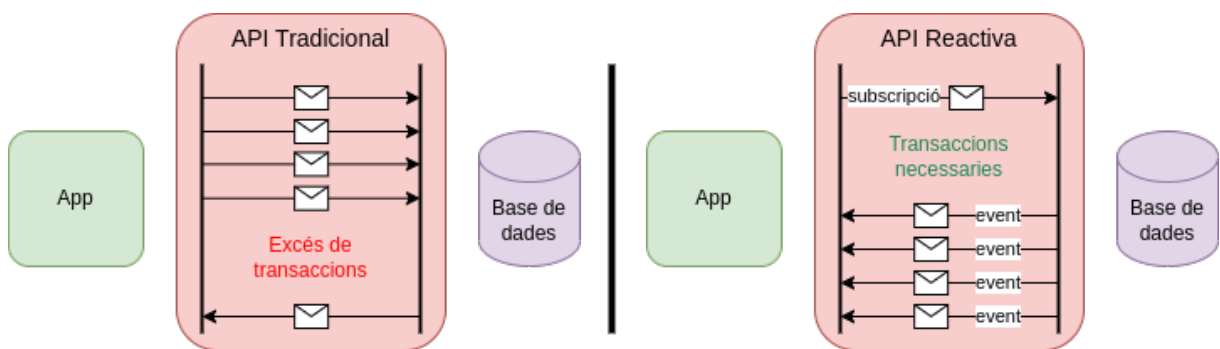


Figura 3.1: Programació reactiva

Des de el punt de vista del codi, per a crear un servici que publique dades a un endpoint reactiu utilitzant Spring WebFlux s'ha d'especificar que aquest generarà dades del tipus "TEXT_EVENT_STREAM_VALUE" i s'ha d'utilitzar la classe Flux de la següent manera:

```
@GetMapping(  
    path = "/path/to/endpoint",  
    produces = MediaType.TEXT_EVENT_STREAM_VALUE  
)  
public Flux<List<Object>> stream() {  
    return Flux.just(repository.getAll());  
}
```

Per a subscriure's des de TypeScript s'utilitza la classe EventSource d'aquesta forma:

```
let stream = new EventSource("/path/to/endpoint");
stream.onmessage = (event) => {
  /* Funció a executar */
}
```

3.2 Estructura i processos

A aquest apartat s'aprofundirà en l'estructura i fluxos d'execució del codi.

Encara que el projecte compta amb diferents mòduls, només s'abordaran l'estructura i processos del backend, ja que és ací on es pot observar una arquitectura més clara i interessant, la resta del codi són fitxers TypeScript que realitzen tasques espontànies al frontend i gestionen peticions HTTP.

3.2.1 Estructura de classes

El backend es pot desgranar en dos parts diferenciades; l'API que s'encarrega de les comunicacions i fer les tasques pròpies d'un servidor, i el motor d'escacs, que s'encarrega de la lògica del joc.

Per a interactuar correctament amb Spring Boot s'han d'anotar les classes com a components (concepte similar a model, vista i controlador), existeixen principalment quatre tipus:

1. Entitat: és una classe POJO (Plain Old Java Object) i representa una taula en la capa de persistència de dades
2. Controlador: el seu propòsit principal és gestionar les peticions HTTP entrants i retornar respostes adequades, és responsable de definir els punts d'entrada de l'aplicació web, coneguts com a endpoints.
3. Servici: és una classe que conté la lògica de negoci i s'encarrega de coordinar les diferents operacions dins de l'aplicació, proporcionant una capa d'abstracció entre els controladors i la capa de persistència de dades.
4. Repositori: és una interfície o classe que proporciona una abstracció per accedir a les dades de l'aplicació i realitzar operacions de persistència, eliminant la complexitat de les operacions de baix nivell i permetent una interacció més senzilla i escalable amb la capa de persistència de dades.

A banda, s'han escrit excepcions personalitzades per a cada tipus d'error i un gestor global per a aquestes, així com classes utilitàries per a encriptació, seguretat o configuració.

