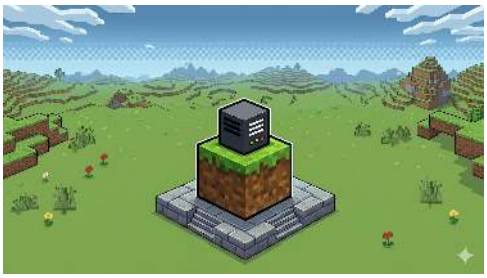
Institut Puig Castellar
CFGS Desenvolupament d'Aplicacions Web · Grup B



SafuHost

Plataforma web d'autogestió de servidors de Minecraft

Autor: **Izan Santiago Fuentes**
Tutor: David Delgado
Curs 2025–2026

Crèdit de Síntesi · Projecte intermodular

Índex:

Resum del projecte

SafuHost és una plataforma web d'autogestió que permet a qualsevol usuari crear i administrar servidors de Minecraft des del navegador, sense necessitat de coneixements avançats d'administració de sistemes. El sistema automatitza el desplegament de contenidors Docker mitjançant la imatge oficial itzg/minecraft-server, assigna ports de forma dinàmica i registra cada servidor en una base de dades SQLite. El backend, desenvolupat en Java amb Spring Boot, exposa una API REST i un canal WebSocket que actuen de pont entre la interfície web i el motor de Docker.

A més del cicle de vida bàsic dels servidors, la plataforma incorpora una consola interactiva accessible des del navegador en temps real, gestió de mods integrada amb l'API pública de Modrinth amb resolució automàtica de dependències, gestió de llista blanca en calent i obertura automàtica de ports al router mitjançant UPnP. El sistema d'autenticació, basat en JWT i BCrypt, garanteix que cada usuari només pugui gestionar els seus propis servidors. El frontend és una aplicació d'una sola pàgina (SPA) desenvolupada en Vue 3.

Paraules clau: Docker · Spring Boot · JWT · BCrypt · Minecraft · API REST · WebSocket · SQLite · Vue 3 · Modrinth · UPnP · Contenidors

Abstract

SafuHost is a self-managed web platform that allows any user to create and manage Minecraft servers directly from the browser, without requiring advanced system administration knowledge. The system automates the deployment of Docker containers using the official itzg/minecraft-server image, dynamically assigns ports, and registers each server in a SQLite database. The backend, developed in Java with Spring Boot, exposes a REST API and a WebSocket channel that act as a bridge between the web interface and the Docker engine.

Beyond the basic server lifecycle, the platform includes a real-time interactive console accessible from the browser, integrated mod management through the public Modrinth API with automatic dependency resolution, live whitelist management, and automatic port forwarding on the router via UPnP. The authentication system, based on JWT and BCrypt, ensures that each user can only manage their own servers. The frontend is a Single Page Application (SPA) built with Vue 3.

Keywords: Docker · Spring Boot · JWT · BCrypt · Minecraft · REST API · WebSocket · SQLite · Vue 3 · Modrinth · UPnP · Containers

1. Presentació del projecte

1.1 Introducció

El present projecte té com a finalitat el disseny i desenvolupament d'una plataforma web d'autogestió de servidors de Minecraft anomenada SafuHost. L'objectiu principal és oferir a qualsevol usuari una eina senzilla amb la qual pugui crear i administrar els seus propis servidors mitjançant una interfície web, sense necessitat d'accedir directament al sistema operatiu ni de disposar de coneixements avançats d'administració de sistemes.

A través de la plataforma, l'usuari pot dur a terme les accions següents:

- Crear servidors de Minecraft seleccionant la versió desitjada, el tipus (Vanilla, Forge o Fabric), la dificultat, el mode de joc i altres paràmetres de configuració.
- Iniciar, aturar i eliminar cada servidor de forma independent.
- Consultar l'estat i les dades de connexió (adreça IP i port) de cada servidor.
- Accedir a la consola del servidor en temps real des del navegador i enviar-hi comandes.
- Gestionar la llista blanca de jugadors autoritzats sense necessitat de reiniciar el servidor.
- Cercar i instal·lar mods automàticament des de l'API de Modrinth, amb resolució automàtica de dependències.
- Gestionar múltiples servidors de forma simultània, cadascun aïllat en el seu propi contenidor.
- Registrar-se i autenticar-se de forma segura, de manera que cada usuari només pugui veure i gestionar els servidors dels quals és propietari.
- Obrir automàticament els ports necessaris al router mitjançant UPnP, permetent que qualsevol jugador es connecti des d'internet.

A nivell tècnic, el sistema s'organitza com una arquitectura client-servidor de tres capes: una interfície web desenvolupada en Vue 3 que es comunica amb un backend en Java (Spring Boot), el qual gestiona la lògica de negoci, la persistència de dades en SQLite i l'execució dels servidors de Minecraft a través de contenidors Docker.

1.2 Context

Minecraft és, a dia d'avui, un dels videojocs més venuts de la història, amb més de 300 milions de còpies venudes en totes les seves plataformes. Més enllà de les xifres, la seva longevitat és el que el fa realment singular: porta actiu des del 2009 i continua sent un dels jocs més jugats del món, amb una comunitat activa que crea contingut, organitza esdeveniments i manté servidors multijugador propis.

Les plataformes de hosting de servidors existents presenten limitacions clares per als usuaris sense coneixements tècnics:

- Cost elevat: la majoria de serveis de pagament oscil·len entre 5 € i 20 € al mes per un servidor bàsic.
- Publicitat agressiva: les alternatives gratuïtes es financen amb publicitat, cosa que es tradueix en llargues cues d'espera per arrencar el servidor.
- Interfícies complexes: molts panells de control estan dissenyats pensant en administradors de sistemes, no en jugadors.
- Restriccions en l'ús de mods: molts serveis limiten o no permeten la instal·lació de mods.

En aquest context es detecta una oportunitat clara: una plataforma que posi l'usuari al centre, amb una interfície senzilla i intuïtiva, sense tecnicismes innecessaris i sense publicitat. SafuHost neix com a resposta a aquesta necessitat.

1.3 Justificació

L'interès per aquest projecte té un origen personal. Minecraft va ser un dels jocs més importants de la infància de l'autor, i amb el temps va despertar una curiositat creixent per entendre com funcionaven les coses per dins: els servidors, les connexions, els mods i la infraestructura que feia possible que desenes de persones juguessin juntes en un mateix món.

Al llarg dels anys s'han experimentat de primera mà les limitacions dels serveis de hosting disponibles. Aquesta experiència, unida als coneixements adquirits durant el cicle formatiu, va plantejar una pregunta: per què no existeix una plataforma que cobreixi les necessitats bàsiques de qualsevol jugador que vulgui muntar un servidor, i que al mateix temps ofereixi les opcions més avançades de forma senzilla?

SafuHost neix amb aquest objectiu. Cobrir les necessitats bàsiques i apropar a l'usuari comú opcions que avui en dia només estan a l'abast de perfils tècnics: instal·lar mods, gestionar una llista blanca, accedir a la consola del servidor en temps real o controlar la versió exacta que s'està executant.

Des del punt de vista formatiu, el projecte integra competències de diverses matèries del cicle: bases de dades, programació orientada a objectes, serveis de xarxa, sistemes operatius i disseny web.

1.4 Objectius

Objectiu general

Dissenyar i implementar una plataforma web que permeti a usuaris sense coneixements avançats d'administració de sistemes crear, configurar i gestionar servidors de Minecraft mitjançant una interfície web, de forma que els servidors estiguin disponibles en remot perquè qualsevol jugador pugui connectar-s'hi des de qualsevol lloc.

Objectius específics

- Implementar un backend en Java amb Spring Boot que exposi una API REST completa.
- Integrar la llibreria docker-java per automatitzar el cicle de vida dels contenidors.
- Dissenyar un model de dades amb SQLite que registri servidors, ports i identificadors.
- Implementar assignació dinàmica de ports amb doble comprobació (base de dades + sistema operatiu).
- Implementar comunicació en temps real via WebSockets per a la consola.
- Integrar l'API de Modrinth amb resolució automàtica de dependències.
- Permetre gestió de la llista blanca en calent sense reiniciar el servidor.
- Implementar autenticació JWT amb contrasenyes xifrades amb BCrypt.
- Implementar obertura automàtica de ports al router via UPnP.
- Aplicar el Principi de Responsabilitat Única (SRP) de SOLID.
- Desenvolupar una interfície web en Vue 3 intuïtiva.
- Documentar el procés complet.

2. Estratègia i planificació

2.1 Estratègia de desenvolupament i viabilitat

Per abordar el projecte es van considerar dues estratègies principals: adaptar un panell d'administració existent, com ara Pterodactyl o Crafty Controller, o construir una solució completament nova des de zero. Es va optar per la segona opció perquè permet tenir control complet sobre l'arquitectura, el codi i el model de dades, així com demostrar de manera clara les competències tècniques adquirides.

Pel que fa a la viabilitat:

- Viabilitat tècnica: totes les tecnologies utilitzades són de codi obert, ben documentades i amb una comunitat activa.
- Viabilitat temporal: planificació flexible dividida en fases ben definides, compatible amb el calendari del cicle.
- Viabilitat econòmica: totes les eines són gratuïtes i de codi obert, sense cost en llicències.

2.2 Estudi econòmic i pressupostari

El desenvolupament de SafuHost ha prioritzat l'ús de tecnologies de codi obert i eines gratuïtes per minimitzar el cost econòmic. No obstant això, per dotar el projecte d'un enfocament realista i professional, s'ha elaborat un pressupost simulat que contempla tant els recursos materials com el cost de la mà d'obra.

