Marc Sala Pellicer

Aplicació web per a la gestió i emmagatzematge de projectes WebAssembly

TREBALL DE FI DE GRAU

dirigit pel Dr. Marc Sánchez Artigas

Grau d'Enginyeria Informàtica

UNIVERSITAT ROVIRA I VIRGILI

Tarragona
2024

Resum

Aquest projecte consisteix en el desenvolupament d'una aplicació web de tipus SaaS (*Software as a Service*) destinada a la gestió i emmagatzematge de fitxers WebAssembly en forma de catàleg.

L'aplicació s'emmarca en el projecte europeu CloudSkin i està dissenyada específicament per al grup de recerca CloudLab.

Una de les principals motivacions d'aquest projecte és la necessitat de centralitzar i estandarditzar la gestió d'aquest tipus de fitxers, que fins ara es trobava dispersa i fragmentada en diferents ubicacions i formats. Amb aquesta aplicació, es pretén unificar tots els recursos en un únic punt, oferint una solució més coherent, eficient i accessible.

Aquest catàleg d'aplicacions Wasm (WebAssembly) no només simplifica la gestió interna dels recursos de CloudLab, sinó que també proporciona una base sòlida per a futures investigacions i desenvolupaments a la comunitat de WebAssembly. Es preveu que aquesta solució es pugui expandir i adaptar a noves necessitats, mantenint un alt nivell d'escalabilitat i integració amb altres sistemes, oferint així un gran valor per la comunitat.

Resumen

Este proyecto consiste en el desarrollo de una aplicación web del tipo SaaS (Software as a Service) destinada a la gestión y almacenamiento de archivos WebAssembly en forma de catálogo.

La aplicación se enmarca en el proyecto europeo CloudSkin y está diseñada específicamente para el grupo de investigación CloudLab.

Una de las principales motivaciones de este proyecto es la necesidad de centralizar y estandarizar la gestión de este tipo de archivos, que hasta ahora se encontraba dispersa y fragmentada en diferentes ubicaciones y formatos. Con esta aplicación, se pretende unificar todos los recursos en un único punto, ofreciendo una solución más coherente, eficiente y accesible.

Este catálogo de aplicaciones Wasm (WebAssembly) no solo simplifica la gestión interna de los recursos de CloudLab, sino que también proporciona una base sólida para futuras investigaciones y desarrollos en la comunidad de WebAssembly. Se prevé que esta solución pueda expandirse y adaptarse a nuevas necesidades, manteniendo un alto nivel de escalabilidad e integración con otros sistemas, ofreciendo así un gran valor para la comunidad.

Abstract

This project involves the development of a SaaS (Software as a Service) web application designed for the management and storage of WebAssembly files in the form of a catalog.

The application is part of the European CloudSkin project and is specifically designed for the CloudLab research group.

One of the main motivations for this project is the need to centralize and standardize the management of these types of files, which until now have been dispersed and fragmented across different locations and formats. With this application, the goal is to unify all resources in a single location, providing a more coherent, efficient, and accessible solution.

This Wasm (WebAssembly) application catalog not only simplifies the internal management of CloudLab's resources but also provides a solid foundation for future research and development within the WebAssembly community. It is anticipated that this solution can be expanded and adapted to new needs, maintaining a high level of scalability and integration with other systems, thus offering significant value to the community.

Índex

Índex de figures

Índex de taules

1. Introducció

El WebAssembly, sovint anomenat Wasm, és una tecnologia innovadora (*creada al 2015*) que ens aporta una gran millora en el rendiment de les aplicacions web. Aquesta tecnologia consisteix en un format de codi binari que permet que el codi s'executi a una velocitat gairebé nativa des del navegador web. Es va desenvolupar com una eina de baix nivell que complementa JavaScript i ofereix la possibilitat de compilar codi escrit en diversos llenguatges, com ara Go, C++, Rust, i altres, per tal de executar-se en un entorn de navegador.

El gran avantatge envers al rendiment, es deu a que a diferència de JavaScript, que és un llenguatge interpretat a temps real pel navegador, el WebAssembly es compila prèviament en un format binari, permetent una execució posterior molt més ràpida i amb una gran portabilitat, ja que el format binari és independent de la plataforma i per tant el mateix codi podrà ser executat en una gran varietat de dispositius.

Com que aquesta tecnologia es tant recent en el món de la informàtica, la informació disponible sobre ella es troba sovint dispersa i poc intuïtiva, cosa que deixa als desenvolupadors amb una comprensió parcial o ambigua en alguns aspectes tècnics, fet que no només dificulta l'aprenentatge i la implementació de Wasm, sinó que també frena la innovació en aquest camp emergent.

WasmDepot, emmarcat dins del projecte europeu CloudSkin i destinat al grup de recerca CloudLab, neix amb l'objectiu de solucionar aquestes mancances a través del desenvolupament d'una aplicació web de tipus SaaS[1].

L'aplicació pretén centralitzar la gestió i l'emmagatzematge dels fitxers Wasm, creant un catàleg estandarditzat, ampliant la documentació disponible i facilitant l'accés a fitxers i recursos relacionats amb aquesta tecnologia, millorant així l'eficiència del treball tant al grup de recerca CloudLab, com a tota la comunitat WebAssembly, i així, ajudar i fomentar al creixement de la mateixa.

Tot i que aquest catàleg s'ha creat inicialment per centralitzar i estandarditzar la gestió de fitxers Wasm, el seu potencial va molt més enllà. A mesura que el WebAssembly evolucioni, es preveu anar afegint noves funcionalitats, adaptant-se a les necessitats dels desenvolupadors, i també es preveu anar integrant eines col·laboratives com fòrums de dubtes o seccions dedicades a la formació.

[1] Software as a Serice

2. WebAssembly

Des de fa gairebé dues dècades, JavaScript ha estat l'únic llenguatge nadiu dels navegadors web. Durant aquest temps, les exigències de rendiment han anat augmentant i això ha fet que en certes situacions el rendiment que proporciona JavaScript sigui millorable. Degut aquest rendiment baix en certs àmbits, com la càrrega d'aplicacions en el navegador, va sorgir com a resposta el WebAssembly.

El WebAssembly és un llenguatge innovador en el món del desenvolupament web, consisteix en compilar codi escrit de llenguatges d'alt nivell a un format binari compactat. Un cop tenim el codi en binari, podrà ser executat en un entorn segur dins del navegador. Tot i que Wasm no està dissenyat per substituir JavaScript, sí que és un molt bon complement que permet oferir una major velocitat de càrrega i execució especialment en casos on es requereixi un rendiment elevat.

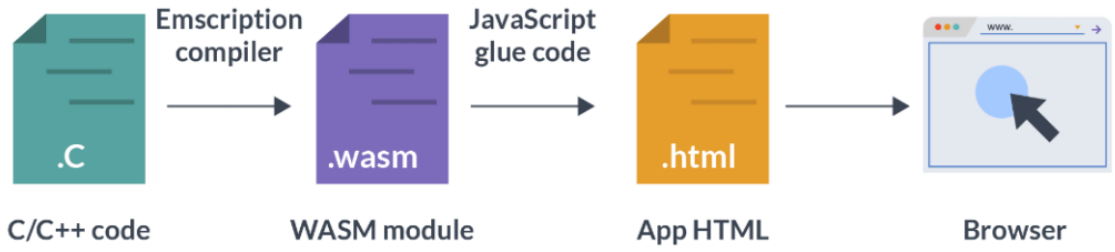


Figura 1 *Funcionament WebAssembly*

2.1. Execució en diferents entorns

Inicialment Wasm es va dissenyar per executar dins del navegador web com s'ha explicat anteriorment, però també hi ha l'opció d'utilitzar Wasm fora del navegador.

-Dins del navegador

Wasm està aïllat del sistema operatiu, la qual cosa significa que no té accés directe als recursos del sistema com el sistema de fitxers o les xarxes sense l'ajuda de JavaScript. Aquesta limitació garanteix la seguretat del codi que s'executa en un navegador, protegint l'usuari de codi potencialment maliciós.

-Fora del navegador

Quan s'executa fora del navegador, adquireix una nova dimensió que li permet interactuar amb el sistema operatiu i altres recursos de l'entorn. S'han d'utilitzar entorns d'execució especialitzats també coneguts com a "runtimes". Aquests runtimes actuen com una màquina virtual que carrega i executa els mòduls Wasm en un entorn controlat. Alguns exemples de runtimes són Wasmtime, Wasmer, etc.

Per poder accedir aquestes capacitats es fa mitjançant WebAssembly System Interface (*WASI*), és una interfície que permet als mòduls Wasm accedir de manera segura a recursos, fitxers, crides de xarxa, etc. Utilitzant Wasi aconseguim donar accés a les parts del sistema concretes evitant qualsevol accés no autoritzat.

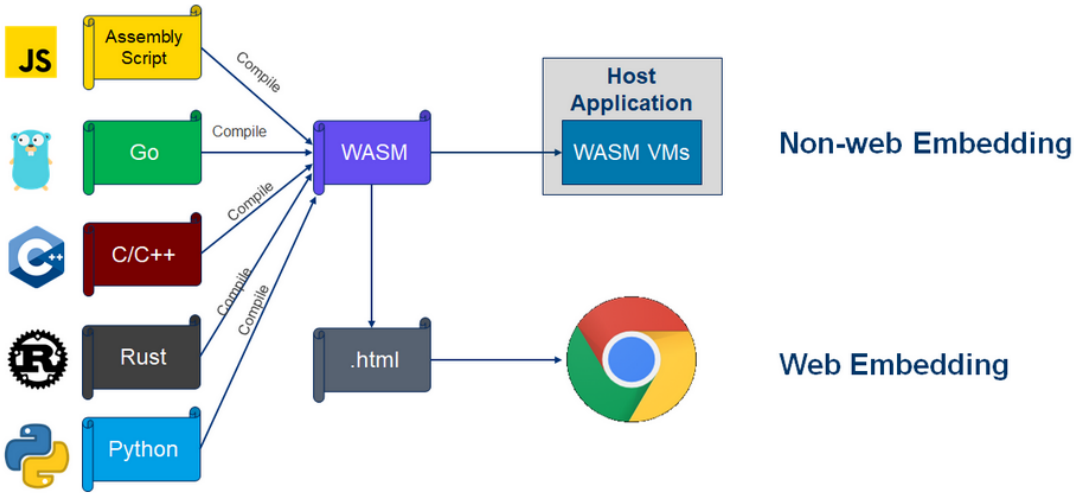


Figura 2 *Entorns WebAssembly*

2.2. Avantatges i inconvenients

Per tant, després de introduir-nos una mica en el món del WebAssembly i saber com funciona i les prestacions que ens aporta, podem destacar certs avantatges, com ara:

- Alt rendiment: ens permet executar codi a una gran velocitat. Ideal per aplicacions de gran intensitat computacional com jocs, simulacions, etc.
- Portabilitat: treballa independentment de la plataforma i arquitectura del sistema, ja que es pot executar en qualsevol dispositiu amb un runtime compatible.
- Seguretat: s'executa en un entorn controlat que limita l'accés a recursos no autoritzats, evitant problemes amb codi maliciós.
- Interoperabilitat: permet als desenvolupadors crear les aplicacions en un gran nombre de llenguatges diferents.
- Execució fora del navegador: permet executar-se fora del navegador i accedint a recursos del sistema, ampliant les seves aplicacions.

Com a inconvenients podem destacar:

- Complexitat en el desenvolupament: els desenvolupadors encara no estan del tot familiaritzats amb la tecnologia i la falta d'informació estandarditzada complica el seu aprenentatge.
- Limitacions amb alguns llenguatges: tot i que Wasm suporta molts llenguatges, hi ha llenguatges més compatibles que altres.
- Manca de suport en algunes característiques avançades: alguna funcionalitat més avançada com el multithreading, encara estan en desenvolupament o limitades.