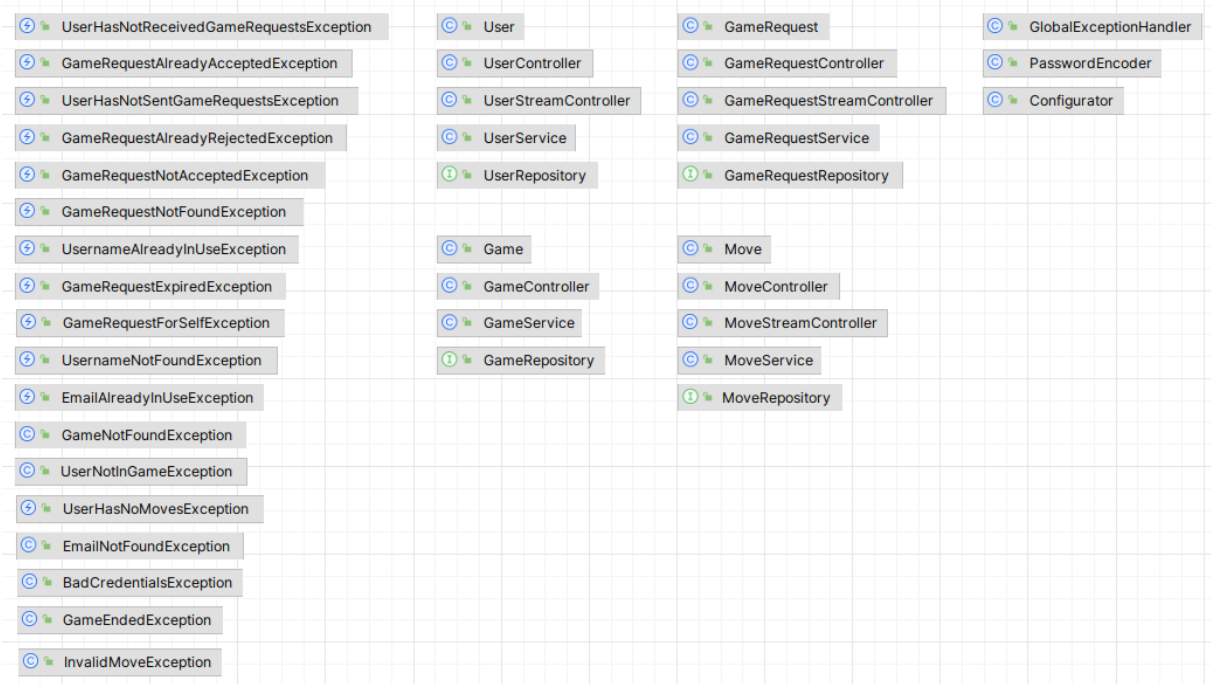


Figura 3.2: Diagrama de classes de l'API

El motor utilitza classes convencionals, que no interactuen amb el framework Spring Boot, es pot observar els diferents tipus de peces i l'herència que comparteixen amb la classe pare, així com classes de presència pràcticament obvia a un motor d'escacs, per exemple la classe tauler o casella. També trobem classes útils per a traduir informació a objectes de Java, s'aprofundirà més en aquest aspecte a la secció d'implementació d'un “chess engine”.

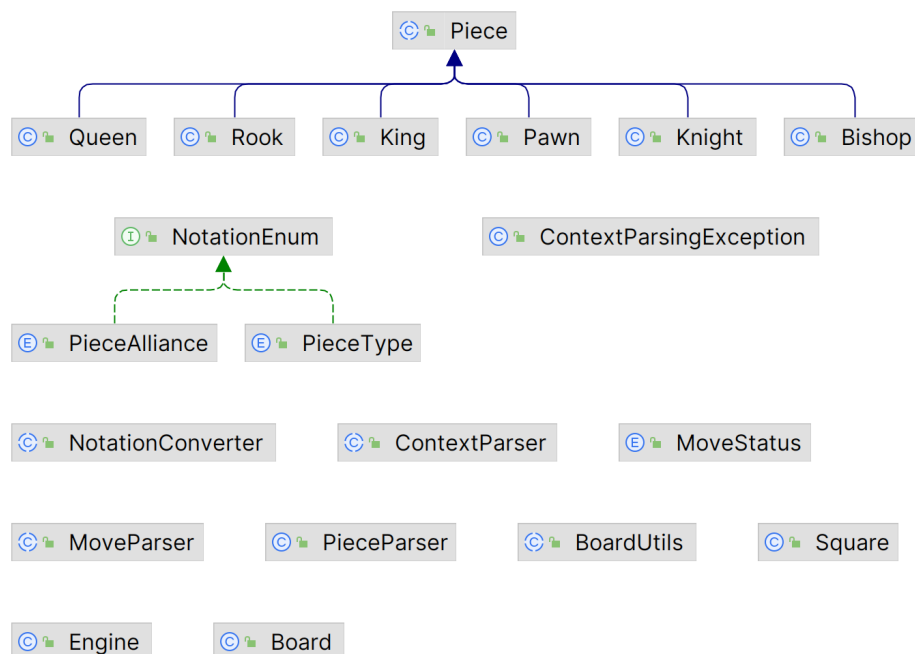


Figura 3.3: Diagrama de classes del motor

3.2.2 Flux d'execució normal

En la majoria dels casos, el flux d'execució de les peticions consta dels següents passos:

1. L'usuari interactua a través del frontend i es genera una sol·licitud HTTP.
2. La sol·licitud aplega al endpoint d'un controlador, aquest la gestiona i la passa al servici que corresponga.
3. El servici realitza alguna acció amb la informació rebuda, cridarà al motor d'escacs o a la base de dades si és necessari i comunicarà el resultat al controlador del pas anterior.
4. El controlador generarà una resposta HTTP i contestarà al client.