Costos d'infraestructura, programari i maquinari

Concepte	Tipus	Cost estimat
IntelliJ IDEA / Visual Studio Code	IDE de desenvolupament	0,00 €
Git i GitHub	Control de versions	0,00 €
Java 21 (Spring Boot)	Entorn servidor	0,00 €
Vue 3 / Vite	Framework frontend	0,00 €
SQLite	Base de dades	0,00 €
Docker Desktop	Motor de contenidors	0,00 €
Postman	Eina de proves	0,00 €
Estació de treball (PC)	Maquinari	800,00 €
TOTAL INFRAESTRUCTURA		**800,00 €**

Costos de personal (mà d'obra)

Assumint el perfil d'un desenvolupador Junior Full-Stack treballant en el projecte durant un període de 2,5 mesos (del 10 de març al 15 de maig de 2025), amb un salari estimat segons les tarifes de portals d'ocupació a Espanya per a perfils júnior:

Perfil professional	Salari mensual	Mesos	Cost total
Desenvolupador Junior Full-Stack	1.400,00 €	2,5	3.500,00 €

Perfil professional	Salari mensual	Mesos	Cost total
TOTAL PERSONAL			**3.500,00 €**

Pressupost total del projecte

Sumant els recursos tècnics (800 €) i la dedicació del desenvolupador (3.500 €), el cost total estimat per a l'execució del projecte ascendeix a 4.300 €. S'estima que els costos de manteniment i actualització de la plataforma tindrien un valor d'entre 200 € i 400 € anuals un cop finalitzat el desenvolupament.

2.3 Metodologia de treball

La metodologia seguida ha estat àgil i iterativa. En lloc de completar cada fase al 100% abans de passar a la següent, s'ha avançat de forma incremental, afegint funcionalitats una a una i verificant que cada una funcionava correctament abans de continuar. Per mantenir el fil del treball entre sessions, s'han seguit dues pràctiques fonamentals: el control de versions amb Git i l'ús de comentaris detallats al codi font.

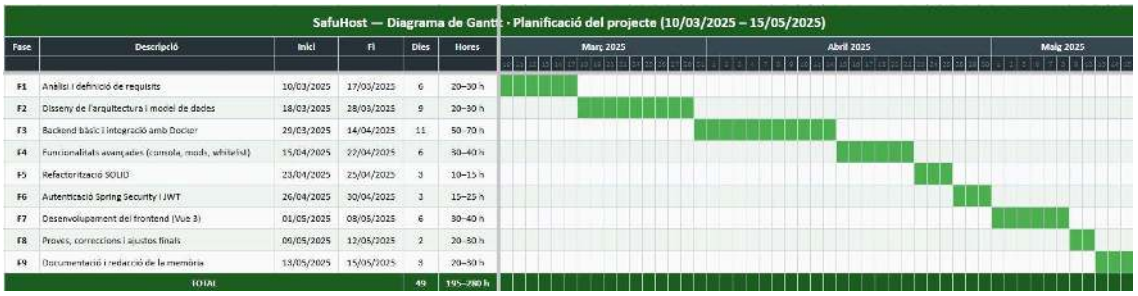
Des del punt de vista de l'arquitectura, s'ha aplicat el patró en capes propi de Spring Boot (Controlador – Servei – Repositori) i s'ha realitzat una refactorització aplicant el Principi de Responsabilitat Única (SRP) de SOLID.

2.4 Planificació

La planificació del projecte s'ha dividit en nou fases seqüencials, del 10 de març al 15 de maig de 2025:

Fase	Descripció	Inici	Fi	Dies	Hores
F1	Anàlisi i definició de requisits	10/03/2025	17/03/2025	6	20–30 h
F2	Disseny de l'arquitectura i model de dades	18/03/2025	28/03/2025	9	20–30 h
F3	Backend bàsic i integració amb Docker	29/03/2025	14/04/2025	11	50–70 h
F4	Funcionalitats avançades (consola, mods, whitelist)	15/04/2025	22/04/2025	6	30–40 h
F5	Refactorització SOLID	23/04/2025	25/04/2025	3	10–15 h
F6	Autenticació Spring Security i JWT	26/04/2025	30/04/2025	3	15–25 h
F7	Desenvolupament del frontend (Vue 3)	01/05/2025	08/05/2025	6	30–40 h
F8	Proves, correccions i ajustos finals	09/05/2025	12/05/2025	2	20–30 h
F9	Documentació i redacció de la memòria	13/05/2025	15/05/2025	3	20–30 h
TOTAL				**49**	**195–280 h**

Diagrama de Gantt del projecte:



SafuHost — Diagrama de Gantt · Planificació del projecte (10/03/2025 – 15/05/2025)

Fase	Descripció	Inici	Fi	Dies	Hores
F1	Anàlisi i definició de requisits	10/03/2025	17/03/2025	6	20–30 h
F2	Disseny de l'arquitectura i model de dades	18/03/2025	28/03/2025	9	20–30 h
F3	Backend bàsic i integració amb Docker	29/03/2025	14/04/2025	11	50–70 h
F4	Funcionalitats avançades (consola, mods, whitelist)	15/04/2025	22/04/2025	6	30–40 h
F5	Refactorització SOLID	23/04/2025	25/04/2025	3	10–15 h
F6	Autenticació Spring Security i JWT	26/04/2025	30/04/2025	3	15–25 h
F7	Desenvolupament del frontend (Vue 3)	01/05/2025	08/05/2025	6	30–40 h
F8	Proves, correccions i ajustos finals	09/05/2025	12/05/2025	2	20–30 h
F9	Documentació i redacció de la memòria	13/05/2025	15/05/2025	3	20–30 h
TOTAL				49	195–280 h

Març 2025 · Abril 2025 · Maig 2025

Figura 1: Diagrama de Gantt del projecte (10/03/2025 – 15/05/2025).

3. Anàlisi

3.1 Casos d'ús

A continuació es descriuen els principals casos d'ús del sistema. L'únic actor és l'usuari registrat, que interactua amb la plataforma a través de la interfície web.

Gestió de compte

- Registrar-se: l'usuari introdueix nom d'usuari, correu electrònic i contrasenya per crear un compte nou.
- Iniciar sessió: l'usuari introdueix les seves credencials i el sistema li retorna un token JWT vàlid durant 24 hores.

Gestió de servidors

- Crear un servidor: l'usuari especifica els paràmetres. El sistema assigna un port lliure, desplega un contenidor Docker i registra el servidor.
- Llistar, iniciar, aturar i eliminar servidors de forma independent.
- Consultar l'estat i les dades de connexió de cada servidor.

Consola

- Consultar la consola en temps real: l'usuari visualitza els registres del servidor via WebSocket.
- Enviar una comanda: el sistema la injecta directament al contenidor Docker.

Gestió de mods

- Cercar, instal·lar (amb resolució automàtica de dependències) i eliminar mods via l'API de Modrinth.

Gestió de la llista blanca

- Consultar, afegir i eliminar jugadors. Si el servidor està en línia, els canvis s'apliquen en calent.

3.2 Requisits funcionals

Gestió d'usuaris

- Registre amb nom d'usuari, correu i contrasenya xifrada amb BCrypt.
- Login retornant un token JWT vàlid durant 24 hores.
- Rebuig de registres amb noms d'usuari o correus ja existents.

Gestió de servidors

- Crear servidor especificant nom, versió, tipus, dificultat, mode de joc, PvP, mode premium, llista blanca, operadors i icona.
- Assignar automàticament un port lliure a partir del 25565, comprovant base de dades i sistema operatiu.
- Desplegar automàticament un contenidor Docker amb la imatge itzg/minecraft-server.
- Permetre iniciar, aturar, eliminar i actualitzar servidors.
- Garantir que cada usuari només pugui gestionar els servidors dels quals és propietari.
- Obrir el port al router via UPnP en iniciar, i tancar-lo en aturar.

Consola, mods i llista blanca

- Oferir un canal WebSocket per rebre els registres en temps real.
- Permetre cercar, verificar compatibilitat, instal·lar (amb dependències) i eliminar mods.
- Permetre gestionar la llista blanca en calent sense reiniciar el contenidor.

3.3 Requisits no funcionals

- Seguretat: BCrypt per al xifratge de contrasenyes, autenticació stateless amb JWT, missatges d'error genèrics i validació contra path traversal.
- Rendiment i concurrència: gestió de múltiples servidors simultanis i de connexions WebSocket simultànies amb ConcurrentHashMap.
- Mantenibilitat: arquitectura en capes i Principi de Responsabilitat Única.
- Usabilitat: interfície web accessible per a usuaris sense coneixements tècnics.
- Portabilitat: funcionament en entorn Windows amb Docker Desktop.

3.4 Anàlisi d'alternatives tecnològiques

Durant la fase d'anàlisi es van estudiar diferents alternatives per a cada component del sistema.

Motor de contenidors

Tecnologia	Avantatges	Inconvenients
Docker	Ecosistema gran, imatge oficial MC, API Java disponible	Requereix Docker Desktop a Windows
LXC/LXD	Lleuger, bon rendiment	Més complex, menor suport Windows
Màquines virtuals	Aïllament total	Molt pesades, inici lent
Procés Java directe	Sense dependències externes	Sense aïllament, gestió complexa

Elecció: Docker. Estàndard actual, imatge oficial específica per a Minecraft i docker-java per controlar des de Java.

Base de dades

Tecnologia	Avantatges	Inconvenients
SQLite	Sense servidor, fitxer únic, compatible JPA	No apte per a alta concurrència
PostgreSQL	Robust, preparat per a producció	Requereix servidor separat
MySQL/MariaDB	Molt estès, bon suport Spring	Requereix servidor separat
H2 en memòria	Fàcil de configurar	Les dades es perden en reiniciar

Elecció: SQLite. Sense servidor addicional i totalment compatible amb JPA/Hibernate.

Framework backend

Tecnologia	Avantatges	Inconvenients
Spring Boot (Java)	Arquitectura en capes, JPA, WebSocket natiu	Corba d'aprenentatge inicial
Node.js + Express	Lleuger, ràpid	Menys estructura, ORM menys madur
PHP	Conegut del cicle formatiu	Menys adequat per a contenidors

Elecció: Spring Boot. Arquitectura clara, suport natiu de WebSockets i disponibilitat de docker-java.

Framework frontend

Tecnologia	Avantatges	Inconvenients
Vue 3	Senzill, reactiu, Composition API	Ecosistema menor que React
React	Gran ecosistema	Major complexitat inicial
Angular	Molt estructurat, complet	Corba d'aprenentatge elevada

Elecció: Vue 3. Senzillesa, Composition API i Vite com a servidor de desenvolupament.

Tecnologies principals utilitzades al projecte:

Spring Boot: Un framework basat en Java que facilita enormement la creació d'aplicacions web i microserveis llestos per a producció. Et permet configurar el servidor de forma ràpida i gestionar les peticions HTTP (API) de manera eficient.

Java: És el llenguatge de programació robust, orientat a objectes i multiplataforma que serveix com a base per a construir tota la lògica de negoci del servidor.