2.3. Aplicacions pràctiques

WebAssembly té moltes aplicacions pràctiques en diferents sectors gràcies a les seves característiques, podríem destacar-ne les següents:

- Aplicacions web de gran rendiment: tot tipus d'aplicacions que requereixin gran rendiment com ara jocs, multimèdia, simulacions, etc.
- Microserveis: es pot utilitzar Wasm per executar microserveis eficients i segurs gràcies a la seva ràpida càrrega i aïllament.
- Dispositius Iot (Internet of Things): ideal per a dispositius amb recursos limitats gràcies al seu rendiment eficient. Pot ser utilitzat per controlar sensors, executar aplicacions lleugeres en dispositius Iot, etc.
- Aplicacions de seguretat: es pot utilitzar per executar codi d'origen desconegut protegint al sistema host de possibles vulnerabilitats.

Aquestes són algunes de les múltiples aplicacions pràctiques que pot tenir el WebAssembly. En alguns casos la implementació d'aquesta tecnologia suposa tants avantatges envers la tecnologia utilitzada actualment, que pot arribar a suposar un gran canvi. Una de les aplicacions més prometedores és la computació distribuïda, ja que el Webassembly ens aporta grans millores envers el desenvolupament amb tecnologies com màquines virtuals, cloud, containers, etc.

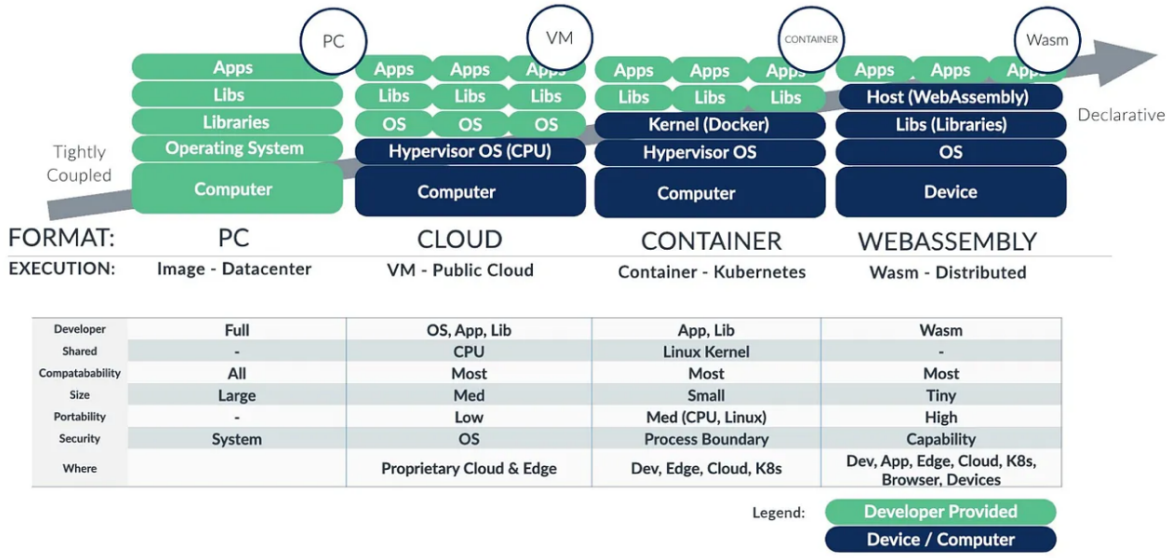


Figura 3 *Comparativa computació distribuida*

3. Descripció del projecte

Aquest projecte pretén cobrir les necessitats que s'han detectat a l'hora d'adoptar la tecnologia WebAssembly. La principal mancança és que la informació al respecte es troba tota molt dispersa i això complica la corba d'aprenentatge pels desenvolupadors, que acaben conformant-se utilitzant tecnologies en les quals estan més familiaritzats però que no s'adeqüen tant a les necessitats que tenen, com ho faria WebAssembly.

També, s'ha trobat que en els pocs llocs on trobem informació, el contingut està poc estandarditzat i cada projecte Wasm mostra la informació de diverses formes.

3.1. Aplicacions existents

Un dels exemples que mostren aquesta problemàtica, és la pàgina web "madewithwebassembly.com" en la qual els diferents projectes mostren la informació amb estructures diferents, sense funcionalitats que facilitin la cerca de tipus de projectes concrets, i amb mancances rellevants d'informació.

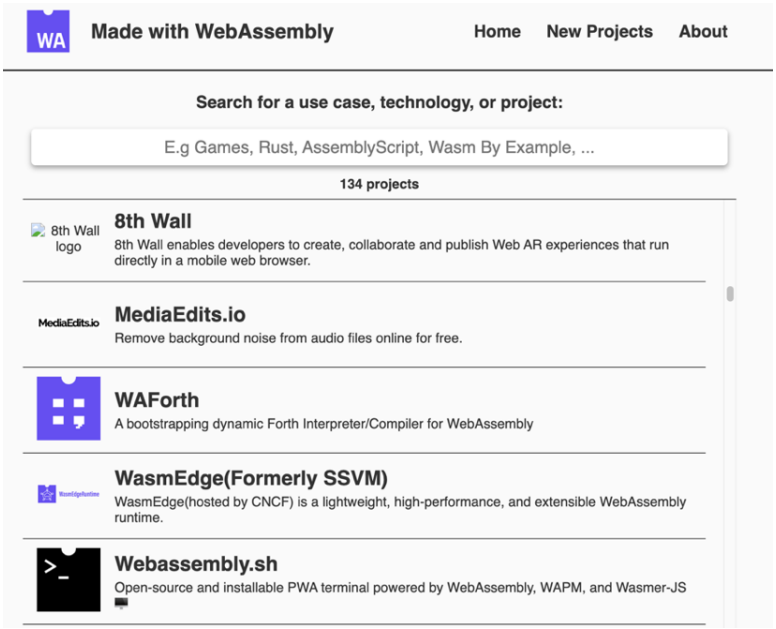


Figura 4 *Vista principal madewithwebassembly.com*

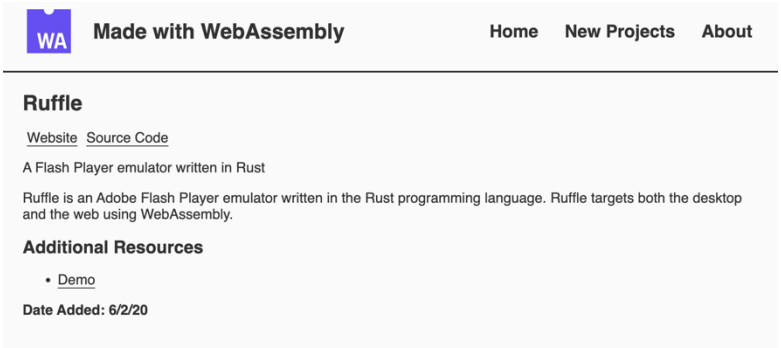


Figura 5 *Exemple de projecte*

3.2. Proposta de solució

Aquesta problemàtica acostuma a ser normal en tecnologies emergents, i per tal de promoure l'adopció, s'ha d'anar millorant poc a poc aquest tipus d'aspectes, facilitant així que la informació sigui més accessible, útil i comprensible.

Des del grup de recerca CloudLab, requereixen una solució per tindre centralitzats els mòduls Wasm que utilitzen, així com, trobar les informacions ben estructurades i entenedores.

Per això, he decidit crear una aplicació web en format de catàleg que permeti tindre centralitzats i ben documentats tots aquests mòduls Wasm i així ajudar a l'adopció d'aquesta tecnologia. En aquesta aplicació es podran crear entrades a projectes propis i consultar els ja existents creats per la comunitat.

Una altre aspecte important, és que es podran pujar mòduls a la web per tal que els usuaris un cop entrin al projecte, se'ls puguin descarregar, i seguint les instruccions, executar-los.

Amb aquesta aplicació volem aconseguir augmentar la comunitat de WebAssembly i facilitar el treball dels desenvolupadors que ja hi treballen.

Figura 6 *Logotip aplicació WasmDepot*

4. Requisits

Per poder complir amb la finalitat de la nostra aplicació és imprescindible definir una sèrie de requisits que ens guiaran tant en el procés de creació de la aplicació, fent-nos una funció de guió, com en la fase avaluació per comprovar si s'han complerts aquest requisits establerts. Definirem dos tipus de requisits, per una banda requisits funcionals, els quals detallaran les accions específiques que el sistema ha de ser capaç de realitzar per complir els objectius del projecte, i per l'altre banda, els requisits no funcionals els quals detallaran com han de fer aquestes funcionalitats.

4.1. Requisits funcionals

Els requisits funcionals els classificarem segons les accions que s'han de poder fer depenent del tipus d'usuari actiu.

Usuari NO Iniciat/Registrat

00. S'ha de poder registrar a un usuari nou.
01. S'ha de poder iniciar sessió.
02. S'ha de poder navegar pels menús.
03. S'ha de poder visualitzar tots els projectes penjats i utilitzar els diferents filtres per facilitar la cerca.
04. S'ha de poder accedir als detalls dels projectes i visualitzar les dades.
05. S'ha de poder descarregar els fitxers en cas de que el projecte en tingui.

Usuari Iniciat/Registrat

Ha de poder fer totes les funcions anteriors més les següents:

06. S'ha de poder tancar sessió.
07. S'ha de poder crear un projecte nou.
08. S'ha de poder navegar pels menús incloent l'apartat "byMe".
09. S'ha de poder afegir projectes a la llista personal de preferits de cada usuari.
10. En el cas de que s'entri a un projecte el qual l'usuari actiu sigui el seu creador, s'ha de poder modificar o afegir contingut en les pestanyes de Usage, Screenshots i Benchmark.
11. En el cas de que s'entri a un projecte el qual l'usuari actiu sigui el seu creador, s'ha de poder eliminar el projecte.

Usuari Administrador

Ha de poder fer totes les funcions anteriors i més les següents:

12. Ha de poder fer les funcions de modificar/afegir contingut en qualsevol projecte.
13. Ha de poder eliminar qualsevol projecte.

4.2. Requisits no funcionals

En aquest apartat es detallen les exigències que el sistema ha de complir per assegurar-nos que, apart de complir amb les funcionalitats requerides, ho faci de una manera correcta sota diferents condicions.

Requisits d'usabilitat:

L'aplicació ha de ser intuïtiva i fàcil d'utilitzar per tots els usuari, ja sigui usuaris amb experiència amb WebAssembly, com usuaris que es volen introduir. Per tant haurà de complir una sèrie de requisits específics com:

- La interfície d'usuari ha de ser clara, amb un disseny visual atractiu i una estructura lògica i organitzada.
- Haurà de tenir una corba d'aprenentatge reduïda.
- La navegació per l'aplicació ha de ser consistent en totes les vistes.
- Feedback immediat. En el moment que es faci alguna acció errònia o correcte, l'usuari ha de percebre el resultat de la mateixa.

Requisits de rendiment:

L'aplicació ha de complir uns certs requeriments de rendiment, com ara:

- Un temps de resposta ràpid.
- Ha de permetre escalabilitat.
- Eficiència en l'ús de recursos

Requisits de seguretat:

L'aplicació ha de complir uns requisits bàsics de seguretat, com ara:

- Utilitzar mètodes d'autenticació i autorització.
- Protecció de dades sensibles com ara contrasenyes.