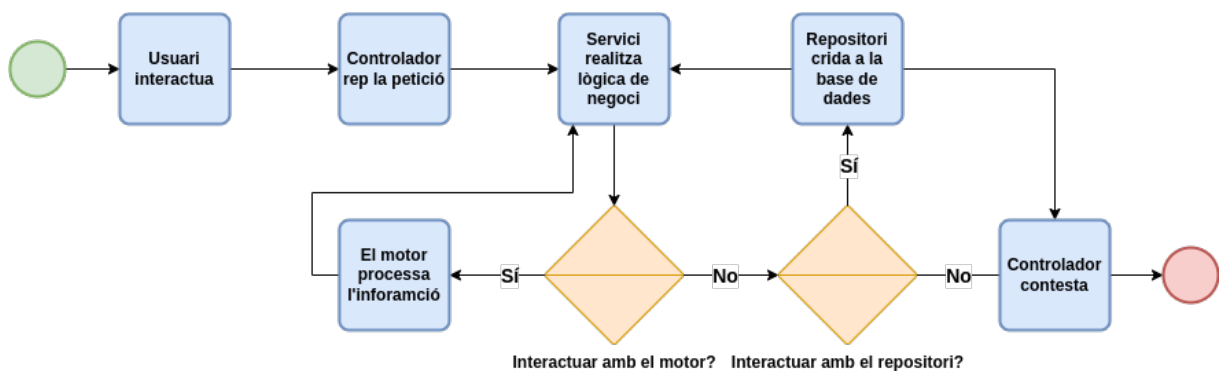


Figura 3.4: Flux d'execució normal

3.3 Codificació

Es detallarà en profunditat com s'ha abordat i resolt cadascun dels requeriments establerts en l'apartat de disseny. Es descriurà el procés i les estratègies utilitzades per garantir que cada requeriment s'ha complert de manera efectiva i satisfactòria. A més, es proporcionaran exemples concrets i evidències de com s'han implementat les solucions per a cada requeriment, mostrant la seva contribució a la qualitat general del programari.

3.3.1 Sistema de registre

El registre consisteix en la validació i creació d'un nou usuari a partir de les dades que s'introdueixen al formulari de registre.

Quan un usuari plena el formulari amb les dades que es sol·liciten i fa clic al botó d'enviar els camps del formulari es tradueixen a format JSON i s'envia una petició HTTP de tipus POST al servidor, aquest és l'encarregat de processar-la i tornar feedback al frontend.

En primer lloc es du a terme una validació les dades, per a açò s'utilitza tant Spring Validation com codi propi de l'aplicació. Per part de Spring Validation, s'utilitzen funcionalitats per a comprovar que ningun dels camps introduïts estan buits, també es comprova que el correu electrònic compleix els requisits definits a una expressió regular, aquest ha de contindre elements com l'arrova (@) o un domini (.com, .es) i només podrà contindre caràcters alfanumèrics. Després es fan consultes a la base de dades per a assegurar-se de que el nom d'usuari i el correu electrònic són únics a l'aplicació.

Una vegada validat, s'ha d'inserir l'usuari a la base de dades, per a fer-ho de forma segura, abans s'encrypta la contrasenya mitjançant funcions criptogràfiques presents al paquet [javax.crypto](#), en concret s'utilitza el algorisme de hash PBKDF2.

Finalment, el servidor tornarà feedback al client, si els passos anteriors s'han executat satisfactòriament, rebrà la informació referent al nou usuari registrat, en cas contrari rebrà un missatge d'error amb la descripció d'aquest. Aquesta resposta es veurà reflectida al formulari, si és favorable es permetrà l'accés, però si no és així es mostrarà una alerta amb la descripció del problema.

És important destacar que s'utilitza la [API Jackson](#) per a amagar la contrasenya del usuari, de manera que mai s'enviarà a través de la xarxa (encara que prèviament ha segut encryptada).

3.3.2 Sistema d'inici de sessió

Aquest sistema rep les credencials que s'introdueixen al formulari d'inici de sessió i les contrasta amb les que es troben a la base de dades.

Comparteix característiques amb l'apartat anterior, per exemple el procés de validació es molt semblant i utilitza les mateixes funcions. La principal diferència la trobem a l'hora de comprovar si la contrasenya introduïda (que serà text plà) coincideix amb la que hi ha al sistema (que estarà encryptada), per a açò s'utilitza un procés d'enginyeria inversa, s'encrypta amb el mateix algorisme que s'havia utilitzat al registre i es comparen byte per byte. D'aquesta forma mai s'intenten descriptar les credencials que ja consten a la base de dades.

Per últim, novament de forma similar a l'apartat anterior, el servidor contesta al client i es produeixen els canvis necessaris.

3.3.3 Sistema d'emparellament

Per a emparellar-se, els jugadors han de poder enviar i rebre sol·licituds de joc, per tant podem diferenciar dos rols, el d'emissor i el de receptor.

Els jugadors registrats a l'aplicació apareixeran a un llistat, des de ací l'usuari emissor podrà seleccionar al receptor i enviar-li una sol·licitud a la seua bústia, que es troba a la pàgina de sol·licituds. En realitat la bústia es una tabla (HTML) subscripta a un endpoint reactiu, on el servidor publica mitjançant un flux de dades les sol·licituds pendents dels usuaris que ho consulten, i que s'actualitza cada vegada que hi ha alguna novetat. Es considera que una sol·licitud està pendent quan no ha expirat ni ha segut acceptada o rebutjada.

En aquest apartat observem un exemple de l'ús de la programació reactiva, tant en la tabla d'usuaris com en la de sol·licituds de joc, ja que aquestes s'actualitzen automàticament quan ocorre algun event als endpoints als que estan subscribes, si s'hagués utilitzat el paradigma tradicional es tindria que haver incorporat un botó per a refrescar-les, augmentant la quantitat de peticions al servidor i a la base de dades.

3.3.4 Implementació d'un "chess engine"

El motor d'escacs funciona com una "caixa negra" en la que el servidor executa les jugades que rep dels clients, aquestes jugades s'envien en format "portable game notation", per tant s'ha d'extraure la informació i transformar-les en objectes Java per a poder ser analitzades.