Vue.js (Vue 3): Un framework de JavaScript progressiu utilitzat per a construir la interfície d'usuari. S'encarrega de fer que la web sigui interactiva, reactiva i visualment dinàmica per a l'usuari.

SQLite: Un motor de base de dades relacional lleuger, ràpid i que no requereix un servidor independent, ja que desa tota la informació directament en un únic fitxer local. Ideal per a projectes compactes o desenvolupaments àgils.

Docker: Una plataforma que permet empaquetar l'aplicació juntament amb totes les seves dependències en un "contenidor" aïllat. Això garanteix que el projecte funcioni exactament igual en el teu ordinador que en qualsevol altre servidor.

JWT (JSON Web Token): Un estàndard obert utilitzat per a transmetre informació de forma segura entre el client i el servidor com un objecte JSON. S'usa principalment per a l'autenticació d'usuaris (gestió d'inicis de sessió i sessions segures).

Modrinth: Una plataforma moderna i oberta per a allotjar contingut de Minecraft (com mods, plugins i datapacks). En el teu projecte, s'utilitza la seva API per a connectar, cercar o gestionar aquest tipus de recursos de forma automatitzada.

4. Disseny

4.1 Arquitectura del sistema

SafuHost segueix una arquitectura client-servidor de tres capes físiques:

- **Frontend (Vue 3 + Vite, port 5173):** SPA que s'executa al navegador. Es comunica amb el backend via HTTP/REST i WebSocket.
- **Backend (Spring Boot, port 3000):** gestiona tota la lògica de negoci, autenticació, comunicació amb Docker, Modrinth i UPnP.
- **Docker Engine (port 2375):** cada servidor de Minecraft s'executa en un contenidor Docker aïllat.

Origen	Destí	Protocol	Finalitat
Frontend	Backend	HTTP/REST (Axios)	Accions puntuals: login, llistar, crear, aturar…
Frontend	Backend	WebSocket	Consola en temps real
Backend	Docker	HTTP (docker-java)	Crear, aturar i eliminar contenidors
Backend	SQLite	JDBC	Persistència d'usuaris i servidors
Backend	Router	SSDP/UPnP	Obrir i tancar ports automàticament
Backend	Modrinth	HTTPS (RestTemplate)	Cercar i descarregar mods

4.2 Model de dades

El model de dades està format per dues entitats principals: Usuari i Servidor, definides com a entitats JPA. La relació entre ambdues és d'un a molts (1:N): un usuari pot tenir zero o més servidors, i cada servidor pertany exactament a un usuari.

Entitat Usuari

Camp	Tipus	Descripció
id	Long	Identificador únic autogenerat
username	String	Nom d'usuari, únic al sistema
email	String	Adreça de correu electrònic, única al sistema
password	String	Contrasenya xifrada amb BCrypt. Mai en text pla

Entitat Servidor

Camp	Tipus	Descripció
id	Long	Identificador únic autogenerat
nom	String	Nom del servidor, únic al sistema
port	Integer	Port assignat dinàmicament a partir del 25565
idContenidor	String	Identificador del contenidor Docker
estat	String	INICIANT, EN_LINIA, APAGAT o PAUSAT
versio	String	Versió de Minecraft (ex: 1.21.1)

Camp	Tipus	Descripció
tipus	String	VANILLA, FORGE o FABRIC
dificultat	String	peaceful, easy, normal o hard
modeJoc	String	survival, creative o adventure
pvp	Boolean	Combat entre jugadors
modeOnline	Boolean	Verificació de comptes de Mojang
usarWhitelist	Boolean	Llista blanca activada/desactivada
listaBlanca	String	Jugadors autoritzats separats per comes
propietari	Usuari	Referència a l'usuari propietari (@ManyToOne)

4.3 Disseny d'interfície

La interfície de SafuHost segueix una estètica inspirada en l'univers visual de Minecraft, amb una paleta de colors terrosos i verds, tipografia pixelada i un fons que recrea el bloc de gespa característic del joc. L'aplicació està formada per quatre vistes principals.

Pantalla d'inici de sessió

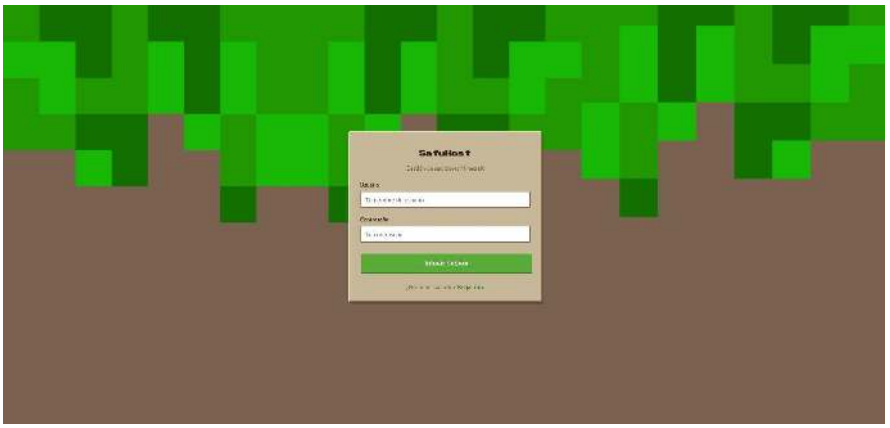


Figura 2: formulari d'inici de sessió.

Pantalla de registre

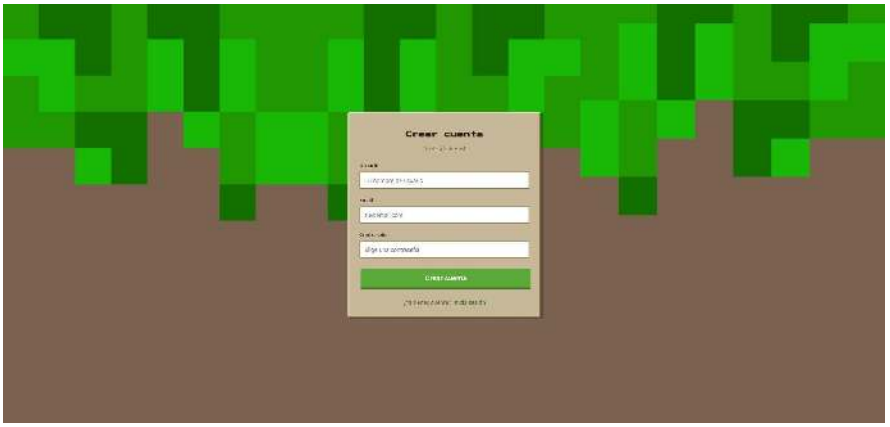


Figura 3: formulari de registre.

Pantalla de gestió de servidors

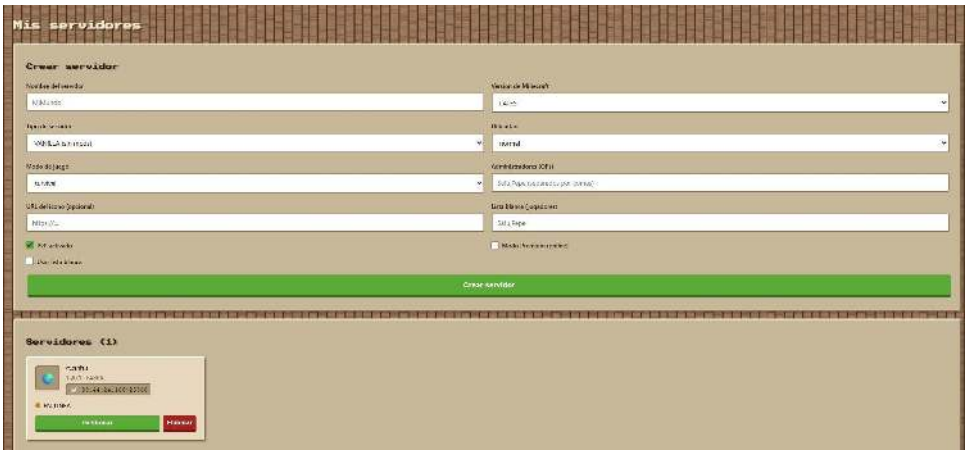


Figura 4: dividida en dues seccions: formulari de creació a la part superior i llistat de servidors a la part inferior.

Pantalla de detall del servidor

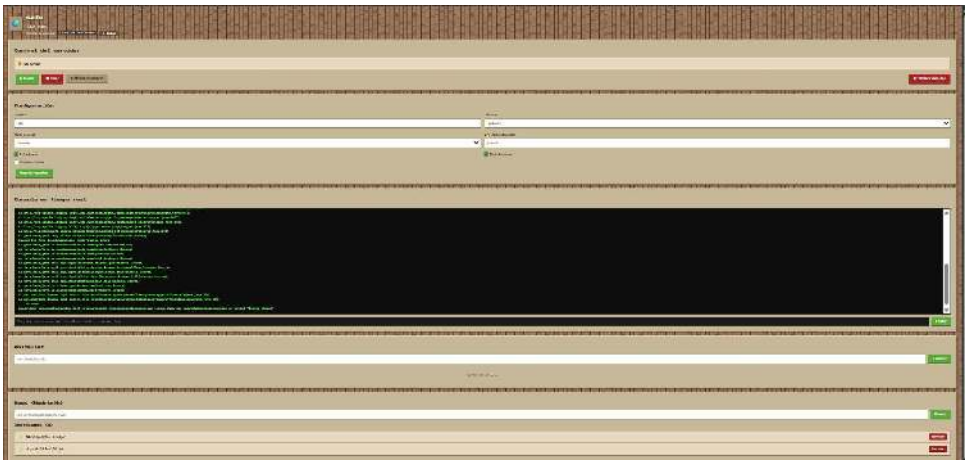


Figura 5: vista completa amb control, configuració, consola en temps real, whitelist i mods.

5. Desenvolupament

5.1 Estructura del projecte

El projecte està organitzat en dos mòduls principals: backend i frontend, cadascun amb la seva pròpia estructura de directoris. A continuació es descriu, per a cada capa del backend i del frontend, quina és la seva responsabilitat dins del sistema i com es relaciona amb la resta de components.