Requisits de manteniment:

Per últim, s'han de complir certs requisits de manteniment per facilitar la feina posteriorment. Alguns requisits que s'han de complir són:

- Codi ben documentat
- Modularitat del codi
- Escalabilitat de l'arquitectura
- Modularitat de l'arquitectura

4.3. Diagrama de casos d'ús

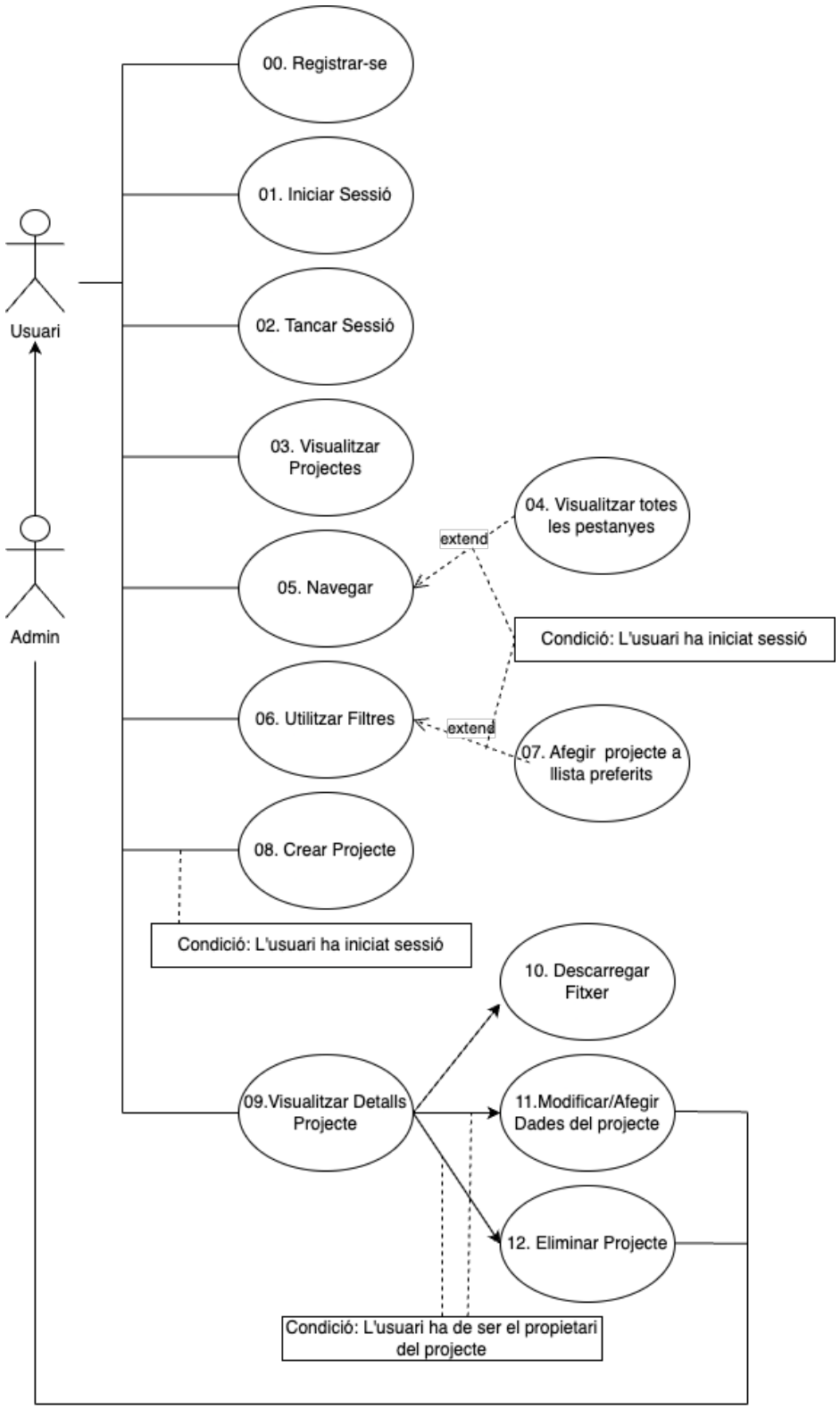


Figura 7 *Diagrama Cassos d'ús*

5. Arquitectura

L'apartat de l'arquitectura de l'aplicació és important per entendre com està dissenyat i organitzat el sistema des d'una perspectiva tècnica. Aquesta secció descriu l'estructura general de l'aplicació, inclosos els components principals, les seves interaccions i les tecnologies utilitzades. L'objectiu és proporcionar una comprensió clara de com s'ha creat l'aplicació, garantint la seva escalabilitat, manteniment i rendiment.

Es detallaran les diferents capes que la componen, com ara el frontend, el backend, la base de dades i qualsevol altre servei o component integrat. També, descriurà com funcionen junts aquests components per dur a terme les funcions previstes.

5.1. Arquitectura de microserveis

Per crear l'aplicació WasmDepot s'ha implementat l'arquitectura de microserveis. Aquest tipus d'arquitectura consisteix en descompondre l'aplicació en una col·lecció de serveis petits, independents i desplegables de manera autònoma. Cada microservei tindrà una funció específica i es comunicarà a través d'interfícies.

L'aplicació s'ha implementat amb aquest tipus d'arquitectura perquè te una sèrie de beneficis molt convenients:

- Independència i desplegament autònom: cada component és autònom, per tant, es poden iniciar, apagar, actualitzar, etc. sense afectar a l'estat de la resta de serveis.

- Escalabilitat: permet escalar individualment els serveis segons les necessitats que tinguem.

- Resiliència i tolerància a errors: com que cada microserveis funciona independentment dels altres, un error en un servei no necessàriament afectarà a un altre servei, permetent reiniciar o arreglar el problema d'aquest servei sense aturar la resta.

- Mantenibilitat: les modificacions es poden implementar amb menys risc i més fàcil de detectar errors ja que els entorns són més reduïts.

Per contrari, aquesta arquitectura comporta una sèrie de complicacions, de les quals podríem destacar les següents:

- Complexitat superior: gestionar i coordinar diferents serveis pot ser complex.
- Consistència de dades: pot ser complex mantenir la consistència de les dades en un sistema basat en arquitectura de microserveis.
- Sobrecàrrega de comunicació: pot afectar a la latència i sobrecàrrega de la xarxa depenent del volum de comunicacions precises pel funcionament.
- Seguretat: la seguretat es complicarà ja que cada servei ha de ser segur per si mateix i les comunicacions entre ells també.

Veient els avantatges i inconvenients podem destacar que tot i el desafiament a la implementació, que serà més complexa que en altres arquitectures, els avantatges que ens aporta són molt notables. És per això que actualment moltes aplicacions ja implementen aquesta arquitectura.

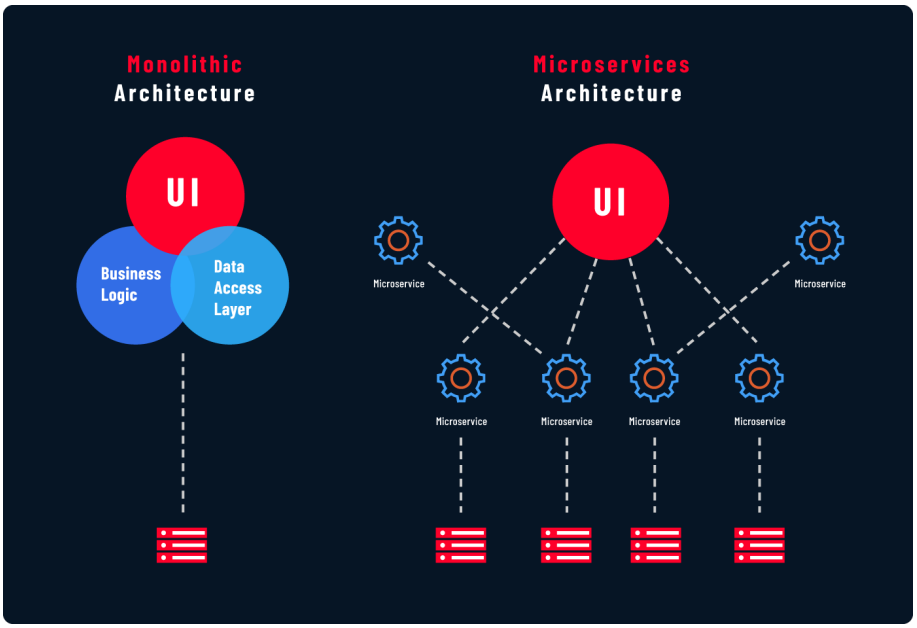


Figura 8 *Arquitectura Monolítica vs Arquitectura Microserveis*

5.2. Docker

Docker és una plataforma de programari que facilita la creació, distribució i execució d'aplicacions dins de contenidors. Els contenidors són una tecnologia de virtualització a nivell de sistema operatiu que permet l'execució de diverses aplicacions independents dins d'un únic servidor.

La diferència principal envers les màquines virtuals, és que aquestes inclouen un sistema operatiu complet, mentre que els contenidors, comparteixen el mateix nucli del sistema operatiu però tenen els seus espais d'execució per separat. Per tant, tindrem els contenidors funcionant de manera independent amb un gran nivell d'aïllament, reduint així els conflictes. Una altre característica destacables és la lleugeresa dels contenidors envers les màquines virtuals. Finalment, la portabilitat que permeten els contenidors és molt superior ja que es podran executar en qualsevol sistema on hi hagi Docker instal·lat.

Funcionament

Docker simplifica la gestió dels contenidors amb eines per crear, desplegar i executar aplicacions dins d'aquest contenidors.

Dins dels contenidor podem crear el nostre espai de treball de diferents formes.

Una opció és la utilització de plantilles creades per altres usuaris anomenades "Docker Images". Aquestes plantilles de només lectura, contenen les instruccions per crear l'entorn amb les especificacions concretes de la imatge. A "Docker Hub" podem trobar un registre públic on els desenvolupadors pugen i comparteixen les imatges Docker.

Una altre opció, és mitjançant "Dockerfile", que és un fitxer de text que conté les instruccions per crear una imatge de Docker. Aquestes instruccions descriuen la instal·lació de paquets, copia de fitxers a l'interior de la imatge, exposició de ports, etc. També, es pot incloure l'ús d'alguna imatge base ja creada i afegir els teus requeriments personalitzats.

Un cop tenim la nostra imatge o la nostra configuració amb Dockerfile crearem els contenidors amb "Docker Engine" el qual es el component central de Docker. S'encarregarà de la creació i execució dels contenidors en el sistema operatiu.

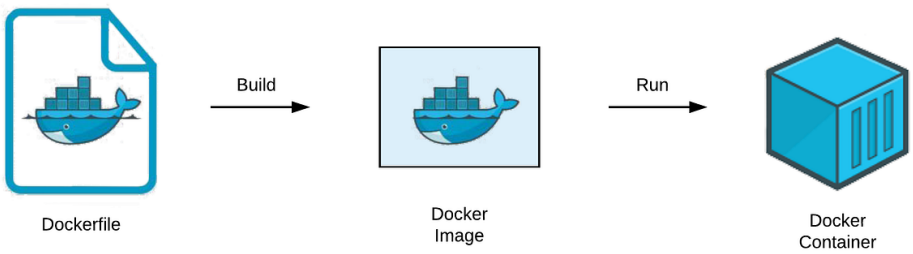


Figura 9 *Funcionament Docker*

5.3. Contenidors

Un cop es va decidir utilitzar l'arquitectura de microserveis pels grans beneficis que aportava al projecte, es va haver d'especificar quines necessitats tindria l'aplicació:

- La creació d'un frontend, per tal que els usuaris puguin interactuar per mitjà d'una interfície.
- L'aplicació necessitarà una Base de Dades per guardar el contingut d'aquesta.
- L'aplicació ha de poder guardar i descarregar fitxers de diferent tipus.
- La creació d'un backend, amb la finalitat d'utilitzar-lo com a API[2] entre el frontend i la resta de serveis.
- Finalment, precisarem d'un servidor que actuï com a frontend públic, servint contingut i redirigint el trànsit. Permetrà augmentar el rendiment cachejant els fitxers estàtics.