La notació de joc portàtil d'escacs, coneguda com a "Portable Game Notation" (PGN), és una forma estandarditzada d'enregistrament de partides d'escacs, els seus moviments i informació relacionada. Permet compartir, analitzar i emmagatzemar les partides d'escacs utilitzant text pla. A més, el PGN permet afegir informació addicional com ara encapçalaments de partida, detalls de l'esdeveniment, noms dels jugadors i anotacions.

La notació PGN representa cada moviment amb una combinació del símbol de la peça i la casella de destí. Ací alguns exemples:

- Moviments de peó: Els peons són representats pel seu símbol (sense lletra) i la casella de destí. Per exemple, e4 representa moure el peó a la casella e4.
- Moviments de peces: Les peces són representades per la seua lletra majúscula, seguida de la casella de destí. Per exemple, Nf3 representa moure un cavall a la casella f3.
- Captures: Les captures es denoten amb una 'x' entre el símbol de la peça i la casella de destí. Per exemple, Bxc5 representa un alfil capturant una peça en la casella c5.
- Enroc: L'enroc al costat del rei es representa amb la notació O-O, mentre que al costat de la reina s'indica amb O-O-O.
- Escac i escac i mat: '+' s'utilitza per indicar un escac, i '#' s'utilitza per indicar escac i mat. Per exemple, Qh5+ representa una reina fent escac al rei oponent en la casella h5.
- Altres símbols: '=' s'utilitza per indicar una promoció de peó, i '!' i '?' s'utilitzen per a denotar moviments bons i dolents, respectivament.

Aquest sistema de notació és complex, per això, a aquest projecte s'ha implementat una versió simplificada i més fàcil d'interpretar de PGN.

Tornant al tema del motor d'escacs, aquest ha de ser capaç de reproduir el context del tauler i crear un objecte a partir d'aquesta informació, de manera que quan volem executar un moviment, no només haurem de proporcionar el seu valor, sinó també dades sobre l'estat de la resta de caselles i les peces que contenen.

Una vegada s'han proporcionat les dades, es transformaran en diferents instàncies d'objectes Java i es podrà analitzar el moviment, aquest estudi serà diferent per a cada tipus de peça. Al joc trobem sis tipus diferents, que hereten d'una classe superior. La principal diferència entre aquestes es troba als moviments vàlids que poden realitzar, tenint en compte les normes del joc, s'han d'implementar aquests comportaments de forma individual per a cada peça. Quan es tracta del comportament de les peces, les podem dividir en dos grups; les que es mouen de forma vectorial i les que no.

- Comportament vectorial: Aquestes peces es desplacen seguint la direcció d'un vector fins que es troben amb altra peça o amb el límit del tauler, bé siga en direcció dels punts cardinals (nord, sud, est, oest) o formant diagonals. En aquest grup trobem a les torres, els alfils i la reina.
- Comportament no vectorial: Aquestes peces tenen un rang de moviment definit per desplaçaments fixes o "offsets" partint de la posició en la que es troben, per exemple el rei es pot moure utilitzant els desplaçaments [-9, -8, -7, -1, 1, 7, 8, 9]. En aquest grup trobem els cavalls, els peons i el rei.

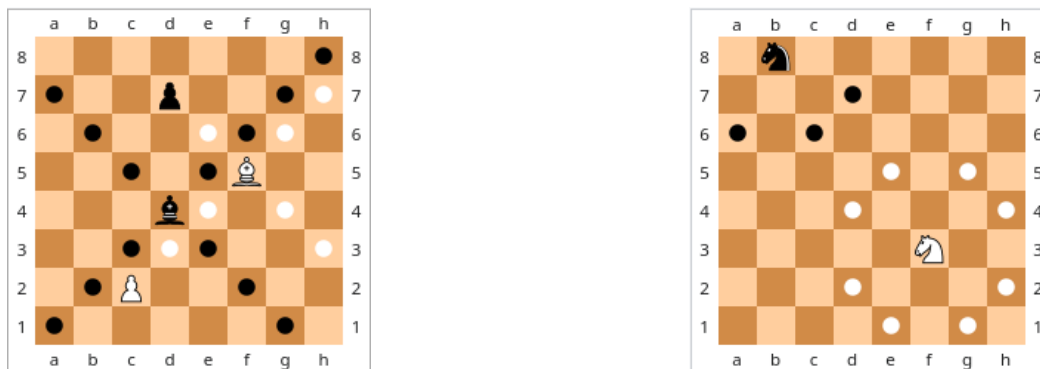


Figura 3.5: Exemples de desplaçament vectorial i no vectorial

Tenint el compte el tipus de peça, es comprovarà el moviment, si és il·legal, no es podrà executar i es tornara un estatus d'error, però si és vàlid, es crearà un tauler "imaginari" a partir del que s'està analitzant, i en aquest s'executarà el moviment, d'aquesta manera podrem avaluar les conseqüències de la jugada sense alterar l'estat del tauler original, creant un mecanisme de "rollback". Després de dur a terme un moviment, poden passar tres coses:

1. El moviment causa escac al rei contrari.
2. El moviment causa escac i mat a l'oponent i per tant el fi de la partida.
3. El moviment exposa al rei del jugador a escac, per tant el moviment s'ha de marcar com no vàlid.

Per a saber si el rei està en escac, s'analitza cadascun dels moviments possibles de cadascuna de les peces de l'oponent, i es busquen aquells que tinguin com a casella de destinació la casella en la que està el rei.

Per a calcular l'escac i mat, s'ha de complir, a més, que el rei no tinga cap moviment vàlid disponible, però també s'ha de comprovar si alguna de les peces pot cobrir el atac al rei, novament s'utilitza la mecànica del tauler "imaginari" per a computar aquests casos.