Capa de model (modelo)

El paquet modelo conté les dues classes que defineixen l'estructura de les dades del sistema. Servidor.java representa un servidor de Minecraft amb tots els seus atributs (nom, port, versió, tipus, dificultat, estat, etc.) i la seva relació amb l'usuari propietari. Usuario.java representa un compte d'usuari registrat a la plataforma, amb el nom d'usuari, el correu electrònic i la contrasenya xifrada. Ambdues classes estan anotades com a entitats JPA, cosa que significa que Hibernate genera automàticament les taules corresponents a la base de dades SQLite en el moment d'arrancar l'aplicació, sense necessitat d'escriure SQL manualment.

Capa de repositori (repositorio)

Els repositoris són interfícies que actuen com a pont entre la lògica de negoci i la base de dades. RepositorioServidor proporciona mètodes per guardar, llistar, buscar i eliminar servidors, a més de dos mètodes personalitzats: existsByPuerto, que comprova si un port ja està registrat (clau per evitar conflictes), i existsByNombre, que evita la creació de servidors amb noms duplicats. RepositorioUsuario ofereix les mateixes operacions bàsiques per als usuaris i inclou findByUsername, que permet localitzar un usuari pel seu nom durant el procés de login.

Capa de servei (servicio)

Aquesta és la capa més important del backend, on resideix tota la lògica de negoci. Després de la refactorització SOLID, està dividida en sis serveis especialitzats:

ServicioServidor actua com a coordinador central. Rep les peticions del controlador i les delega al servei corresponent. És l'únic servei amb el qual el controlador interactua directament. Les seves responsabilitats pròpies són: validar que el nom del servidor no estigui duplicat, orquestrar la seqüència completa de creació (demanar port → crear contenidor → guardar a la base de dades) i gestionar l'inici, l'aturada i l'eliminació de servidors.

ServicioPuertos s'encarrega exclusivament de trobar ports de xarxa lliures. Implementa la doble comprovació: primer consulta la base de dades per saber si el port ja està assignat a un altre servidor, i després comprova el sistema operatiu intentant obrir un ServerSocket en aquell port. Si ambdues comprovacions passen, el port es considera lliure. A més, gestiona l'obertura i el tancament automàtic de ports al router mitjançant el protocol UPnP.

ServicioDocker gestiona tota la comunicació amb el motor Docker. Les seves funcions principals són: consultar l'estat real d'un contenidor (en execució, aturat, etc.) i executar comandes dins d'un contenidor mitjançant mc-send-to-console. Aquesta última operació és la que permet tant l'enviament de comandes per la consola com els canvis de whitelist en calent.

ServicioMods s'encarrega de la integració amb l'API de Modrinth. Permet cercar mods per nom, verificar la seva compatibilitat amb el tipus de servidor (Forge o Fabric) i la versió de Minecraft, descarregar els arxius .jar dels mods i de les seves dependències automàticament, i eliminar mods instal·lats validant el nom del fitxer per prevenir atacs de path traversal.

ServicioWhitelist gestiona la llista blanca de jugadors autoritzats. Quan l'usuari afegeix o elimina un jugador, el servei actualitza el registre a la base de dades i, si el servidor està en línia, executa la comanda whitelist add o whitelist remove directament dins del contenidor Docker perquè el canvi s'apliqui immediatament sense reiniciar.

ServicioUsuario gestiona el registre i l'autenticació d'usuaris. En el registre, valida que el nom d'usuari i el correu no estiguin duplicats i xifra la contrasenya amb BCrypt abans de guardar-la. En el login, compara la contrasenya introduïda amb el hash emmagatzemat i, si coincideix, genera un token JWT signat que l'usuari utilitzarà per autenticar-se en totes les peticions posteriors.

Capa de controlador (controlador)

Els controladors són el punt d'entrada de l'API REST. ControladorServidor rep les peticions HTTP relacionades amb servidors (crear, llistar, iniciar, aturar, eliminar, consola, mods, whitelist) i les delega a ServicioServidor. ControladorUsuario gestiona les peticions de registre i login. Ambdós controladors no contenen lògica de negoci: la seva única responsabilitat és rebre la petició, cridar al servei corresponent i retornar la resposta en format JSON.

Capa de configuració (configuracion)

ConfiguracionDocker estableix la connexió amb el motor Docker mitjançant TCP al port 2375 i crea el DockerClient que tots els serveis utilitzen per comunicar-se amb Docker.

ConfiguracionWebSocket registra el handler de la consola en temps real, definint la ruta /ws/consola/{id} on el frontend es connectarà per rebre els registres del servidor.

ConfiguracionSeguridad configura Spring Security: defineix quins endpoints són públics (registre i login), quins requereixen autenticació (tots els altres), registra el filtre JWT que valida el token a cada petició i configura BCrypt com a algorisme de xifratge de contrasenyes.

Capa de seguretat (seguridad)

UtilJwt és la classe responsable de generar i validar els tokens JWT. En generar un token, hi inclou el nom d'usuari i una data d'expiració de 24 hores, i el signa amb una clau secreta HMAC-SHA. En validar un token, extreu el nom d'usuari i comprova que no hagi expirat.

FiltroJwt és un filtre que intercepta cada petició HTTP, extreu el token de la capçalera Authorization, el valida mitjançant UtilJwt i, si és correcte, registra l'usuari autenticat al SecurityContextHolder de Spring perquè la resta del sistema pugui saber qui està fent la petició.

Capa WebSocket (websocket)

ConsolaWebSocketHandler gestiona les connexions WebSocket de la consola en temps real. Quan un usuari obre la pantalla de detall d'un servidor, el frontend estableix una connexió WebSocket. El handler obre un stream de registres del contenidor Docker corresponent i reenvia cada nova línia al navegador en temps real. Per gestionar múltiples usuaris connectats simultàniament a diferents servidors, utilitza un ConcurrentHashMap que associa cada sessió WebSocket amb el seu stream de Docker.

Frontend (Vue 3 + Vite)

views/ conté les quatre vistes principals de l'aplicació: Login.vue (formulari d'inici de sessió), Registro.vue (formulari de registre), Servidores.vue (pantalla principal amb creació i llistat de servidors) i Servidor.vue (pantalla de detall amb control, consola, whitelist i mods).

components/ conté els components reutilitzables: Consola.vue (terminal en temps real amb connexió WebSocket), Whitelist.vue (gestió de la llista blanca) i Mods.vue (cerca i instal·lació de mods des de Modrinth).

api.js configura el client Axios amb un interceptor que adjunta automàticament el token JWT a la capçalera Authorization de cada petició. Si el token no existeix o ha expirat, redirigeix l'usuari a la pantalla de login.

router.js defineix les rutes de l'aplicació i les guardes de navegació: si un usuari no autenticat intenta accedir a una ruta protegida, el sistema el redirigeix automàticament a la pantalla d'inici de sessió.

5.1.2. Comunicació entre capes i flux d'una petició

Una de les característiques més importants de l'arquitectura de Spring Boot és que les capes no es coneixen entre elles directament: cada component declara quines dependències necessita mitjançant l'anotació @Autowired, i és Spring qui s'encarrega de crear les instàncies i connectar-les automàticament en el moment d'arrancar l'aplicació. Aquest mecanisme s'anomena injecció de dependències i és el que fa possible que el sistema funcioni de forma modular.

Les regles de comunicació entre capes són estrictes i unidireccionals:

- El controlador només parla amb el servei coordinador (ServicioServidor o ServicioUsuario). Mai accedeix directament a un repositori ni a Docker.
- El servei coordinador (ServicioServidor) delega en els serveis especialitzats quan la operació ho requereix.
- Els serveis especialitzats accedeixen directament al repositori per obtenir les dades del servidor que necessiten, i a les eines externes que els corresponen (DockerClient, RestTemplate per a Modrinth, etc.).
- Cap servei especialitzat crida a ServicioServidor: les dependències flueixen sempre en una sola direcció, de dalt a baix, per evitar dependències circulars.

Flux de creació d'un servidor

Quan l'usuari emplena el formulari de creació i prem el botó "Crear servidor", el frontend envia una petició POST a /api/servidores/crear amb totes les dades de configuració en format JSON. El ControladorServidor rep la petició, extreu les dades i crida a ServicioServidor.crearServidor(). ServicioServidor executa la seqüència de creació en un ordre molt concret: primer valida que el nom no estigui duplicat consultant RepositorioServidor.existsByNombre(); després demana a ServicioPuertos.encontrarPuertoLibre() que busqui un port disponible fent la doble comprovació (base de dades + sistema operatiu); a continuació construeix i llança un contenidor Docker amb totes les variables d'entorn configurades; finalment guarda el servidor a la base de dades mitjançant RepositorioServidor.save() i retorna les dades del servidor creat. El controlador empaqueta la resposta en format JSON i la retorna al frontend amb un codi 200 OK.

Flux d'enviament d'una comanda per consola

Quan l'usuari escriu una comanda al camp de text de la consola i prem "Enviar", el frontend envia una petició POST a /api/servidores/{id}/consola/comando. El ControladorServidor rep la petició i crida a ServicioServidor.enviarComandoConsola(id, comanda). ServicioServidor delega directament a ServicioDocker.enviarComandoConsola(id, comanda). ServicioDocker busca el servidor a la base de dades per obtenir el seu identificador de contenidor, construeix una comanda exec dins del contenidor Docker especificant l'usuari 1000 i executa mc-send-to-console amb la comanda proporcionada. El resultat de la comanda no es retorna per aquesta via: l'usuari el veu automàticament a la consola en temps real a través de la connexió WebSocket que ja té oberta.

Flux de la consola en temps real (WebSocket)

La consola funciona de forma diferent a la resta d'operacions perquè utilitza WebSocket en lloc d'HTTP. Quan l'usuari accedeix a la pantalla de detall d'un servidor, el frontend obre una connexió WebSocket a /ws/consola/{id}. ConsolaWebSocketHandler rep la connexió, busca el servidor a la base de dades per obtenir l'identificador del contenidor i obre un stream de registres del contenidor Docker mitjançant la crida logContainerCmd de docker-java.

A partir d'aquest moment, cada línia nova que el servidor de Minecraft escriu als seus registres arriba al handler, que la reenvia immediatament al navegador de l'usuari a través de la connexió WebSocket oberta. Si diversos usuaris estan connectats al mateix servidor, cadascun té la seva pròpia sessió WebSocket i el seu propi stream de Docker, gestionats mitjançant un ConcurrentHashMap. Quan l'usuari tanca la pantalla de detall o navega a una altra pàgina, el frontend tanca la connexió WebSocket i el handler tanca el stream de Docker corresponent per alliberar recursos.