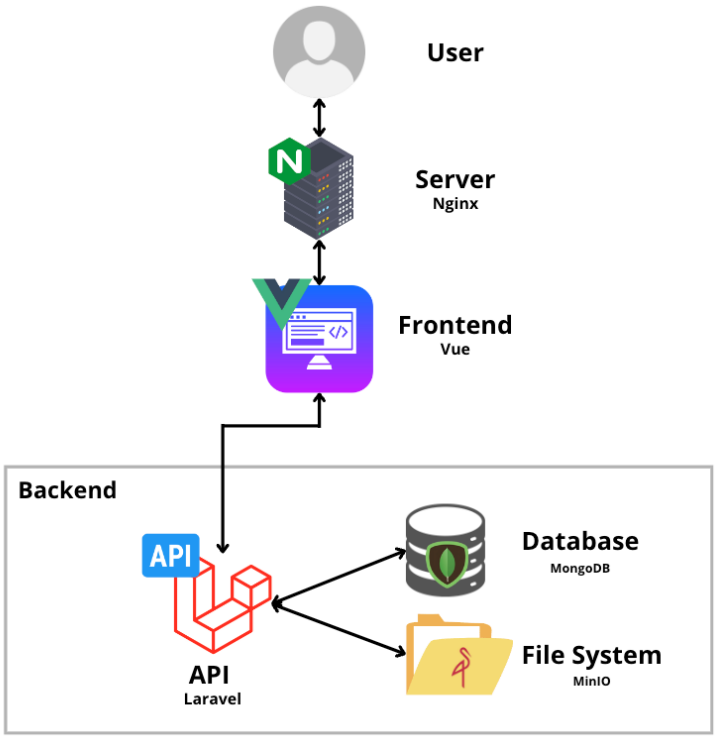


Figura 10 *Diagrama arquitectura aplicació*

Un cop les necessitats estaven definides, es va decidir que precisaríem de 5 contenidors diferents per crear l'aplicació, cada un amb les característiques per fer la seva funció de la manera òptima. Es va fer una cerca de imatges Docker que poguessin ser adients.

[2]Application Programming Interface

Contenidor Frontend

En aquest contenidor s'ha de crear un espai de treball per poder implementar i executar una aplicació frontend, que en el cas d'aquest projecte s'ha decidit que sigui amb Vue.js. Per aconseguir-ho, s'ha creat un Dockerfile on es descriuen les accions que s'han de fer per crear el contenidor correctament. Aquest es el Dockerfile del contenidor de Frontend:

```
# Utilitza una imatge base de Node.js
FROM node:latest

# Configura el directori de treball
WORKDIR /app

# Copia els fitxers package.json i package-lock.json al directori de treball
COPY package*.json ./

# Instal·la les dependències del projecte, incloent Vuetify
RUN npm install vuetify

# Copia la resta del codi del projecte Vue.js al contenidor
COPY . .

# Instal·la les dependències restants amb npm
RUN npm install

# Exposa el port 8080
EXPOSE 8080

# Comanda per iniciar l'aplicació de desenvolupament
CMD npm run serve
```

Figura 11 *Dockerfile contenidor Frontend*

En aquest fitxer podem veure com el contenidor es construirà a partir d'una imatge base de Node.js, assegurant que s'utilitzi la versió més recent.

També, es faran unes certes configuracions com definir el directori de treball, copiar certs fitxers, instal·lar dependències del projecte com ara vuetify, exposar el port, etc.

Prèviament, a la creació del contenidor s'ha creat el projecte Vue en una carpeta. Quan es crea el contenidor, el Dockerfile indica que s'han de fer les accions definides, entre elles copiar el contingut del directori on esta el projecte Vue dins del contenidor.

Contenidor Base de Dades

La base de dades, sempre és una part molt important alhora de dissenyar un projecte. Per això s'han mirat les necessitats que ha de complir i s'ha arribat a la conclusió de fer servir una base de dades no relacional.

Normalment les bases de dades relacionals, acostumen a ser millors en casos en què la integritat de les dades, la normalització i les operacions transaccionals són crítiques.

En el cas WasmDepot és més útil tenir una gran flexibilitat per guardar les dades, una gran capacitat de manejar grans volums i tenir una alta disponibilitat en entorns distribuïts.

S'ha optat per una base de dades no relacional orientada a documents com és MongoDB. En aquesta base de dades es guardarà el contingut en format JSON[3].

Al contenidor s'hi ha carregat la imatge de Docker més recent de MongoDB.

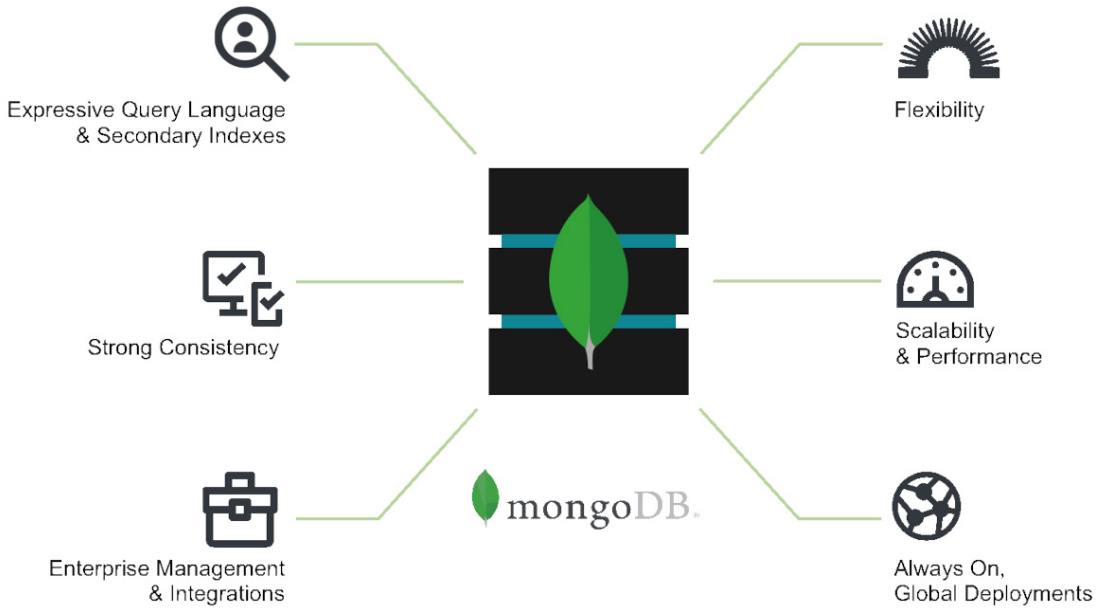


Figura 12 *Característiques de MongoDB*

[3]JavaScript Object Notation

Contenidor Emmagatzematge de Fitxers

S'ha decidit utilitzar MinIO com a solució per a la gestió i emmagatzematge de fitxers a l'aplicació.

MinIO proporciona una interfície d'emmagatzematge d'objectes que permet guardar i recuperar fitxers mitjançant una senzilla API. A nivell bàsic, les operacions de càrrega i descàrrega de fitxers es gestionen mitjançant operacions de lectura i escriptura sobre "buckets", que són contenidors d'objectes dins del sistema. Això permet una gestió eficient de grans volums de dades sense necessitat de preocupar-se per la complexitat de l'emmagatzematge físic.

La compatibilitat amb Amazon S3 permet que MinIO serveixi com a solució intermediària flexible. Si en el futur es decideix migrar a Amazon S3 o un altre proveïdor d'emmagatzematge basat en S3, el canvi pot ser realitzat amb mínimes alteracions en el codi que utilitza la interfície d'API. Això proporciona una escalabilitat i adaptabilitat a llarg termini, assegurant que el sistema pugui evolucionar per satisfer les necessitats canviants, sense requerir una reestructuració major.

En el contenidor s'ha utilitzat la imatge més recent de MinIO.

Figura 13 *Logotip MinIO*

Contenidor Backend

Per interactuar entre frontend, base de dades i emmagatzematge de fitxers, s'ha de crear un contenidor amb funció de backend. Per fer aquesta funció s'ha decidit utilitzar l'estructura de Laravel, un framework de PHP[4].

```
FROM php:8.2-fpm

# Instal·lar les dependències del sistema
RUN apt-get update && apt-get install -y \
    libpng-dev \
    libjpeg62-turbo-dev \
    libfreetype6-dev \
    zip \
    unzip \
    git \
    curl \
    libssl-dev \
    pkg-config \
    libcurl4-openssl-dev \
    libonig-dev \
    libxml2-dev \
    libzip-dev

# Instal·la les extensions PHP necessàries
RUN docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install -j$(nproc) pdo pdo_mysql mbstring exif pcntl bcmath gd zip

# Instal·la Composer
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer

# Instal·lar l'extensió de MongoDB
RUN pecl install mongodb && docker-php-ext-enable mongodb

# Estableix el directori de treball
WORKDIR /var/www

# Copia els arxius del projecte
COPY . .

# Instal·la les dependències del projecte
RUN composer install

# Dona permisos
RUN chown -R www-data:www-data /var/www

EXPOSE 9000

CMD ["php-fpm"]
```

Figura 14 *Dockerfile contenidor Backend*

En aquest Dockerfile podem veure com el contenidor del backend tindrà com a imatge base la versió 8.2 de PHP amb PHP-FPM.

Es pot apreciar com defineix les dependències del sistema que s'han d'instal·lar, tant com les extensions de PHP. També, instal·larà Composer, que és l'eina de gestió de dependències de PHP.

[4]Hypertext Prepocessor

Per poder interactuar amb la base de dades MongoDB farà falta instal·lar la extensió que ho permet.

Prèviament, s'haurà creat un projecte Laravel on gestionaran les peticions que arribin dels altres contenidors.

A continuació, farà les configuracions del directori de treball, copiarà el projecte Laravel dins del contenidor, instal·larà més dependències del projecte amb Composer, modificarà una sèrie de permisos per assegurar que l'usuari tingui accés complet al codi de l'aplicació i exposarà el port que s'utilitzarà, per defecte el 9000.

Per acabar, executarà la comanda que permet iniciar el servidor PHP-FPM per gestionar les sol·licituds HTTP[5] quan el contenidor estigui creat.

Contenidor Servidor

En el projecte, s'ha triat Nginx com a servidor per gestionar i servir tota l'aplicació. Aquesta elecció es basa en diverses raons tècniques que ofereix Nginx, tot i que hi havia altres opcions que també complien les necessitats.

Finalment, la combinació de rendiment, eficiència, facilitat d'ús, i compatibilitat amb Docker fa que sigui la imatge elegida.