3.3.5 Implementació d'una interfície gràfica d'usuari

Es poden diferenciar tres tipus d'elements principals, que componen les distintes pàgines; els formularis, les taules i el tauler principal.

Per a les pàgines d'inici de sessió i registre, s'han fet servir formularis als quals s'ha modificat l'acció que fan quan s'envien (SubmitEvent) mitjançant un oient d'esdeveniments (EventListener), de manera que s'executa una funció personalitzada que transforma les dades introduïdes a format JSON i després utilitza l'API Fetch per a fer un POST al endpoint corresponent.

Després es processa la resposta del servidor i es mostra un missatge de benvinguda o d'error.

Figura 3.6: Captures dels formularis d'inici de sessió i registre

Quan l'usuari es valida al formulari d'inici de sessió, es mostrarà la pàgina principal, que compta amb un tauler d'escacs i dues taules, una que mostra els moviments que es realitzen a la partida i altra que mostra els jugadors disponibles. D'aquesta pàgina és important destacar alguns aspectes, per exemple:

- A la zona dreta de la barra superior es mostra el nom del usuari amb el que s'ha iniciat sessió.
- Al seleccionar una peça aliada al tauler, es pinten els moviments vàlids, així com els moviments d'atac, però no es mostrarà si es selecciona una peça del oponent. Açò s'aconsegueix mitjançant una consulta al servidor.
- Al llistat de moviments, es mostra el valor del moviment i la icona que correspon a la peça desplaçada, a més mostra els noms dels jugadors que participen a la partida.
- El llistat de jugadors es tracta d'una simplificació del tauler complet que es troba a la pàgina "Llistat de jugadors", també es pot accedir a aquest fent clic al botó blau que conté la icona de una lupa.
- Les alertes o missatges emergents es mostraran dalt de la barra superior. Es pot veure un exemple d'aquesta funcionalitat a la figura d'escac i mat que es troba a la secció "Resultat i conclusions finals".

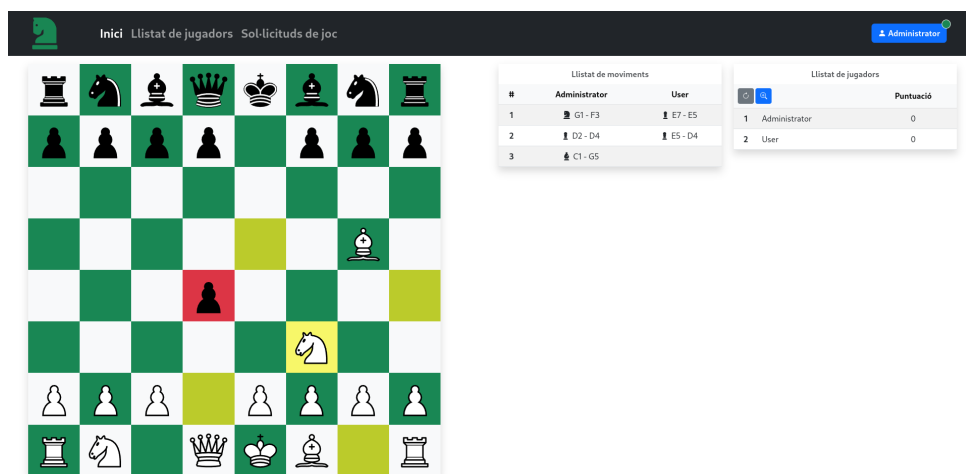


Figura 3.7: Captura de la pàgina principal

Tant la pàgina "Llistat de jugadors" com la de "Sol·licituds de joc" compten amb taulers similars, ambdós estan subscrits a un flux de dades, per tant, es recarreguen de forma automàtica. També compten amb un filtre per a facilitar la cerca.

Llistat de jugadors

	Buscar jugadors...		Puntuació	Accions
1	Administrator		0	[+]
2	User		0	[+]
3	Victor		0	[+]

Sol·licituds de joc

	Buscar sol·licituds...		Puntuació	Enviada a les	Accions
1	Victor		0	16:38	[✓] [✗]
2	Administrator		0	16:03	[✓] [✗]

Figura 3.8: Captures dels taulers de jugadors i sol·licituds

3.3.6 Persistència de dades rellevants a les partides

La utilització d'un ORM simplifica molt aquest apartat, ja que no s'han descriure consultes SQL ni fer operacions de baix nivell de forma explícita. S'han definit diferents classes amb l'estereotip de repositori (anotació `@Repository`) que proporciona Spring Boot. Es poden observar al diagrama de classes de l'API a la secció "estructura de classes".

3.4 Proves

Per a provar la part de la API s'han executat proves de tipus funcional mitjançant les ferramentes de depuració de Postman. Aquestes proves es centren en comprovar si el sistema compleix els requisits funcionals establerts, verificant que les funcionalitats del sistema funcionen com s'espera i produeixen els resultats correctes.

Postman és una plataforma de col·laboració i desenvolupament d'API que permet als desenvolupadors provar, depurar i documentar les seves API de manera eficient. És una eina molt utilitzada per enviar sol·licituds HTTP a les API, comprovar el seu funcionament i generar documentació interactiva.