5.2 Implementació de funcionalitats

Autenticació d'usuaris

El sistema d'autenticació es basa en JWT i BCrypt. En el registre, la contrasenya es xifra de forma irreversible amb BCrypt. En el login, el sistema compara la contrasenya introduïda amb el hash emmagatzemat mitjançant passwordEncoder.matches(), sense desxifrar-la. Si les credencials són correctes, el backend genera un token JWT signat amb HMAC-SHA, vàlid 24 hores. Cada petició protegida passa pel filtre FiltroJwt, que valida el token i registra l'usuari al SecurityContextHolder.

Creació i gestió de servidors

Quan l'usuari crea un servidor, el sistema valida que el nom no estigui duplicat, cerca el primer port lliure a partir del 25565 (doble comprobació: base de dades i sistema operatiu), desplega un contenidor Docker amb totes les variables de configuració, crea la carpeta de dades persistent i registra el servidor associant-lo al propietari. Les operacions d'inici i aturada actuen sobre el contenidor i obren o tanquen el port via UPnP.

Consola en temps real

La consola s'implementa via WebSocket. En obrir la pantalla de detall, el frontend estableix una connexió WebSocket amb el backend. El handler ConsolaWebSocketHandler obre un stream de registres del contenidor Docker i remet cada nova línia al navegador. Les connexions simultànies es gestionen amb un ConcurrentHashMap.

Gestió de mods

La integració amb Modrinth permet cercar mods, verificar compatibilitat amb el tipus de servidor i versió de Minecraft, i descarregar-los des del CDN. En instal·lar un mod, el sistema resol i instal·la automàticament les dependències. L'eliminació inclou validació contra path traversal.

Gestió de la llista blanca

La llista blanca es guarda com una cadena de noms separats per comes. En afegir o eliminar un jugador, el sistema actualitza la base de dades i, si el servidor està en línia, executa la comanda al contenidor via mc-send-to-console, aplicant el canvi en calent.

Refactorització aplicant SOLID

Durant el desenvolupament, ServicioServidor va créixer fins a superar les 400 línies barrejant cinc responsabilitats. Es va aplicar el SRP separant la lògica en ServicioPuertos, ServicioDocker, ServicioMods,

ServicioWhitelist i ServicioServidor com a coordinador. El controlador segueix interactuant exclusivament amb ServicioServidor, que actua com a façana del sistema.

5.3 Proves

Les proves s'han dut a terme de forma manual i progressiva. Durant les primeres fases es van verificar tots els endpoints de l'API REST mitjançant Postman, comprovant tant les respostes correctes com el comportament davant d'errors. Un cop disponibles les funcionalitats principals, es va desenvolupar una interfície web provisional sense disseny per provar el flux complet. Finalment, amb la interfície definitiva implementada, es va realitzar una validació general del conjunt del sistema.

5.4 Imprevistos i solucions

Al llarg del desenvolupament del projecte s'han trobat diversos problemes tècnics que no estaven previstos en la planificació inicial. Aquesta secció recull els més significatius, juntament amb les solucions adoptades, ja que reflecteixen el procés real de desenvolupament i justifiquen part del temps invertit en el projecte.

Connexió entre docker-java i Docker Desktop en Windows

Problema: en intentar crear el primer contenidor Docker des del codi Java, l'aplicació no podia connectar-se amb Docker Desktop. L'error indicava que no trobava el socket de comunicació. Això succeeix perquè en Linux Docker utilitza un socket UNIX, però en Windows amb Docker Desktop la comunicació es realitza a través d'un named pipe que docker-java no suporta de forma nativa.

Solució: es va configurar el DockerClient perquè utilitzés la connexió TCP localhost:2375, habilitant prèviament a Docker Desktop l'opció "Expose daemon on tcp://localhost:2375 without TLS". Es va afegir també la dependència docker-java-transport-httpclient5 al pom.xml com a client HTTP de transport.

EULA de Minecraft Server no acceptat automàticament

Problema: els contenidors Docker es creaven correctament, però el servidor de Minecraft no arrencava. En revisar els registres del contenidor, es va detectar que el servidor es detenia immediatament perquè l'EULA (End User License Agreement) de Minecraft no estava acceptat.

Solució: es va afegir la variable d'entorn EULA=TRUE a la configuració del contenidor en el moment de crear-lo des del codi Java. Amb això el servidor arrenca correctament sense intervenció manual.

Conflictes de ports al crear múltiples servidors

Problema: en intentar crear un segon servidor, el sistema intentava assignar-li el mateix port 25565 que ja estava utilitzant el primer. El sistema de detecció de ports lliures només comprovava el sistema operatiu, però no consultava la base de dades. Just després de crear el contenidor, el servidor triga un temps en arrencar; durant aquest temps el port està reservat a la base de dades però el sistema operatiu encara no el reporta com a ocupat.

Solució: es va modificar el mètode encontrarPuertoLibre() per fer una doble comprovació: primer consulta la base de dades (existsByPuerto) i només si no hi és comprova també el sistema operatiu. Un port només es considera lliure si passa ambdues comprovacions.

InspectContainerCmd fallava amb rutes de Windows

Problema: en implementar la consulta d'estat d'un contenidor, la crida inspectContainerCmd fallava sistemàticament en entorn Windows. docker-java intentava interpretar les rutes dels volums muntats (C:/mc-servers/...) com a volums amb format Linux, on el caràcter ":" actua com a separador.

Solució: es va substituir inspectContainerCmd per listContainersCmd amb withShowAll(true) i withIdFilter, que retorna la informació d'estat necessària sense intentar parsejar els volums i no es veu afectat per les rutes de Windows.

AUTOPAUSE bloquejava l'execució de comandes

Problema: inicialment els contenidors es creaven amb AUTOPAUSE=true, una característica de la imatge itzg que pausa la JVM del servidor quan no hi ha jugadors connectats. En implementar la gestió de whitelist en calent i l'enviament de comandes per consola, les proves fallaven: les comandes no s'aplicaven fins que un jugador es connectava. Amb la JVM pausada, mc-send-to-console no podia injectar comandes a la consola.

Solució: es va desactivar AUTOPAUSE durant el desenvolupament (AUTOPAUSE=false). Com a millora futura, es podria implementar una lògica que reactivés temporalment el contenidor abans d'injectar comandes i el tornés a pausar després.

Mc-send-to-console requeria variables específiques i un usuari concret

Problema: les primeres proves d'enviament de comandes a través de mc-send-to-console fallaven amb dos errors: missatges indicant que la "console pipe" no existia i errors de permisos al executar la comanda dins del contenidor.

Solució: es van identificar dos requisits no documentats de la imatge itzg: la variable d'entorn CREATE_CONSOLE_IN_PIPE=true ha d'estar activa al crear el contenidor perquè generi la pipe de consola, i les comandes exec han d'executar-se com l'usuari intern 1000, ja que la imatge executa el servidor amb aquest usuari i no amb root.

Creixement descontrolat de ServicioServidor (God Class)

Problema: tras incorporar les funcionalitats avançades, la classe ServicioServidor va superar les 400 línies barrejant cinc responsabilitats completament diferents: cicle de vida del servidor, cerca de ports, comunicació amb Docker, gestió de mods i gestió de whitelist. Qualsevol modificació obligava a comprendre l'arxiu sencer i augmentava el risc de trencar funcionalitats no relacionades.

Solució: es va aplicar el Principi de Responsabilitat Única (SRP de SOLID), separant la classe en cinc serveis especialitzats: ServicioPuertos, ServicioDocker, ServicioMods, ServicioWhitelist i ServicioServidor com a coordinador. La refactorització es va realitzar preservant tots els comentaris originals i sense modificar el ControladorServidor.

Spring Security bloquejava tots els endpoints

Problema: en afegir la dependència spring-boot-starter-security al pom.xml, tots els endpoints existents van començar a retornar respostes 401 Unauthorized, tot i que cap codi de seguretat s'havia escrit encara. Spring Security bloqueja automàticament tots els endpoints en quan s'afegeix al projecte.

Solució: es va crear una primera versió de ConfiguracionSeguridad.java que deixava tots els endpoints oberts amb permitAll(). Això va permetre treballar pas a pas: primer el registre, després el login amb JWT, i només en l'últim pas tancar els endpoints que requerien autenticació.

Servidors orfes en afegir la relació amb Usuari

Problema: en afegir el camp propietari a l'entitat Servidor amb @ManyToOne, tots els servidors creats durant les fases anteriors van deixar d'aparèixer a les consultes. Aquests servidors tenien el camp propietari_id com a NULL, ja que s'havien creat quan encara no existia la relació.

Solució: es va decidir començar de zero amb una base de dades neta, eliminant l'arxiu safuhosting.db, ja que es trobava en fase de desenvolupament i els servidors de prova no contenien dades importants. En un entorn de producció, hauria calgut planificar una migració de dades.

5.5 Disseny de la base de dades

La base de dades de SafuHost és intencionadament simple. Es va dissenyar pensant en el cas d'ús principal: una instal·lació en una màquina local on un o pocs usuaris gestionen els seus propis servidors. No cal la complexitat d'un motor de base de dades com PostgreSQL o MySQL quan els requisits de concurrència són baixos i la prioritat és la facilitat de desplegament.

El model de dades consta de dues entitats principals: Usuari i Servidor. La relació entre elles és senzilla: un usuari pot tenir molts servidors, i cada servidor pertany exactament a un usuari. Aquesta relació s'implementa a la base de dades com una clau forana propietari_id a la taula de servidors que apunta a la taula d'usuaris.

L'entitat Usuari

La taula d'usuaris emmagatzema la informació mínima necessària per a l'autenticació: un identificador numèric generat automàticament, un nom d'usuari únic que identifica el compte públicament, una adreça de correu electrònic única, i la contrasenya xifrada amb BCrypt. Cal remarcar que la contrasenya mai s'emmagatzema en text pla: el que es guarda a la base de dades és el hash resultant de l'algorisme BCrypt, un valor irreversible que no permet recuperar la contrasenya original.

El camp del correu electrònic s'utilitza durant el registre per garantir que no existeixin dos comptes amb la mateixa adreça, però en l'autenticació s'empra el nom d'usuari. Aquesta decisió es va prendre per simplicitat, ja que el nom d'usuari és més curt i fàcil de recordar per a l'usuari.