En el fitxer default.conf es configura el servidor:

```
server {
    listen 80;

    server_name localhost;

    # Serveix l'aplicació Vue
    location / {
        proxy_pass http://vue:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Serveix API Laravel
    location /api {
        proxy_pass http://laravel:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Configuració per gestionar fitxers PHP
    location ~ \.php$ {
        root /var/www;
        fastcgi_pass laravel:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

Figura 15 *Default.conf de NGINX*

Podem veure el port que escolta i on es serveixen les diferents peticions.

[5]Hypertext Transfer Protocol

5.4. Docker Compose

Un cop tenim totes les imatges i configuracions de cada contenidor cal configurar el Docker Compose.

El Docker Compose consisteix en una eina que permet definir i gestionar les aplicacions multi-contenidor de manera senzilla, mitjançant un sol fitxer anomenat "Docker-compose.yml". Ens permetrà orquestrar tots els serveis que componen l'aplicació, facilitant el desplegament, l'escala, i la gestió dels diversos contenidors. En aquest fitxer es configuraran diferents paràmetres, podem destacar-ne els següents:

- "services": defineix els diferents serveis/contenidors que formen part de l'aplicació.

- "build": especifica que el servei ha de construir la imatge Docker a partir d'un Dockerfile.

- "image": defineix la imatge que s'ha d'utilitzar per aquest servei concret.

- "volumes": els volums són molt importants ja que proporcionen persistència de dades, això vol dir que les dades es conservaran tot i tancar el contenidor. En aquesta secció es defineixen.

- "networks": defineix la xarxa que utilitzaran els serveis per comunicar-se entre ells.

- "depends-on": en aquesta secció s'indicarà de quins serveis depèn aquest servei, per tant, s'haurà d'esperar a que els altres serveis estiguin actius per iniciar-se.

- "enviroment": configura les variables d'entorn del contenidor.

- "ports": definineix els ports que utilitza cada contenidor.

Configuracions del Docker Compose

En aquest apartat s'explicarà com està configurat cada contenidor en el Docker Compose.

Contenidor Backend

```yaml
version: '3.8'

services:

  laravel:
    build: ./backend
    volumes:
      - ./backend:/var/www
    environment:
      - DB_CONNECTION=mongodb
      - DB_HOST=mongodb
      - DB_PORT=27017
      - DB_DATABASE=tfg
      - DB_USERNAME=root
      - DB_PASSWORD=marcpass
    ports:
      - "8000:8000"
    depends_on:
      - minio
      - mongodb
    networks:
      - app-network
    command: php artisan serve --host=0.0.0.0
```

Figura 16 *Configuració contenidor backend al docker compose*

Podem observar que el contenidor es crearà utilitzant el Dockerfile que s'ha creat anteriorment. Es defineixen els volums, ports i network. Modifica les variables d'entorn del contenidor, en aquest cas les variables respecte a la base de dades, per poder accedir a ella.

Els serveis de MinIO i MongoDB, hauran d'iniciar-se abans que aquest ja que així ho mostra el paràmetres depends_on.

Finalment, executa una comanda dins del contenidor que el que fa és llançar un servidor web de desenvolupament.

Contenidor Frontend

```yaml
vue:
  build: ./frontend
  volumes:
    - ./frontend:/app
  depends_on:
    - laravel
  ports:
    - "8080:8080"
  networks:
    - app-network
```

Figura 17 *Configuració contenidor frontend al docker compose*

En aquest servei, el contenidor utilitzarà el Dockerfile creat anteriorment. Es defineix els volums, ports i network. I s'iniciarà un cop el servei Laravel estigui actiu.

Contenidor Base de Dades

```yaml
mongodb:
  image: mongo:latest
  ports:
    - "27017:27017"
  environment:
    - MONGO_INITDB_ROOT_USERNAME=root
    - MONGO_INITDB_ROOT_PASSWORD=marcpass
  volumes:
    - mongodb_data:/data/db
  networks:
    - app-network
```

Figura 18 *Configuració contenidor Base de Dades al docker compose*

S'utilitzarà la última imatge de MongoDB per crear-lo. Es defineixen els ports, volums i network, i es modifica les variables d'entorn del contenidor referent al usuari i contrasenya per accedir-hi.

Contenidor Emmagatzematge de Fitxers

```yaml
minio:
  container_name: minioStorage
  image: minio/minio
  environment:
    MINIO_ROOT_USER: "minio"
    MINIO_ROOT_PASSWORD: "minio123"
  volumes:
    - minio_data:/data
  ports:
    - "9000:9000"
    - "9001:9001"
  networks:
    - app-network
  command: server /data --console-address ":9001"
```

Figura 19 *Configuració contenidor emmagatzematge de fitxers*

Es crearà a partir de la imatge Docker de MinIO. Definim el nom del contenidor, els volums, la network i els ports, que en aquest cas n'utilitzarà dos, un per connectar-se a la API i l'altre per veure la Web User Interface (WebUI) al navegador. També, es modificaran les variables d'entorn referents al accés al servei i finalment s'executarà una comanda per assignar el port 9001 al WebUI, evitant que s'assigni un port aleatori.

Contenidor Servidor

```yaml
nginx:
  image: nginx:latest
  volumes:
    - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    - ./frontend/dist:/usr/share/nginx/html
  ports:
    - "80:80"
  depends_on:
    - vue
    - laravel
  networks:
    - app-network
```

Figura 20 *Configuració contenidor Servidor al docker compose*

Es construirà a partir de la imatge de Nginx. Defineix el port que escoltarà el contenidor, la network i els volums. Aquest contenidor dependrà dels serveis Vue i Laravel.

Al final del Docker Compose tenim definit els volums i la network que utilitza el driver "bridge".

```yaml
volumes:
  mongodb_data:
  minio_data:

networks:
  app-network:
    driver: bridge
```

Figura 21 *Definició de "volumes" i "network" del docker compose*

6. Implementació

En aquest apartat, es detallaran els aspectes clau de la implementació de l'aplicació, explicant com s'han integrat les diferents tecnologies i components per a construir el sistema complet.

L'objectiu és proporcionar una visió clara de l'estructura de cada part del projecte, fitxers importants i les seves funcions, explicar les decisions que s'han pres i detallar el funcionament d'algunes funcions importants.

Abans de centrar-nos en cada part més específica, explicar que he decidit utilitzar les tecnologies de Laravel, pel backend i Vue amb Vuetify, pel frontend ja que vaig veure que es podien complementar, i que tenien suport per treballar tant amb MongoDB com amb MinIO.

Figura 22 *Tecnologies utilitzades*

6.1. Frontend

El frontend és la part visible per l'usuari, on es gestiona la interacció amb l'aplicació a través d'una interfície web. Com ja s'ha avançat anteriorment, per implementar el frontend s'ha utilitzat Vue.js.

He triat Vue perquè es un framework lleuger i flexible que facilita el desenvolupament d'interfícies d'usuari modernes i responsives. Unes de les característiques principals és la seva capacitat per crear components reutilitzables que milloren l'eficiència del desenvolupament i el seu gran ecosistema de plugins i eines que permeten treballar amb molta comoditat.

Destacar que un cop creat el projecte de Vue, s'han afegit extensions com Vuetify que aporta una col·lecció de components, com ara botons, formularis, menús, etc. que permeten construir interfícies d'usuari atractives i coherents de manera ràpida i eficient. A més, Vuetify facilita el disseny responsiu, característica que millora molt l'experiència del usuari.

6.1.1 Estructura

En aquesta secció s'explicarà l'estructura del projecte Vue, destacant-ne els fitxers més importants i les seves funcionalitats.

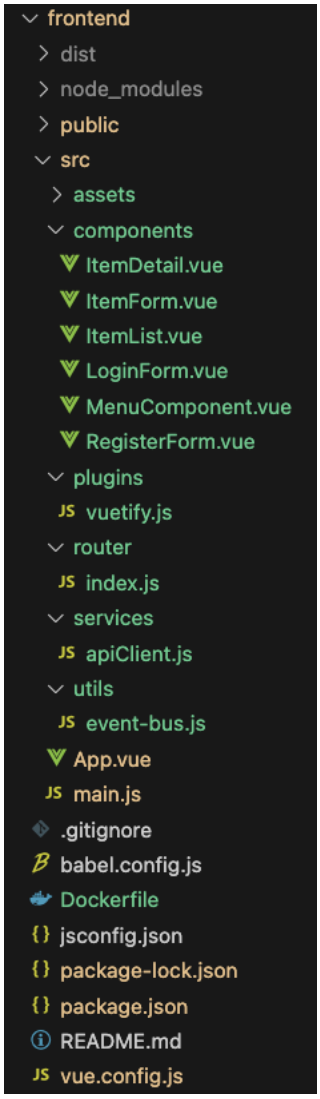


Figura 23 *Estructura fitxers projecte Vue*

Com es pot veure en la figura 22, el projecte Vue s'estructura amb diferents carpetes, com la carpeta "públic", que conté els recursos estàtics públics, la carpeta "node_modules", que es on s'instal·len totes les dependències del projecte al executar "npm install", o la carpeta "src", la més important de totes ja que serà on tinguem el codi de l'aplicació.

En aquest cas la carpeta "src", conté 6 directoris més i dos fitxers.

En el fitxer main.js, es crea l'aplicació i es configura la instància de Vue, s'importen els components globals, plugins i altres llibreries. Bàsicament es on es posa en marxa l'aplicació.

```
import Vue from 'vue';
import App from './App.vue';
import router from './router';
import Vuetify from 'vuetify';
import 'vuetify/dist/vuetify.min.css';

// Utilitza Vuetify
Vue.use(Vuetify);

const vuetify = new Vuetify();

Vue.config.productionTip = false;

// Crea una instància de Vue
new Vue({
  router,
  vuetify,
  render: h => h(App),
}).$mount('#app');
```

Figura 24 *Fitxer main.js*

En aquest fitxer li passarem vuetify i router al projecte Vue i se li indicarà que App.vue és el component arrel que s'ha de renderitzar.

L'altre fitxer que hi ha a la carpeta "src" és el component arrel App.vue, que actuarà com a punt d'entrada principal a l'aplicació. Defineix l'estructura bàsica i conte el disseny general. En l'apartat de components s'explicaran les funcions i decisions de disseny importants d'aquest component arrel.

Cada una de les carpetes tindrà una funció diferent, destacant les carpetes “services”, “router” i “components”. En la carpeta “services” hi ha el fitxer “apiClient.js”.

La seva funció és configurar una instancia d’Axios per gestionar les peticions HTTP de l’API de l’aplicació.

```
import axios from 'axios';

// Crear una instància d'Axios
const apiClient = axios.create({
  baseURL: 'http://localhost:8000/api',
  headers: {
    'Content-Type': 'application/json',
  },
});

// Afegir un interceptor per incloure el token JWT
apiClient.interceptors.request.use(
  config => {
    const token = localStorage.getItem('token'); // Obtenir el token emmagatzemat
    console.log(token);
    if (token) {
      console.log('ha entrat');
      config.headers['Authorization'] = `Bearer ${token}`; // Afegir el token a les encapçalaments
      console.log('ha sortit');
    }
    return config;
  },
  error => Promise.reject(error)
);

export default apiClient;
```

Figura 25 *Fitxer apiClient.js*

Les accions principals són la creació d’una instància Axios amb una configuració predefinida, incloent la URL base de l’API (http://localhost:8000/api) i els encapçalaments (Content-Type: application/json).

També, s’afegeix un interceptor de sol·licituds que el que farà serà afegir al encapçalament de les sol·licituds el token de autorització, per tal de tenir una major seguretat. Finalment, s’exporta per poder-ho utilitzar en altres fitxers.

En la carpeta “router” hi ha el fitxer “index.js”. Aquest fitxer té com a funció configurar el sistema de rutes per l’aplicació Vue. Utilitza el paquet “vue-router”

```
Vue.use(Router);

const routes = [
  {
    path: '*',
    redirect: '/menu',
  },
  {
    path: '/register',
    name: 'Register',
    component: RegisterForm,
  },
  {
    path: '/login',
    name: 'Login',
    component: LoginComponent,
  },
  {
    path: '/menu',
    name: 'Menu',
    component: MenuComponent,
  },
  {
    path: '/newItem',
    name: 'NewItem',
    component: ItemForm,
  },
  {
    path: '/list',
    name: 'List',
    component: ItemList,
  },
  {
    path: '/items/:id',
    name: 'ItemDetail',
    component: ItemDetail,
    props: true,
  },
];
```

Figura 26 *Fitxer index.js, rutes frontend*

Com es pot comprovar en la figura 25, es defineixen les diferents rutes que es podran accedir des del navegador. En cada ruta es posa el path i el component al que criden.