El mòdul del motor d'escacs s'ha provat utilitzant proves d'unitat amb la ferramenta JUnit5. Aquestes proves es centren en comprovar el correcte funcionament de les unitats individuals del sistema. Es prova cada mòdul o component per separat per assegurar-se que funciona com s'espera i compleix les especificacions.

JUnit és un marc de proves unitàries per a Java que permet als desenvolupadors comprovar el comportament i la funcionalitat de les seves classes i mètodes de manera automatitzada. Proporciona anotacions i assertions per definir casos de prova i verificar els resultats esperats. JUnit és àmpliament utilitzat per assegurar la qualitat del codi i detectar errors.

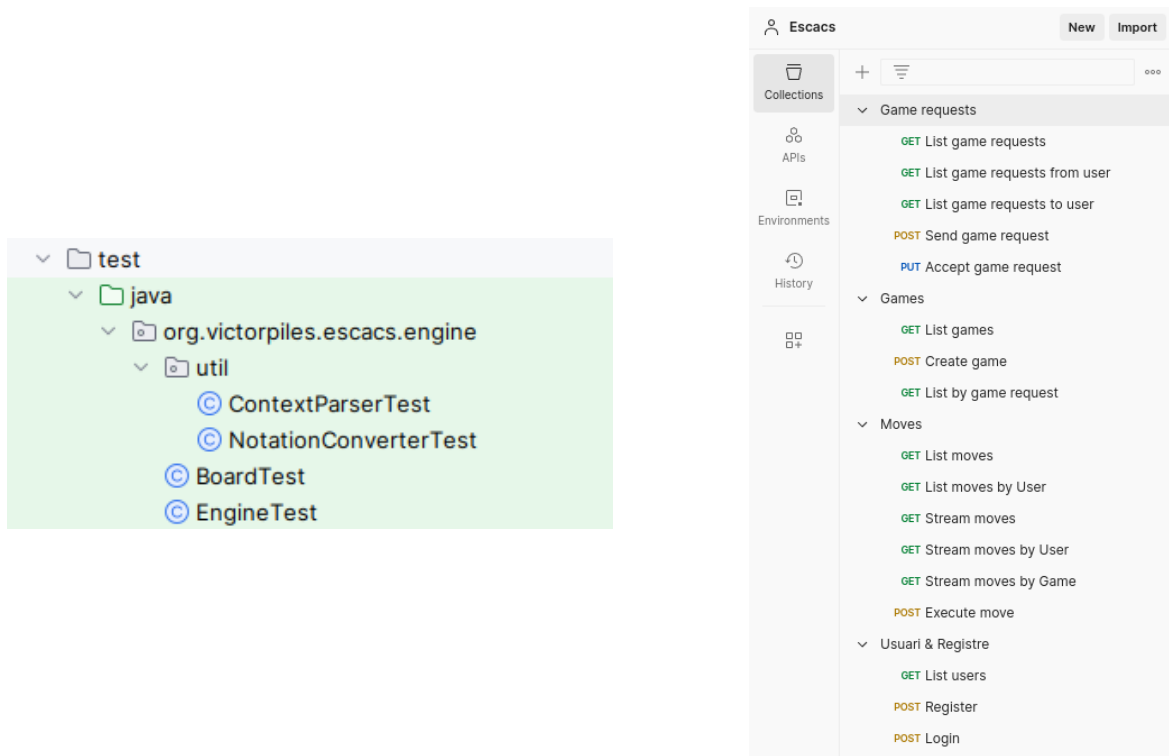


Figura 3.9: Tests JUnit i Postman

3.5 Implantació i configuració

En aquesta secció s'abordaran els passos necessaris per a la instal·lació, configuració i posada en marxa del software perquè pugui ser utilitzat a un entorn real. S'ha de tindre en compte que la forma de desplegar del servidor serà diferent a la del client, ja que utilitzen tecnologies diferents.

3.5.1 Implantació del servidor: Maven Deploy i GitHub Packages

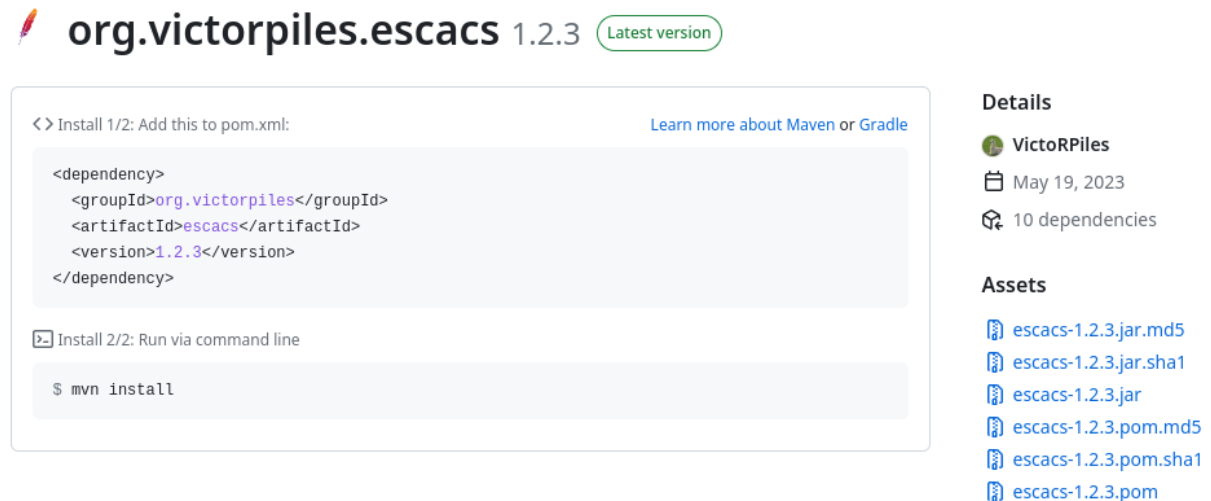
El cicle de vida del codi del servidor s'ha gestionat amb Maven, aquesta és una eina utilitzada per construir i gestionar projectes de software, principalment en el llenguatge Java. Ajuda els desenvolupadors a gestionar les dependències del projecte (biblioteques externes i frameworks en què es basa el projecte) i a automatitzar el procés de construcció. També proporciona una estructura estàndard per organitzar el codi font i els recursos dels projectes.