L'entitat Servidor

La taula de servidors és el cor de l'aplicació. Cada fila representa un servidor de Minecraft i conté tota la informació necessària per reconstruir l'estat del sistema en qualsevol moment: l'identificador numèric intern, el nom llegible que l'usuari va assignar al servidor, el port de xarxa assignat dinàmicament, l'identificador del contenidor Docker, el tipus de servidor (VANILLA, FORGE o FABRIC), la versió de Minecraft, la dificultat, el mode de joc, i diverses marques booleanes per a opcions com el PvP o la verificació de comptes premium.

A més dels camps de configuració, la taula emmagatzema la llista blanca com una cadena de text amb els noms separats per comes. Aquesta aproximació simplifica les consultes i les actualitzacions: en lloc de tenir una taula separada per als jugadors de la llista blanca, tota la informació queda continguda en un sol camp. Això és adequat per al volum d'usuaris que es preveu, on rarament una llista blanca superarà unes poques dotzenes d'entrades.

El camp de l'identificador del contenidor Docker és especialment important. Quan es crea un contenidor, Docker li assigna un hash hexadecimal llarg i únic. Aquest identificador és el que el sistema utilitza en totes les operacions posteriors: iniciar, aturar, consultar l'estat, llegir els registres o executar comandes. Sense ell, el backend no podria localitzar el contenidor corresponent a cada servidor.

Persistència amb JPA i SQLite

Spring Data JPA abstreu completament les operacions de base de dades. El desenvolupador no escriu SQL manualment: defineix mètodes als repositoris seguint una convenció de noms, i JPA els tradueix automàticament a consultes SQL. Per exemple, el mètode existsByNombre() es tradueix a una consulta SELECT COUNT que comprova si existeix alguna fila amb aquell nom, i findByPropietariUsername() genera un JOIN entre les dues taules per retornar tots els servidors d'un usuari concret.

La tria de SQLite com a motor de base de dades va requerir una configuració addicional respecte als motors més habituals. JPA i Hibernate no inclouen suport nadiu per a SQLite, de manera que va ser necessari afegir la dependència hibernate-community-dialects, que proporciona el dialecte específic que Hibernate necessita per generar SQL compatible amb SQLite.

5.6 El sistema d'autenticació en profunditat

L'autenticació és un dels aspectes més crítics de qualsevol aplicació web. Un sistema d'autenticació defectuós pot exposar dades sensibles de tots els usuaris. SafuHost implementa un sistema basat en estàndards moderns i ben provats: BCrypt per al xifratge de contrasenyes i JWT per a la gestió de sessions sense estat.

Per què BCrypt és la tria correcta per a contrasenyes

BCrypt és un algorisme de hash dissenyat específicament per a contrasenyes, a diferència d'algorismes com MD5 o SHA-256, que estan dissenyats per ser el més ràpids possible. La lentitud de BCrypt és intencionada i és precisament el que el fa segur: fa que un atac de força bruta o de diccionari sigui computacionalment molt costós.

A més de la lentitud, BCrypt incorpora automàticament un "salt": un valor aleatori que s'afegeix a la contrasenya abans de calcular el hash. Gràcies al salt, dues contrasenyes idèntiques generen hashes completament diferents, cosa que fa impossible els atacs de taules de rainbow precomputades. Quan l'usuari intenta autenticar-se, el sistema no desxifra el hash emmagatzemat (cosa que seria impossible), sinó que aplica BCrypt a la contrasenya introduïda i compara el resultat amb el hash guardat.

Com funciona JWT en detall

Un JSON Web Token és un estàndard (RFC 7519) per transmetre informació de forma segura entre dues parts com un objecte JSON. La seva estructura consta de tres parts separades per punts: la capçalera, el cos (payload) i la signatura. La capçalera identifica el tipus de token i l'algorisme de signatura. El cos conté les "claims": parells clau-valor amb informació com el nom d'usuari i la data d'expiració. La signatura garanteix que el token no ha estat alterat: es genera aplicant l'algorisme HMAC-SHA256 al contingut del token amb una clau secreta que només coneix el servidor.

Quan l'usuari inicia sessió correctament, el backend genera un token JWT i el retorna al frontend. A partir d'aquest moment, el frontend adjunta aquest token a totes les peticions HTTP en la capçalera Authorization amb el prefix "Bearer ". El backend, en rebre cada petició, extreu el token, verifica la signatura amb la clau secreta i, si és vàlid, permet el pas. Si el token ha estat manipulat o ha expirat, la verificació falla i la petició és rebutjada amb un codi 401.

La gran avantatge del sistema JWT és que és completament sense estat (stateless): el servidor no necessita emmagatzemar cap informació de sessió. Tota la informació necessària per autenticar l'usuari va codificada dins del token. Això simplifica enormement l'arquitectura, especialment en un sistema com SafuHost on podria haver múltiples instàncies del backend.

El filtre JWT i el flux de seguretat

Spring Security processa cada petició HTTP a través d'una cadena de filtres. El FiltroJwt s'insereix en aquesta cadena i s'executa abans que qualsevol controlador processi la petició. El seu funcionament és el següent: extreu el valor de la capçalera Authorization, comprova que comenci per "Bearer ", extreu el token, l'envia a UtilJwt per a la validació, i si és correcte, crea un objecte d'autenticació i el registra al SecurityContextHolder, que és el mecanisme de Spring per compartir informació de seguretat durant el processament d'una petició.

Gràcies a aquest filtre, els controladors no necessiten preocupar-se per l'autenticació: si una petició arriba a un controlador, el token ja ha estat validat. A més, qualsevol component del sistema pot obtenir el nom de l'usuari autenticat en qualsevol moment consultant el SecurityContextHolder, cosa que permet implementar la verificació de propietat dels servidors de forma centralitzada.

5.7 Gestió de ports i UPnP

El problema de l'assignació de ports

Cada servidor de Minecraft necessita un port de xarxa exclusiu per acceptar connexions de jugadors. El port predeterminat de Minecraft és el 25565, però si es volen executar múltiples servidors simultàniament, cadascun ha de tenir el seu propi port. SafuHost gestiona aquesta assignació de forma completament automàtica: l'usuari mai ha d'especificar un port manualment.

El repte és trobar un port que no estigui en ús. Una comprovació a nivell de sistema operatiu no és suficient, perquè hi ha una finestra de temps entre el moment en què es troba un port lliure i el moment en què el contenidor Docker comença a escoltar en aquell port. Durant aquesta finestra, si un segon usuari intentés crear un servidor simultàniament, el sistema podria assignar-li el mateix port. La solució implementada a SafuHost és una doble comprovació: primer es consulta la base de dades per verificar que el port no estigui reservat per cap servidor existent, i si passa aquesta primera verificació, es comprova a nivell de sistema operatiu que el port no estigui en ús. Un port només es considera disponible si passa ambdues comprovacions.

UPnP: obertura automàtica de ports al router

Per defecte, els routers domèstics bloquejen les connexions entrants des d'internet. Quan un servidor de Minecraft s'executa a una màquina domèstica, els jugadors d'internet no poden connectar-s'hi a menys que el port estigui obert al router. Configurar manualment l'obertura de ports al router ("port forwarding") és un procés tècnic que molts usuaris no saben fer.

UPnP (Universal Plug and Play) és un protocol que permet a les aplicacions demanar al router que obri un port de forma automàtica, sense intervenció manual. SafuHost utilitza la biblioteca weupnp per enviar aquesta petició al router quan l'usuari inicia un servidor, i per tancar el port quan l'atura. D'aquesta manera, qualsevol jugador pot connectar-se al servidor des d'internet sense que el propietari hagi de tocar la configuració del router.

Cal remarcar que UPnP ha de estar habilitat al router per al seu correcte funcionament, cosa que és el cas en la majoria de routers domèstics per defecte. En el cas que el router no admeti UPnP o que estigui desactivat, el servidor funcionarà igualment en la xarxa local, però no serà accessible des d'internet sense configurar el port forwarding manualment.

5.8 Integració amb l'API de Modrinth

Modrinth és una de les plataformes de mods de Minecraft més importants, amb una API pública i gratuïta que permet cercar, descarregar i obtenir informació detallada sobre mods. SafuHost s'integra

amb aquesta API per oferir als usuaris la possibilitat de gestionar mods directament des del panell de control, sense necessitat de descarregar res manualment.

Cerca i compatibilitat

La cerca de mods a través de l'API de Modrinth permet filtrar per termes de cerca i per la versió de Minecraft i el tipus de loader (Forge o Fabric). Això és especialment important perquè un mod dissenyat per a Fabric no funcionarà en un servidor Forge, i un mod per a la versió 1.20 pot no ser compatible amb la versió 1.21. SafuHost filtra automàticament els resultats per mostrar només els mods compatibles amb el servidor en qüestió, evitant que l'usuari instal·li mods incompatibles.

Quan l'usuari selecciona un mod per instal·lar, el sistema consulta l'API de Modrinth per obtenir l'URL de descàrrega de la versió específica compatible amb el servidor. Aquesta URL apunta al CDN de Modrinth, des d'on el backend descarrega el fitxer JAR del mod i el col·loca a la carpeta de mods del servidor. Gràcies als volums Docker, aquest fitxer és visible immediatament des del contenidor.

Resolució automàtica de dependències

Molts mods no funcionen de forma independent, sinó que requereixen que altres mods estiguin instal·lats. Aquestes dependències poden ser obligatòries (el mod no funciona sense elles) o opcionals (el mod ofereix funcionalitats addicionals si la dependència és present). SafuHost gestiona automàticament les dependències obligatòries: quan l'usuari instal·la un mod, el sistema consulta l'API de Modrinth per obtenir la llista de dependències, verifica quines ja estan instal·lades al servidor, i descarrega i instal·la les que falten.

Aquest procés es realitza de forma recursiva: si una dependència té al seu torn altres dependències, el sistema les resol totes automàticament. El resultat és que l'usuari simplement selecciona el mod que vol i el sistema s'encarrega de tot el procés d'instal·lació, incloses totes les biblioteques necessàries.