```
router.beforeEach((to, from, next) => {
  const isAuthenticated = !!localStorage.getItem('token');

  if (isAuthenticated && (to.path === '/login' || to.path === '/register')) {
    next('/menu'); // Redirigeix a la pàgina de l'usuari autenticat si ja està logejat
  } else {
    next(); // Continua amb la navegació
  }
});

export default router;
```

Figura 27 *Fitxer index.js, mètode beforeEach*

També s’ha implementat una funció per tal que no es pugui accedir a login o registre si ja hi ha un usuari actiu. I al final s’exporta.

6.1.2 Components

En aquest apartat es descriuen els components de Vue que s'han creat específicament per a l'aplicació. Els components són les peces fonamentals que conformen la interfície d'usuari i cadascun compleix una funció especifica dins del sistema.

En cada component tindrem 3 parts en el codi. En primer lloc, el template, aquesta secció defineix l'estructura visual del component mitjançant el llenguatge HTML.

La segona part del codi és l'script emprant JavaScript, aquesta secció contindrà la lògica del component, amb dades, mètodes, propietats computades, etc.

Finalment l'última part del codi són els styles amb CSS. Aquí si definiran els estils dels elements del template. En el cas d'aquest projecte, s'han implementat "scoped", per tal d'evitar conflictes amb altres parts de l'aplicació fent així els estils específics per cada component.

App.vue

El component principal, com s'ha mencionat anteriorment, és el fitxer App.vue ja que és el component arrel i defineix l'estructura de l'aplicació.

A nivell de template, aquest component es basa en crear el "app toolbar", on podrem veure les opcions de canviar entre menús, registrar-nos, crear nous projectes, etc. Aquestes opcions variaran en funció de si hi ha un usuari actiu o no.

```
  </v-app-bar>

  <v-main class="main-content">
    <router-view />
  </v-main>
</v-app>
```

Figura 28 *Router-view en App.vue*

Destacar l'element router-view ubicat després del app-bar. Això crea l'estructura principal de l'aplicació ja que tota la aplicació seguirà aquesta estructura de tenir la barra de navegació a dalt i l'element router-view canviarà segons el component actiu en aquell moment.

De la part lògica del component destacar la manera de detectar si hi ha un usuari actiu. En aquesta aplicació quan un usuari inicia sessió al sistema, es guarda un token per autenticar les sol·licituds i les propietats del usuari en el localStorage, això permet accedir a aquestes dades des de tota l'aplicació.

```
async logout() {
  try {
    await apiClient.post('/logout');
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    this.updateAuthStatus({ isAuthenticated: false });
    this.goToMenu()
  } catch (error) {
    console.error('Error en tancar sessió:', error.response?.data);
  }
},
```

Figura 29 *Mètode logout, App.vue*

Destacar en aquest component, l'acció de tancar sessió. Com podem veure en el codi, primerament es fa una interacció amb l'API, per tal de notificar al backend que l'usuari vol tancar sessió, seguidament s'eliminen els elements guardats al localStorage , es crida a un mètode per informar que l'usuari no esta actiu en el frontend i finalment es redirigeix al menú amb el mètode goToMenu(). Comentar també l'ús d'un event-bus que permet vigilar l'estat del localStorage, per saber si es tanca sessió i actualitzar les opcions que es mostren al toolbar sense haver de refrescar la pàgina.

LoginForm.vue

Aquest component és el que permetrà a l'usuari Iniciar sessió a l'aplicació. És un component bastant senzill, conté bàsicament un formulari on l'usuari introduirà les seves credencials per poder iniciar la sessió. També té la possibilitat de anar al component de registre si no està registrat encara.

```
//metode per enviar l'informacio a API
async submitForm() {
  try {
    const actUser = {
      email: this.email,
      password: this.password,
    };

    const response = await apiClient.post('/login', actUser);

    const token = response.data.token;
    localStorage.setItem('token', token);

    const user = await apiClient.post('/getUser');
    localStorage.setItem('user', JSON.stringify(user.data.user));

    // Notifica mitjançant event bus que s'ha canviat status
    EventBus.$emit('authStatusChanged', { isAuthenticated: true, user: user.data.user });

    this.$router.push({ path: '/menu' });
  } catch (error) {
    if (error.response) {
      console.error('Login error:', error.response.data.error || 'Unknown error');
      alert(`Error: ${error.response.data.error || 'Unknown error'}`);
    } else {
      console.error('Connection error:', error.message);
      alert('Connection error. Please try again later.');
    }
  }
},
```

Figura 30 *Mètode submitForm, de LoginForm.vue*

D'aquest component destacar el disseny implementat per fer-lo atractiu visualment i el mètode submitForm(), que enviarà les dades al backend. Podem veure que el mètode el que fa es guardar les dades del formulari en una constant, aquestes s'envien al backend utilitzant la instància de apiClient. Un cop es rep la resposta de l'API amb el token, aquest es guarda a localStorage, seguidament es torna a fer una altre petició per obtenir la informació de l'usuari actiu i també es guarda al localStorage.

Mitjançant un event-bus emet una senyal notificant que s'ha modificat l'estat d'autenticació. Finalment es fa un router.push al menú.

RegisterForm.vue

Aquest component és molt semblant a l'anterior. En termes de disseny són iguals. El que canvia és que el formulari té els camps necessaris per donar d'alta un usuari nou. Es validen les dades introduïdes com també es feia en el anterior i el procés de fer el enviament del formulari es gairebé el mateix.

S'ha utilitzat un vídeo diferent que al login per tal de ser més atractiu visualment.

MenuComponent.vue

Aquest és un dels components més complexos de l'aplicació. Bàsicament és la pagina d'inici. En aquesta vista, es pot veure el banner de l'aplicació seguit d'un petit text introductori.

Després es pot veure una secció de categories populars de projectes de WebAssembly. Aquestes categories que es mostren, es poden clicar i et redirigeixen al llistat de projectes WebAssembly d'aquella categoria en concret, una funció útil per trobar fàcilment el tipus de projectes ens quals estàs interessat.

Cada projecte es mostra en forma de targeta, cada targeta, té la imatge, el títol, la sinopsis, la categoria del projecte i el tipus de contingut. També tenen un botó en forma de cor per afegir els projectes a la llista de preferits de l'usuari. Hi ha una altre etiqueta que mostra l'usuari creador del projecte, en el cas de que l'usuari creador del projecte sigui l'usuari actiu, l'etiqueta es veurà verda, evidenciant que l'usuari pot modificar-lo.

Sota d'aquesta secció hi haurà dos apartats més, una llista de preferits, que només es mostrarà quan hi hagi un usuari actiu i cadascun amb la seva llista pròpia, i una altre dels projectes més nous. L'usuari podrà navegar per aquests apartats mitjançant les fletxes, i si es vol mostrar tot el llistat complet tant de preferits com de projectes nous, s'haurà de clicar el títol de l'apartat.

La lògica del component no és massa complicada ja que consisteix en mètodes per facilitar la navegació de l'usuari i funcionalitats com la llista de preferits

ItemForm.vue

Aquest component s'utilitza per crear els projectes nous. S'hi pot accedir des de la toolbar clicant el símbol de més, que apareix quan hi ha un usuari actiu. El component consisteix en un formulari on es demanen els camps imprescindibles per crear un projecte nou. Destacar que tenim diferents tipus d'inputs, ja siguin text-fields, v-select, file-input, etc.

La manera d'enviar les dades del formulari és la següent:

```
async submitForm() {
  const formData = new FormData();
  formData.append('title', this.form.title);
  formData.append('sinopsis', this.form.sinopsis);
  formData.append('description', this.form.description);
  formData.append('image', this.form.image);
  formData.append('web', this.form.web);
  formData.append('github', this.form.github);
  formData.append('categories', this.form.categories);
  formData.append('type', this.form.type);
  formData.append('language', this.form.language);
  formData.append('supportedPlatform', this.form.supportedPlatform);
  formData.append('compilerToolchain', this.form.compilerToolchain);
  formData.append('creatorId', this.form.creatorId);
  formData.append('creatorName', this.form.creatorName);

  try {
    const response = await apiClient.post('/newItem', formData, {
      headers: {
        'Content-Type': 'multipart/form-data',
      },
    });
    console.log(response.data);
    this.$router.push({ path: `/items/${response.data.itemId}` });
  } catch (error) {
    console.error('Error submitting the form:', error);
  }
},
handleFileChange(type, event) {
  this.form[type] = event.target.files[0];
},
```

Figura 31 *Mètode submitForm, fitxer ItemForm.vue*

Com es pot veure en la figura 30, es crea una constant anomenada formData amb tot el contingut del formulari, incloent-hi informació de l'usuari que l'esta creant. Aquesta constant s'envia mitjançant una instància de apiClient amb el header 'Content-Type': 'multipart/form-data' que permet passar camps de diferent tipus ja que tenim imatges i text en la informació enviada a l'API.

Un cop es rep la resposta es fa un router.push al component dels detalls del projecte que s'acaba de crear, per canviar el router-view que es veu en el App.vue.

ItemList.vue

Aquest component consisteix en mostrar la llista de projectes. Els projectes que es mostrin dependran dels filtres que s'apliquin. Aquesta funcionalitat ajuda a trobar el contingut que busca l'usuari més fàcilment. Els filtres que es poden aplicar són: depenent de la categoria del projecte, el tipus de contingut, els preferits del usuari o els projectes que pertanyen al usuari actiu. També hi ha una barra de cerca per buscar els projectes pel nom.

Aquest component s'hi pot accedir per diferents rutes depenent des d'on s'accedeixi. Per la URL es passen paràmetres utilitzant querys, que indicaran quin tipus de filtre s'ha d'aplicar al llistat alhora de mostrar-lo. Per detectar els diferents casos s'ha utilitzat un watcher, que és una característica potent per observar canvis en dades reactivament i executar codi addicional en resposta a aquests canvis.

```
watch: {
  '$route.query': {
    handler() {
      this.updateFilters();
    },
    immediate: true,
  }
```

Figura 32 *Watcher fitxer IntemList.vue*

Com es pot veure en la figura, el watcher observa la query de la URL i quan hi ha algun canvi, crida al mètode updateFilters(), fent així que es mostrin els projectes amb els nous filtres demanats.