Per a crear una nova versió en format JAR del projecte, s'ha configurat una acció de GitHub. Les accions de GitHub són una característica que permet automatitzar els fluxos de treball de desenvolupament en la plataforma GitHub. Permet configurar tasques automatitzades que s'executin en resposta a esdeveniments específics del repositori. Aquestes accions es defineixen en fitxers de configuració i poden incloure passos com la compilació, les proves i el desplegament del codi. Utilitzar les accions de GitHub millora l'eficiència, la qualitat del codi i facilita la col·laboració amb altres desenvolupadors.

En concret s'ha configurat una acció que, quan es publica una nova versió del programa a la secció "releases" del [repositori de GitHub](#), executa el comandament "maven:deploy" i després

utilitza la configuració del pom.xml per a pujar el paquet al núvol de Maven i annexar-lo al repositori.

Per tant, per a utilitzar el servidor s'ha de descarregar el paquet al [repositori de GitHub](#) i després simplement executar el JAR.



org.victorpiles.escacs 1.2.3 Latest version

< > Install 1/2: Add this to pom.xml: [Learn more about Maven or Gradle](#)

```
<dependency>
  <groupId>org.victorpiles</groupId>
  <artifactId>escacs</artifactId>
  <version>1.2.3</version>
</dependency>
```

Install 2/2: Run via command line

```
$ mvn install
```

Details

- VictorPiles**
- May 19, 2023
- 10 dependencies

Assets

- escacs-1.2.3.jar.md5
- escacs-1.2.3.jar.sha1
- escacs-1.2.3.jar
- escacs-1.2.3.pom.md5
- escacs-1.2.3.pom.sha1
- escacs-1.2.3.pom

Figura 3.10: Paquet del servidor al [repositori de GitHub](#)

3.5.2 Implantació del client: Node Package Manager i Electron Forge

Per a la gestió de dependències i cicle de vida del codi al client, s'ha utilitzat el sistema de gestió de paquets de NodeJS, anomenat npm (Node Package Manager), que ofereix una àmplia varietat de mòduls i llibreries predefinites per a la construcció d'aplicacions web.

Per a empaquetar l'aplicació com un executable, s'ha utilitzat Electron Forge, aquesta és una eina per empaquetar i distribuir aplicacions Electron. Combina molts paquets d'ús específic per crear un flux de construcció que funciona amb poca configuració, inclou signatura de codi, instal·ladors i publicació d'artefactes. Per a fluxos de treball avançats, es pot afegir lògica de construcció personalitzada en el cicle de vida de Forge mitjançant la seva API de connectors. Les destinacions personalitzades de construcció i emmagatzematge es poden gestionar configurant els "Makers" i "Publishers".

Per a crear un instal·lador, específic per al sistema operatiu amfitrió, s'ha d'executar el comandament "npm run make", després s'ha d'instal·lar (doble clic en Windows o utilitzant el gestor de paquets a Linux i MacOS).



Figura 3.11: Client empaquetat per a Debian i Fedora Linux

4 Resultat i conclusions finals

En aquesta secció s'exposaran les conseqüències del desenvolupament del projecte, així com les possibles actualitzacions que es podrien realitzar a aquest, tant a nivell tècnic i com des de un punt de vista subjectiu.

4.1 Avaluació i aprenentatge

4.1.1 Problemes i obstacles

El desenvolupament d'aquest projecte no ha segut exactament lineal, ja que s'han provat i rebutjat una gran varietat de possibles solucions i tecnologies abans d'aplegar al resultat final.

El principal obstacle a l'hora de dur a terme el projecte ha segut l'obsolescència i complexitat de les alternatives que ofereix Java per a realitzar interfícies gràfiques. Les llibreries Swing i JavaFX tenen un funcionament i estil visual antiquat, encara que observem moltes aplicacions modernes construïdes amb aquestes ferramentes, tindre un bon resultat amb els coneixements i recursos dels que es disposen per a aquest projecte era una tasca molt complicada.

La solució aparentment més senzilla era canviar de llenguatge de programació a algun web i utilitzar HTML i CSS per a crear una interfície atractiva i més escalable. Aquesta idea es rebutja després de fer diverses proves per dos motius:

1. La fragilitat i rendiment dels llenguatges web, ja que no compten amb la robustesa que ofereix un compilador, ni altres característiques com la programació orientada a objectes o un fort sistema de tipus, que (en al meua opinió) són molt beneficioses quan s'escriu software. Aquests motius es troben darrere de la creixent popularitat del llenguatge TypeScript i de l'elecció d'aquest per a codificar el frontend.
2. La voluntat d'utilitzar tecnologies pròpies del cicle de desenvolupament d'aplicacions multiplataforma i la intenció de realitzar un projecte que utilitzara els conceptes que s'han après a classe, tractant de diferenciar-lo del que (en al meua opinió) més bé podria ser un projecte del cicle de desenvolupament d'aplicacions web .