Eliminació segura de mods

L'eliminació d'un mod implica esborrar el fitxer JAR corresponent de la carpeta de mods del servidor. Aquesta operació, tot i semblar senzilla, presenta un risc de seguretat: si el nom del mod contingués seqüències de caràcters com "../", un atacant podria fer que el sistema eliminés fitxers fora de la carpeta de mods, comprometent la integritat del servidor o del sistema. SafuHost implementa una validació contra atacs de path traversal que comprova que la ruta resultant de la concatenació del directori de mods i el nom del fitxer no surti mai del directori permès.

5.9 L'arquitectura del frontend en Vue 3

El frontend de SafuHost és una aplicació d'una sola pàgina (SPA) construïda amb Vue 3 i Vite. A diferència de les aplicacions web tradicionals, on cada acció de l'usuari genera una nova petició al servidor que retorna una pàgina HTML completa, una SPA carrega el codi de l'aplicació una sola vegada i actualitza dinàmicament la interfície en resposta a les accions de l'usuari, comunicant-se amb el servidor únicament per intercanviar dades en format JSON.

Reactivitat i el sistema de components

Vue 3 introdueix el sistema de composició (Composition API) com a alternativa al sistema d'opcions de Vue 2. La reactivitat en Vue funciona de la següent manera: quan es declara una variable com a reactiva mitjançant ref() o reactive(), Vue estableix un sistema de seguiment que detecta automàticament quan el valor canvia i actualitza el DOM (Document Object Model) corresponent. Això significa que el

desenvolupador no ha de manipular el DOM directament: simplement modifica les dades i Vue s'encarrega de reflectir els canvis a la interfície.

L'aplicació s'organitza en components reutilitzables. Cada component és un fitxer .vue que encapsula la seva plantilla HTML, la seva lògica JavaScript i els seus estils CSS. Aquesta estructura fa que cada part de la interfície sigui independent i fàcil de mantenir: el component Consola.vue, per exemple, gestiona completament la connexió WebSocket i la visualització dels registres, sense que la resta de l'aplicació hagi de saber com funciona internament.

La gestió de l'estat d'autenticació

Quan l'usuari inicia sessió, el backend retorna un token JWT. El frontend emmagatzema aquest token al localStorage del navegador, que és un mecanisme d'emmagatzematge persistent que manté les dades entre sessions. D'aquesta manera, si l'usuari tanca el navegador i el torna a obrir, no cal que torni a iniciar sessió: el token continua present al localStorage.

El fitxer api.js configura un client Axios amb un interceptor de peticions. Un interceptor és una funció que s'executa automàticament en cada petició HTTP abans d'enviar-la. L'interceptor de SafuHost extreu el token del localStorage i l'afegeix a la capçalera Authorization de la petició. Si el token no existeix, l'interceptor detecta que l'usuari no ha iniciat sessió i el redirigeix automàticament a la pantalla de login.

El router i les guardes de navegació

Vue Router és la biblioteca oficial per gestionar la navegació en aplicacions Vue. Permet definir quines vistes es mostren per a cada URL de l'aplicació. SafuHost defineix quatre rutes: /login per a la pantalla d'inici de sessió, /registro per al registre, /servidores per al llistat principal de servidors i /servidor/:id per a la pantalla de detall d'un servidor concret.

Les guardes de navegació permeten executar lògica abans de canviar de ruta. SafuHost utilitza una guarda global que s'executa en cada canvi de ruta i comprova si l'usuari té un token vàlid al localStorage. Si l'usuari intenta accedir a qualsevol ruta protegida sense autenticar-se, la guarda el redirigeix automàticament a /login. Aquesta comprovació és una mesura de seguretat al costat del client que millora l'experiència d'usuari, però no substitueix la validació del backend: cada petició a l'API és validada pel FiltroJwt independentment del que faci el frontend.

5.10 Consideracions de seguretat

La seguretat d'una aplicació web és un tema multidimensional que afecta tots els components del sistema. SafuHost, tot i ser un projecte de cicle formatiu, ha implementat les mesures de seguretat fonamentals que qualsevol aplicació web hauria de tenir.

Missatges d'error genèrics

Un error comú en el disseny d'aplicacions web és retornar missatges d'error massa detallats. Per exemple, si un sistema d'autenticació retorna "Contrasenya incorrecta" quan la contrasenya és errònia i "Usuari no trobat" quan l'usuari no existeix, un atacant pot usar aquests missatges per deduir si un nom d'usuari concret existeix al sistema. SafuHost retorna un missatge genèric per a qualsevol error d'autenticació, sense revelar si el problema és el nom d'usuari o la contrasenya.

Aïllament per propietari

Un principi de seguretat fonamental és que cada usuari només ha de poder accedir als recursos que li pertanyen. SafuHost implementa aquesta restricció a tots els endpoints que operen sobre un servidor:

quan el backend rep una petició per iniciar, aturar, eliminar o interactuar amb un servidor concret, no és suficient que l'usuari estigui autenticat; cal que l'usuari autenticat sigui el propietari del servidor. Aquesta comprovació es realitza consultant la base de dades i comparant el propietari del servidor amb l'usuari que ha fet la petició.

Sense aquesta verificació, un usuari mal intencionat podria, coneixent l'identificador numèric d'un servidor aliè, iniciar-lo, aturar-lo o eliminar-lo. Amb la verificació de propietat implementada, qualsevol intent d'accedir a un servidor aliè retorna un error 403 Forbidden, independentment del token JWT presentat.

Validació d'entrades i CORS

Totes les dades que arriben al backend des del frontend es tracten com a potencialment no fiables. Les validacions inclouen la comprovació de camps obligatoris, la verificació de formats (com el correu electrònic) i la protecció contra atacs d'injecció. En el cas de la gestió de mods, la validació contra path traversal és especialment important perquè les operacions impliquen lectura i escriptura de fitxers al sistema de fitxers del servidor.

CORS (Cross-Origin Resource Sharing) és un mecanisme de seguretat del navegador que impedeix que un lloc web faci peticions a un domini diferent del seu. Durant el desenvolupament, el frontend s'executa al port 5173 i el backend al port 3000, que el navegador tracta com a dominis diferents. La configuració CORS del backend permet explícitament les peticions des de l'origen del frontend, permetent que l'aplicació funcioni correctament en entorn de desenvolupament.

5.11 Escalabilitat i possibles millores d'arquitectura

SafuHost ha estat dissenyat per funcionar en una màquina local com a projecte educatiu, però l'arquitectura adoptada té en compte els patrons que s'usarien en un sistema de producció real. Analitzar les limitacions actuals i les possibles millores és un exercici útil per comprendre les decisions de disseny preses.

Limitacions de l'arquitectura actual

La principal limitació de l'arquitectura actual és que tots els components (frontend, backend, Docker) s'executen en la mateixa màquina. Això implica que els recursos de la màquina es comparteixen entre el sistema operatiu, el backend Spring Boot, i tots els contenidors de Minecraft actius. Cada servidor de Minecraft pot consumir entre 500 MB i 2 GB de RAM depenent de la configuració i el nombre de jugadors, de manera que el nombre de servidors simultanis que la màquina pot suportar depèn directament de la seva capacitat de memòria.

Una altra limitació és la base de dades SQLite, que no suporta bé les escriptures concurrents de múltiples fils d'execució simultanis. En un escenari amb molts usuaris creant servidors simultàniament, podrien produir-se conflictes d'escriptura. Per a un sistema en producció amb alta concurrència, caldria migrar a un motor com PostgreSQL, que suporta connexions concurrents de forma nativa.

Camí cap a un sistema en producció

Si SafuHost es volgués desplegar com un servei real accessible per internet, caldria abordar diverses millores. En primer lloc, el sistema de seguretat hauria de completar-se: la validació del token JWT a la connexió WebSocket, que és absent en la versió actual, és una millora prioritària. En la versió actual, qualsevol persona que conegués l'URL del WebSocket i l'identificador d'un servidor podria connectar-s'hi i veure els registres en temps real.

En segon lloc, caldria afegir transport xifrat TLS tant per a l'API REST (HTTPS) com per al WebSocket (WSS). Sense xifrat, les comunicacions entre el frontend i el backend viatgen en text pla per la xarxa, cosa que en una xarxa local és acceptable però en internet representa un risc de seguretat important.

En tercer lloc, caldria implementar un sistema de límits de recursos per contenidor, per evitar que un servidor mal configurat consumís tots els recursos de la màquina i afectés la disponibilitat dels altres servidors. Docker permet establir límits de CPU i memòria per contenidor, i integrar aquesta funcionalitat al panell de creació de servidors seria una millora natural de l'arquitectura actual.

6. Conclusions

6.1 Conclusions generals

El desenvolupament de SafuHost ha donat com a resultat una plataforma web funcional que cobreix tots els objectius plantejats inicialment: qualsevol usuari pot crear i gestionar servidors de Minecraft des del navegador, sense necessitat de coneixements tècnics, amb els servidors accessibles des d'internet de forma automàtica.

L'aspecte més complex ha estat la integració de múltiples tecnologies que havien de funcionar de forma coordinada: el backend Spring Boot orquestrant contenidors Docker, la consola en temps real via WebSockets, la instal·lació automàtica de mods amb resolució de dependències des de Modrinth i l'obertura de ports al router via UPnP.

La refactorització aplicant el Principi de Responsabilitat Única ha estat una decisió clau per a la sostenibilitat del codi. El resultat final és una plataforma sòlida amb una interfície intuïtiva i una arquitectura ben estructurada.

6.2 Consecució d'objectius

Objectiu	Estat
Backend Spring Boot amb API REST completa	**Assolit**
Integració amb Docker mitjançant docker-java	**Assolit**
Model de dades amb SQLite i JPA	**Assolit**
Assignació dinàmica de ports sense conflictes	**Assolit**
CRUD complet de servidors	**Assolit**
Control del cicle de vida del contenidor	**Assolit**
Consola en temps real mitjançant WebSockets	**Assolit**
Gestió de la llista blanca en calent	**Assolit**
Instal·lació de mods des de Modrinth amb dependències	**Assolit**
Obertura automàtica de ports via UPnP	**Assolit**
Refactorització aplicant SOLID	**Assolit**

Objectiu	Estat
Sistema d'autenticació amb Spring Security i JWT	**Assolit**
Xifratge segur de contrasenyes amb BCrypt	**Assolit**
Aïllament de servidors per usuari propietari	**Assolit**
Interfície web (Vue 3)	**Assolit**

6.3 Valoració de la metodologia i planificació

La metodologia àgil i iterativa adoptada ha funcionat adequadament. Treballar en iteracions curtes, verificant cada funcionalitat abans de passar a la següent, ha permès detectar errors de forma primerenca. El principal desafiament ha estat l'estimació de temps: alguns problemes tècnics han consumit més temps del previst, com la configuració de Docker en entorn Windows o el descobriment dels requisits no documentats de la imatge itzg/minecraft-server.