```
computed: {
  filteredItems() {
    const showMyItemsOnly = this.$route.query.myItems === 'true';
    const showFavoritesOnly = this.$route.query.favorites === 'true';

    let filtered = this.items.filter(item => {
      const belongsToUser = !showMyItemsOnly || item.creatorId === this.currentUserId;
      const matchesCategory = !this.selectedCategory || item.categories === this.selectedCategory;
      const matchesType = !this.selectedType || item.type === this.selectedType;
      const matchesSearch = item.title.toLowerCase().includes(this.searchQuery.toLowerCase());

      return belongsToUser && matchesCategory && matchesType && matchesSearch;
    });

    if (showFavoritesOnly) {
      filtered = filtered.filter(item => this.favorites.includes(item._id));
    }

    if (this.$route.query.filter === 'latest') {
      filtered = filtered.sort((a, b) => new Date(b.created_at) - new Date(a.created_at));
    }

    return filtered;
  },
},
```

Figura 33 *filteredItems del fitxer ItemList.vue*

Com es pot veure a figura 32, tenim una "computed propertie" que recalcula els projectes que s'han de mostrar al component. Ho aconsegueix, aplicant un filtre a la llista dels projectes, comprovant quins filtres s'estan precisant en aquest moment.

ItemDetail.vue

Aquest component és el que mostrarà tota la informació de cada projecte. És molt important que tot el contingut es vegi clar i sense confondre a l'usuari. Per això, s'ha decidit implementar un sistema de pestanyes, on cada pestanya té un contingut específic.

Damunt d'aquestes pestanyes tindrem el títol, la sinopsis i a la dreta els botons per accedir a la web i al GitHub del projecte. En el cas de que l'usuari actiu sigui el creador del projecte, també apareixerà un botó per poder-lo eliminar.

La pestanya principal s'anomena Overview. En aquesta pestanya s'hi pot veure la informació bàsica que introduïm al crear el projecte.

La segona pestanya és la de Usage, on s'hi pot trobar informació per instal·lar el contingut del projecte, un botó per veure el fitxer Readme i en el cas de que n'hi hagi, un botó per descarregar el fitxer del projecte.

La tercera pestanya és la de Screenshots. El creador pot afegir totes les captures que cregui necessàries.

Finalment, l'última pestanya és la de Benchmark. El creador pot introduir resultats obtinguts de diferents benchmarks o el rendiment del projecte utilitzant el WebAssembly. També, pot afegir algunes imatges per ajudar a la comprensió dels resultats obtinguts dels benchmarks.

Les tres darreres pestanyes poden ser editades pel creador en el moment que es vulgui, canviant textos, fitxers, afegint fotos, etc. Els usuaris que no siguin el creador no tindran accés a aquestes funcions, només podran visualitzar el contingut i si hi ha un fitxer guardat, descarregar-lo.

De la implementació del codi destacar que aquestes pestanyes s'han creat amb els elements v-tabs i v-tabs-items, on a dins d'aquest elements es definien les estructures de cada pestanya.

Els mètodes implementats en aquest component són semblants als emprats per enviar dades a l'API anteriorment, fent petites modificacions depenent dels diferents tipus de dades que s'envien. Cada cop que es modifica un projecte la data de actualització queda registrada.

6.2. Backend

El backend és la part que gestiona la lògica de l'aplicació, la interacció amb la base de dades i l'emmagatzematge de fitxers, i s'encarrega de gran part de la seguretat. Per implementar el backend s'ha utilitzat Laravel, un framework de PHP.

He triat Laravel per diverses raons. Primerament, és un framework que ofereix una estructura clara i organitzada, fet que facilita el manteniment i l'evolució de l'aplicació a llarg termini.

A més, Laravel inclou un sistema de rutes potent i senzill que permet definir com s'assignen als diferents controladors les sol·licituds HTTP rebudes. Això, es complementa amb el sistema de middlewares, que ens permet aplicar una lògica a les rutes.

També, destacar la facilitat per poder integrar altres servei i funcionalitats a través de packages. En el cas d'aquest projecte els packages més importants utilitzats són els de mongodb, el jwt-auth i el flysystem-aws-s3-v3

6.2.1. Estructura

A continuació, es mostrarà l'estructura del projecte Laravel, i s'explicarà les funcions dels fitxers més importants del projecte.

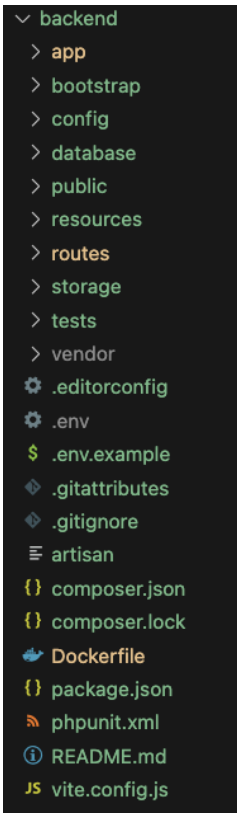


Figura 34 *Estructura principal projecte Laravel*

Com es pot veure en la figura 33, el projecte s'estructura amb una sèrie de carpetes i fitxers, els quals tenen cada un funcions específiques pel correcte funcionament del projecte.

Dels fitxers que es poden veure a la figura 33, destacar el fitxer "composer.json", en el qual hi haurà totes les dependències del projecte i la seva versió.

L'altre fitxer molt important és el .env. En aquest fitxer s'emmagatzemen les variables d'entorn del projecte. Un dels punts claus d'aquest projecte ha estat configurar tot l'entorn perquè funcioni correctament, i aquest fitxer té una gran importància alhora de connectar funcionalitats d'altres contenidors amb el backend.

Destacarem les configuracions següents:

```
DB_CONNECTION=mongodb
DB_HOST=mongodb
DB_PORT=27017
DB_DATABASE=tfg
DB_USERNAME=root
DB_PASSWORD=marcpass
```

Figura 35 *Configuració Base de Dades al backend, fitxer .env*

Una de les configuracions que s'han fet ha estat la de la base de dades, en la qual indicarem al projecte quin tipus de base de dades s'utilitzarà i com s'hi haurà de connectar, definint ports, usuari i contrasenya.

```
FILESYSTEM_DISK=minio

AWS_ACCESS_KEY_ID=rd8sCZCEGxowLvivRsMe
AWS_SECRET_ACCESS_KEY=jQ6DVUrZFqhRBSm62JAzP2mTLJ5cuan4N8q3ilM2
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=tfgbucket
AWS_ENDPOINT=http://minioStorage:9000
AWS_URL=http://minioStorage:9000
AWS_USE_PATH_STYLE_ENDPOINT=true
```

Figura 36 *Configuració MinIO, fitxer .env*

També, tenim la definició de les variables d'entorn referents al container de MinIO. On s'introdueixen les credencials obtingudes en el procés de crear el bucket que utilitzarem i definim els endpoints i entrades al servidor.

```
AUTH_GUARD=api
AUTH_PASSWORD_BROKER=users
AUTH_PASSWORD_TIMEOUT=10800
JWT_SECRET=aVYuOBPXcgtbqDPfrftZ2MKgcxo1bH9ttVmoGTHJMLGgHZ25P9Gs2JtwmX3SHVAY
```

Figura 37 *Configuració JWT, fitxer .env*

La configuració del sistema d'autentificació de l'aplicació. Es defineix la clau secreta per signar i validar els tokens generats per JWT i el temps que un token serà vàlid.

Destacar les carpetes app, config i routes de la Figura 33.

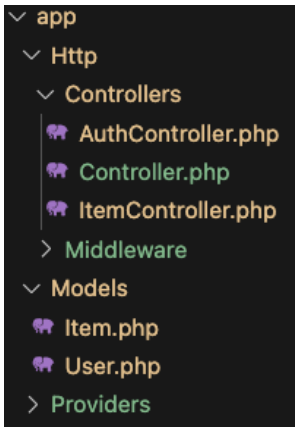


Figura 38 *Estructura carpeta app*

Dins de "app" els fitxers més importants són els models de "item" i de "user", els quals representen les entitats de la base de dades, s'utilitzen per interactuar amb les dades, com per exemple per validar si les credencials d'un usuari són correctes.

Els controladors són els fitxers on hi ha tota la lògica de les sol·licituds HTTP que rep l'aplicació. És on es troben tots els mètodes implementats per interactuar amb els diferent contenidors. En l'apartat 6.2.2. s'explicaran més concretament.

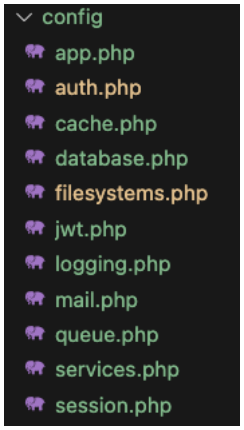


Figura 39 *Carpeta config al projecte Laravel*

En la carpeta "config", hi ha tots els fitxers de configuració de les diferents funcions del projecte. Els fitxers auth.php, filesystems.php i jwt.php han estat els més importants alhora del correcte funcionament de l'aplicació. En cada un d'ells es configura les diferents funcionalitats mitjançant les variables d'entorn configurades en el fitxer .env, d'entre altres paràmetres.

En la carpeta “routes” tenim el fitxer api.php.

```php
<?php

use App\Http\Controllers\AuthController;
use App\Http\Controllers\ItemController;
use Illuminate\Support\Facades\Route;


Route::post('/login', [AuthController::class, 'login']);

Route::post('/logout', [AuthController::class, 'logout'])->middleware('auth:api');

Route::post('/getUser', [AuthController::class, 'getAuthenticatedUser'])->middleware('auth:api');

Route::post('/register', [AuthController::class, 'register']);

Route::post('/newItem', [ItemController::class, 'submit'])->middleware('auth:api');

Route::get('/list', [ItemController::class, 'index']);

Route::get('/items/{id}', [ItemController::class, 'show']);

Route::put('/items/{id}/favorites', [AuthController::class, 'updateFavorites'])->middleware('auth:api');

Route::post('/items/{id}/installation', [ItemController::class, 'saveInstallation'])->middleware('auth:api');

Route::post('/items/{id}/screenshots', [ItemController::class, 'saveScreenshots'])->middleware('auth:api');

Route::post('/items/{id}/benchmarkText', [ItemController::class, 'saveBenchmark'])->middleware('auth:api');

Route::post('/items/{id}/benchmarkImage', [ItemController::class, 'uploadBenchmark'])->middleware('auth:api');

Route::delete('/items/{id}', [ItemController::class, 'destroy'])->middleware('auth:api');
```

Figura 40 *Fitxer api.php*

En aquest fitxer es defineixen les rutes específiques d’accés a l’API. Cada ruta indica el tipus de petició que accepta, el mètode del controlador on es dirigeix la petició i en els cassos necessaris, se li aplica un middleware d’autenticació que comprova si la sol·licitud conté el token corresponent per una major seguretat.

6.2.2. Funcions API

A l'API implementada en el backend, s'hi ha creat dos controladors un pels usuaris i l'altre pels ítems de l'aplicació. Aquests controladors seran els encarregats d'aplicar la lògica a cada petició.

Implementacions destacades AuthController.php

Aquest controlador és el que actuarà envers les peticions referents als usuaris. Conté cinc mètodes que aportaran les funcionalitats de registrar un nou usuari, d'iniciar sessió, tancar sessió, obtenir la informació de l'usuari actiu i finalment actualitzar la llista de preferits de l'usuari.

El mètode de registre s'ha implementat de la següent manera:

```php
// Funció per a registrar-se
public function register(Request $request)
{
    // Validació de les dades d'entrada
    Log::info('Dades rebudes:', $request->all());
    $validator = Validator::make($request->all(), [
        'name' => 'required|string|max:255',
        'email' => 'required|string|email|max:255|unique:users',
        'password' => 'required|string|min:6|confirmed',
        'role' => 'required|string|max:50',
    ]);


    if ($validator->fails()) {
        $errors=$validator->errors();
        Log::error('Errors de validació:', $errors->toArray());
        return response()->json($validator->errors(), 422);
    }

    // Creació d'un nou usuari
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
        'role' => $request->role,
        'favorites'=> []

    ]);