Després de formular la pregunta: "Açò com es fa al *món real*?" i investigar aquest aspecte, sorgeix la idea de utilitzar el model d'API RESTful, ja que permet combinar el millor de dos mons, la robustesa i fiabilitat dels llenguatges compilats (Java en aquest cas) i la qualitat de les interfícies gràfiques que ofereix la web.

4.1.2 Aprenentatge i conceptes nous

Per a dur a terme aquest projecte s'han utilitzat tecnologies ja vistes a classe, però també algunes noves, per això, ha segut necessari el aprenentatge autodidacta de ferramentes com

Spring Boot, Bootstrap, TypeScript, el protocol HTTP, programació reactiva, programació funcional, GitHub avançat o \LaTeX , entre altres.

4.2 Proposta de millores

Encara que el projecte ha complert el seu objectiu principal de desenvolupar una versió digital i multijugador del joc d'escacs, encara hi ha alguns aspectes que queden per implementar o que tenen marge de millora.

Han quedat pendents de desenvolupar algunes funcionalitat pròpies del joc d'escacs, com el rellotge, o alguns moviments especials de peces, per exemple l'enroc o la promoció de peons.

Des del punt de vista de l'emparellament multijugador, es podria dissenyar un sistema de cues, per a crear partides amb jugadors aleatoris que també estiguen a l'espera.

També queda sense resoldre el mode un jugador i, per tant, la introducció d'una intel·ligència artificial capaç de entendre les normes del joc i realitzar moviments al tauler.

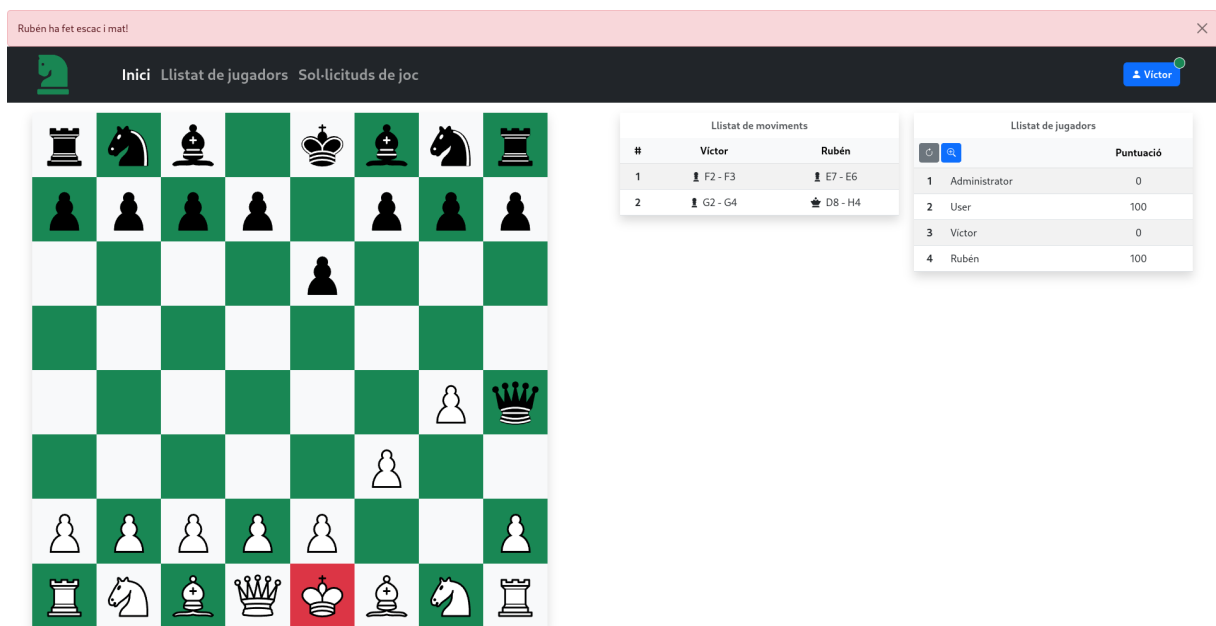


Figura 4.1: Escac i mat

Referències

- [1] Baeldung. Learn spring boot.
- [2] Bootstrap. Bootstrap docs.
- [3] GitHub. Dependabot.
- [4] GitHub. Java ci with maven.
- [5] GitHub. Maven package.
- [6] GitHub. Node.js ci.
- [7] Jackson. Processador json d'alt rendiment per a java.
- [8] Java. Funcions criptogràfiques.
- [9] Junit. Proves d'unitat en java.
- [10] Lombok. Lombok docs.
- [11] Mozilla. Using the fetch api.
- [12] Overleaf. Documentation.
- [13] Overleaf. Learn latex in 30 minutes.
- [14] Reactor. Programació reactiva.
- [15] Spring. Spring boot docs.
- [16] Wikipedia. Bishop.
- [17] Wikipedia. En passant.
- [18] Wikipedia. King.
- [19] Wikipedia. Knight.
- [20] Wikipedia. Pawn.
- [21] Wikipedia. Pieces.
- [22] Wikipedia. Queen.
- [23] Wikipedia. Rook.