En retrospectiva, hauria estat útil dedicar més temps a la fase d'investigació prèvia. L'ús de Git ha estat fonamental per mantenir el fil del treball i revertir canvis quan ha estat necessari.

6.4 Visió de futur

SafuHost disposa d'una base sòlida sobre la qual es poden construir moltes funcionalitats addicionals:

- Validació del token JWT a la connexió WebSocket, com a millora prioritària de seguretat.
- Visualització del consum de CPU i RAM de cada contenidor en temps real.
- Sistema de rols per a supervisió global de la plataforma.
- Suport per a modpacks complets en format .mrpack.
- Còpies de seguretat automàtiques dels mons dels servidors.
- Reactivació automàtica amb AUTOPAUSE abans d'injectar comandes.
- Desplegament en producció amb domini propi i certificat TLS.
- Sistema de notificacions per correu o Discord per a esdeveniments del servidor.

7. Glossari

Terme	Definició
API REST	Interfície de programació mitjançant peticions HTTP retornant dades en JSON.
Spring Boot	Framework Java per a backend amb configuració mínima i servidor web integrat.
JPA / Hibernate	Estàndard per mapejar classes Java a taules de base de dades. Hibernate n'és la implementació.
Docker	Plataforma de contenidors per executar aplicacions en entorns aïllats.
docker-java	Llibreria Java per comunicar-se amb l'API de Docker.
SQLite	Base de dades relacional en un únic fitxer, sense servidor separat.
WebSocket	Protocol bidireccional per a missatgeria en temps real.

Terme	Definició
JWT	JSON Web Token. Token signat per autenticar sense sessions al servidor.
BCrypt	Algorisme de xifratge irreversible per a contrasenyes, lent per disseny.
UPnP	Universal Plug and Play. Obertura automàtica de ports al router.
Modrinth	Plataforma de mods de Minecraft amb API pública gratuïta.
Vue 3	Framework JavaScript per a interfícies web reactives.
SPA	Single Page Application. Navegació sense recarregar la pàgina.
Vite	Servidor de desenvolupament i bundler per a Vue 3 amb HMR.
Axios	Client HTTP amb interceptors per adjuntar el token JWT automàticament.
SOLID	Cinc principis de disseny OO. SafuHost aplica el SRP.
SRP	Principi de Responsabilitat Única: una classe, una responsabilitat.
God Class	Antipatró: classe que acumula massa responsabilitats.
Whitelist	Llista blanca de jugadors autoritzats al servidor.
Path Traversal	Atac que usa .. per accedir a fitxers fora de la carpeta permesa.
Stateless	El servidor no manté sessions. Cada petició porta el seu token JWT.
ConcurrentHashMap	HashMap segur per a múltiples fils d'execució simultanis.
itzg/minecraft-server	Imatge Docker oficial per llançar servidors MC via variables d'entorn.

8. Webgrafia

Documentació oficial

- Spring Boot Documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/
- Spring Security Reference. https://docs.spring.io/spring-security/reference/index.html
- Spring WebSocket Documentation. https://docs.spring.io/spring-framework/reference/web/websocket.html
- Vue 3 Documentation. https://vuejs.org/guide/introduction.html
- Docker Engine API. https://docs.docker.com/engine/api/
- SQLite Documentation. https://www.sqlite.org/docs.html
- Modrinth API. https://docs.modrinth.com/api/
- Hibernate ORM. https://hibernate.org/orm/documentation/

Llibreries i repositoris

- JJWT GitHub. https://github.com/jwtk/jjwt
- docker-java GitHub. https://github.com/docker-java/docker-java
- itzg/minecraft-server Docker Hub. https://hub.docker.com/r/itzg/minecraft-server
- itzg/docker-minecraft-server GitHub. https://github.com/itzg/docker-minecraft-server

Articles i tutorials

- Baeldung. Spring Security with JWT. https://www.baeldung.com/spring-security-oauth-jwt
- Baeldung. Spring WebSocket. https://www.baeldung.com/websockets-spring
- JWT.io. Introduction to JSON Web Tokens. https://jwt.io/introduction

Principis de disseny

- Martin, Robert C. The Single Responsibility Principle. https://blog.cleancoder.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html

9. Annexos

Annex 1 — Estructura de fitxers del projecte

```
SafuHost/
├── backend/
│   ├── pom.xml
│   └── src/main/java/com/safuhost/backend/
│       ├── modelo/          Servidor.java, Usuario.java
│       ├── repositorio/     RepositorioServidor.java, RepositorioUsuario.java
│       ├── servicio/        ServicioServidor.java (coordinador),
│       │                    ServicioPuertos.java, ServicioDocker.java,
│       │                    ServicioMods.java, ServicioWhitelist.java,
│       │                    ServicioUsuario.java
│       ├── controlador/     ControladorServidor.java, ControladorUsuario.java
│       ├── configuracion/   ConfiguracionDocker.java,
│       │                    ConfiguracionWebSocket.java,
│       │                    ConfiguracionSeguridad.java
│       ├── seguridad/       UtilJwt.java, FiltroJwt.java
│       └── websocket/       ConsolaWebSocketHandler.java
└── frontend/
    ├── package.json, vite.config.js
    └── src/
        ├── main.js, App.vue, api.js, router.js
        ├── views/           Login.vue, Registro.vue,
        │                    Servidores.vue, Servidor.vue
        └── components/      Consola.vue, Whitelist.vue, Mods.vue
```

Annex 2 — Dependències principals (pom.xml)

Dependència	Versió	Ús
spring-boot-starter-web	3.x	API REST i servidor web integrat
spring-boot-starter-data-jpa	3.x	Accés a la base de dades amb JPA
spring-boot-starter-websocket	3.x	WebSockets per a la consola en temps real
spring-boot-starter-security	3.x	BCrypt, filtres JWT, configuració de rutes
jjwt-api / jjwt-impl / jjwt-jackson	0.12.6	Generació i validació de tokens JWT
sqlite-jdbc	3.45.1.0	Driver JDBC per a SQLite

Dependència	Versió	Ús
hibernate-community-dialects	6.4.4	Dialecte de Hibernate per a SQLite
docker-java-core	3.3.6	Llibreria principal de docker-java
docker-java-transport-httpclient5	3.3.6	Transport HTTP per a Docker
lombok	—	Generació de getters/setters amb anotacions

Annex 3 — Variables d'entorn dels contenidors

Variable	Valor exemple	Funció
EULA	TRUE	Accepta el contracte de llicència (obligatori)
VERSION	1.21.1	Versió de Minecraft a instal·lar
TYPE	VANILLA / FORGE / FABRIC	Tipus de servidor
DIFFICULTY	peaceful / easy / normal / hard	Dificultat del joc
MODE	survival / creative / adventure	Mode de joc
PVP	true / false	Combat entre jugadors
ONLINE_MODE	true / false	Verificació de comptes de Mojang
ENABLE_WHITELIST	true / false	Activa/desactiva la llista blanca
WHITELIST	Jugador1,Jugador2	Llista inicial de jugadors autoritzats
OPS	Admin1,Admin2	Jugadors amb permisos d'administrador
AUTOPAUSE	false	Desactivat per garantir mc-send-to-console
CREATE_CONSOLE_IN_PIPE	true	Necessari per injectar comandes

Annex 4 — Endpoints de l'API REST

Mètode	Endpoint	Auth	Descripció
POST	/api/usuarios/registro	No	Crear compte
POST	/api/usuarios/login	No	Retorna token JWT
GET	/api/servidores/todos	Sí	Servidors de l'usuari
GET	/api/servidores/{id}	Sí + prop.	Detall del servidor
POST	/api/servidores/crear	Sí	Crear servidor
PUT	/api/servidores/{id}	Sí + prop.	Actualitzar servidor
DELETE	/api/servidores/{id}	Sí + prop.	Eliminar servidor
POST	/api/servidores/{id}/iniciar	Sí + prop.	Arrencar servidor
POST	/api/servidores/{id}/parar	Sí + prop.	Aturar servidor
GET	/api/servidores/{id}/estado	Sí + prop.	Estat en directe

Mètode	Endpoint	Auth	Descripció
POST	/api/servidores/{id}/consola/comando	Sí + prop.	Enviar comanda
GET	/api/servidores/{id}/mods	Sí + prop.	Mods instal·lats
GET	/api/servidores/mods/buscar	Sí	Cercar a Modrinth
POST	/api/servidores/{id}/mods/instalar/{id}	Sí + prop.	Instal·lar mod
DELETE	/api/servidores/{id}/mods/{nom}	Sí + prop.	Eliminar mod
GET	/api/servidores/{id}/whitelist	Sí + prop.	Llista blanca
POST	/api/servidores/{id}/whitelist/{j}	Sí + prop.	Afegir jugador
DELETE	/api/servidores/{id}/whitelist/{j}	Sí + prop.	Treure jugador
WS	/ws/consola/{id}	No	Consola en temps real

Nota sobre l'ús d'intel·ligència artificial

Durant el desenvolupament d'aquest projecte s'han utilitzat eines d'intel·ligència artificial en dues vessants diferenciades.

En la fase de desenvolupament, la IA ha estat una eina de suport tècnic valuosa, especialment per descobrir tecnologies i llibreries que donaven solució a problemes concrets: des de la configuració de la connexió amb Docker en entorn Windows fins a l'ús de mc-send-to-console per injectar comandes en calent, passant per la gestió de dependències de mods o l'obertura automàtica de ports via UPnP. En tots els casos, la decisió d'adoptar o descartar cada solució ha estat presa i assumida íntegrament per l'autor.

Pel que fa a la redacció de la memòria, la IA ha intervingut com a suport ortogràfic i estilístic, ajudant a expressar les idees de l'autor de forma més clara i acadèmica. Totes les idees, decisions, explicacions i valoracions recollides en aquest document són de l'autor; la IA únicament ha contribuït a millorar-ne la comprensió i la qualitat expositiva.

Llicència