    // Generació d'un token JWT
    $token = JWTAuth::fromUser($user);

    return response()->json(compact('user', 'token'), 201);
}
```

Figura 41 *Implementació registre d'usuari*

Com es pot observar en la figura 40, el primer pas que es realitza és validar les dades rebudes i comprovar que segueixen les condicions requerides. Un cop les dades estan validades es crea l'usuari nou introduint les dades rebudes i modificant amb la funció de Hash la contrasenya per tenir una major seguretat. En el moment que es crea l'usuari, el mateix procés de creació guarda el contingut a la base de dades configurada anteriorment. Això, simplifica el procés ja que un cop configurat, s'automatitza molt el procediment.

El funcionament de l'aplicació és el següent en el moment que es vol crear un usuari nou:

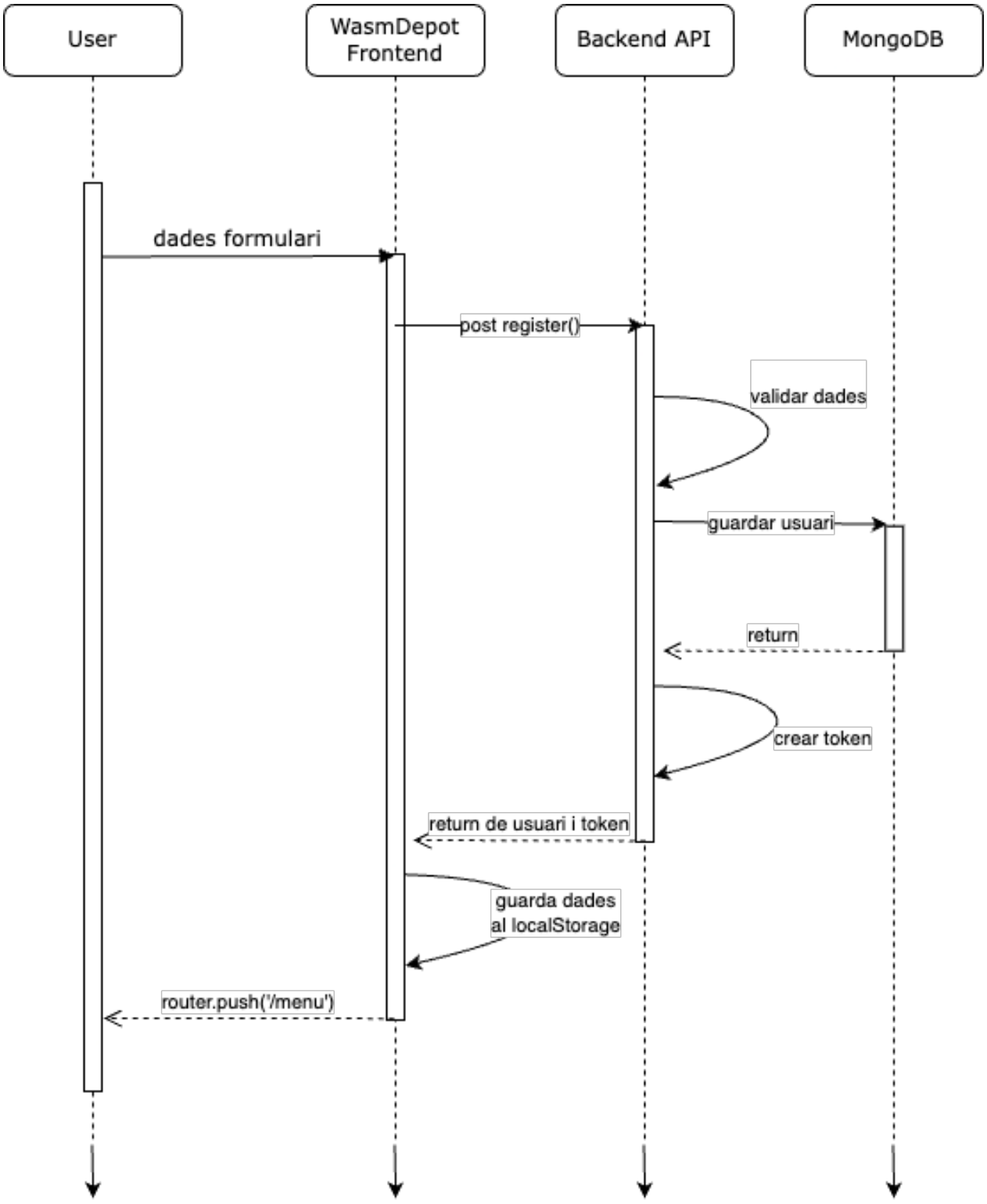


Figura 42 *Diagrama de seqüència funció registre*

El mètode de login implementat és:

```php
// Funció per a iniciar sessió
public function login(Request $request)
{
    // Credencials d'entrada
    $credentials = $request->only('email', 'password');
    Log::error('valors:', $credentials);
    try {
        // Intent d'iniciar sessió
        if (!$token = JWTAuth::attempt($credentials)) {

            return response()->json(['error' => 'Invalid credentials'], 401);
        }

    } catch (JWTException $e) {
        return response()->json(['error' => 'Could not create token'], 500);
    }

    // Retorna el token si l'autenticació és correcta
    return response()->json(compact('token'), 201);
}
```

Figura 43 *Implementació mètode login*

Quan arriba la petició de login, el primer pas és iniciar sessió mitjançant JWTAuth amb les credencials introduïdes, si són incorrectes es retorna error, en el cas de que siguin correctes i s'hagi iniciat la sessió, es retorna el token per poder-lo introduir a les sol·licituds posteriors.

Implementacions destacades ItenmController.php

El controlador "ItenmController" gestionarà les peticions envers als projectes que tinguem a l'aplicació. Hi ha mètodes per crear projectes nous, per eliminar-los i per modificar informació. La implementació per crear nous projectes és la següent:

```php
public function submit(Request $request)
{
    try {
        // Validació de dades
        Log::info('Request Data:', ['data' => $request->all()]);
        $request->validate([
            'title' => 'required|string|max:255',
            'sinopsis' => 'required|string',
            'description' => 'required|string',
            'image' => 'required|file|mimes:jpeg,png,jpg,gif,svg|max:2048',
            'web' => 'required|url',
            'github' => 'required|url',
            'categories' => 'required|string',
            'type' => 'required|string',
            'language' => 'nullable|string',
            'supportedPlatform' => 'nullable|string',
            'compilerToolchain' => 'nullable|string',
            'creatorId' => 'required|string',
            'creatorName' => 'required|string',
        ]);

        $url = 'http://localhost:9000/tfgbucket/';

        // Maneig de la imatge (si existeix)
        if ($request->hasFile('image')) {
            $imageName = $request->file('image')->store('images', 'minio');
            $imagePath = $url . $imageName;
        } else {
            $imagePath = null;
        }

        $formData = new Item();
        $formData->title = $request->title;
        $formData->sinopsis = $request->sinopsis;
        $formData->description = $request->description;
        $formData->image_path = $imagePath;
        $formData->web = $request->web;
        $formData->github = $request->github;
        $formData->categories = $request->categories;
        $formData->type = $request->type;
        $formData->language = $request->language;
        $formData->supportedPlatform = $request->supportedPlatform;
        $formData->compilerToolchain = $request->compilerToolchain;
        $formData->creatorId = $request->creatorId;
        $formData->creatorName = $request->creatorName;
        $formData->save();

        return response()->json(['message' => 'Item created successfully', 'itemId'

    } catch (\Exception $e) {
        Log::error('Error creating item', ['error' => $e->getMessage()]);

        return response()->json(['message' => 'Error creating item'], 500);
    }
}
```

Figura 44 *Implementació mètode submit*

Les dades rebudes es validen, es guarda la imatge a la carpeta "images" de MinIO mitjançant el mètode Store, configurat al fitxer filesystem.php anteriorment. Un cop s'ha guardat, retorna el nom assignat al fitxer i es crea el path per obtenir-lo. Per acabar, es crea un nou ítem amb tots els camps, guardant-lo automàticament a la base de dades MongoDB i es retorna al frontend.

El funcionament de la creació d'un nou projecte té les següents accions entre les diferents parts de l'aplicació:

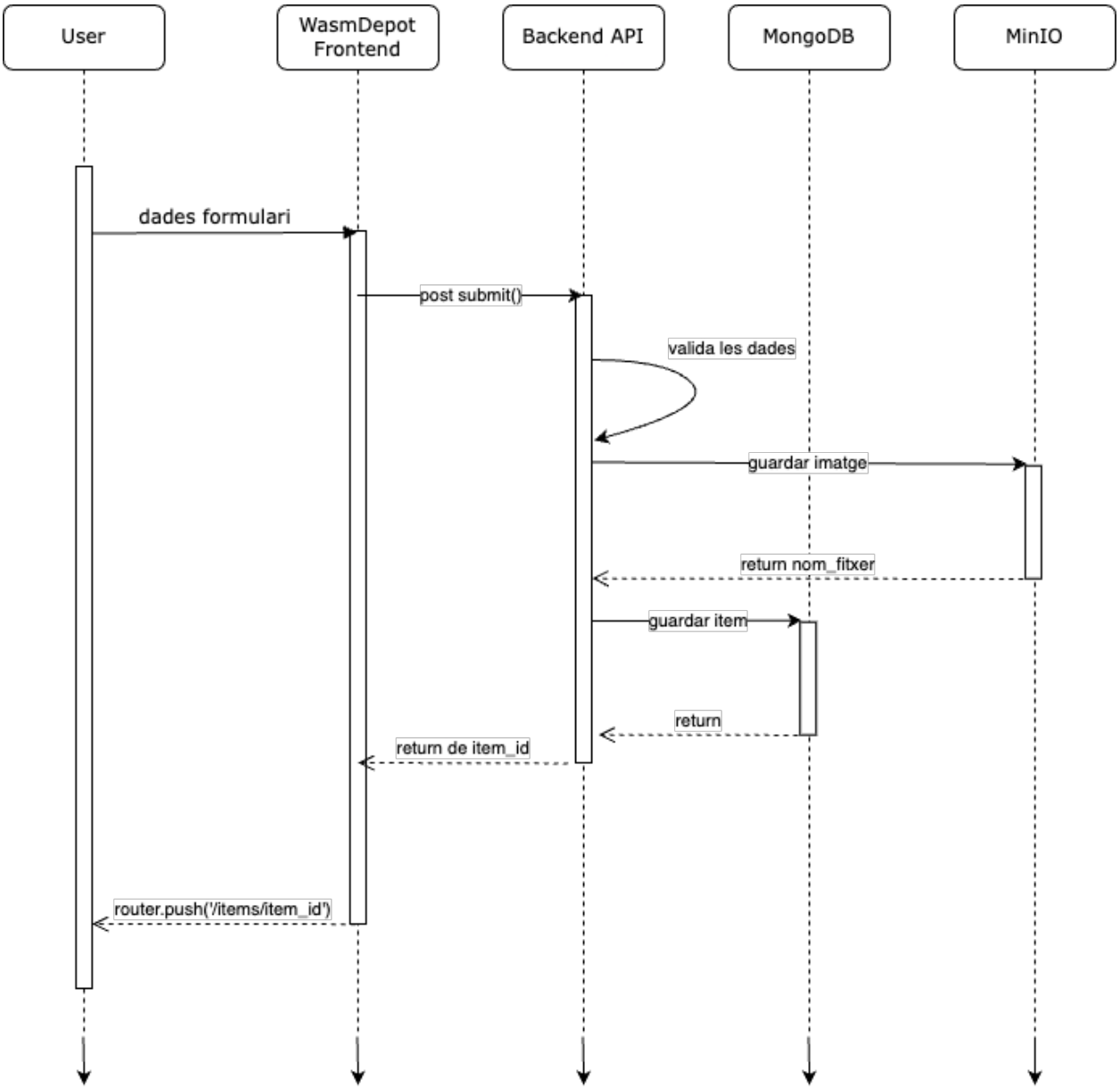


Figura 45 *Diagrama seqüència crear projecte nou*

A continuació, es mostra la implementació de la funció saveInstallation que permet modificar el contingut de la pestanya “Usage”:

```php
public function saveInstallation(Request $request, $id)
{
    try {
        Log::info('Request Data:', ['data' => $request->all()]);

        // Validació de dades
        $request->validate([
            'installation' => 'nullable|string',
            'file' => 'nullable|file|max:10240',
        ]);

        // Trobar l'element per ID
        $item = Item::findOrFail($id);

        // Actualitzar les instruccions d'instal·lació si arriben dades
        if ($request->filled('installation')) {
            $item->installation = $request->input('installation');
        }

        // Maneig del fitxer d'instal·lació si arriba un fitxer nou
        if ($request->hasFile('file')) {
            // Eliminar l'arxiu antic si existeix
            if ($item->file_path) {
                Storage::disk('minio')->delete($item->file_path);
            }

            // Guardar el nou fitxer
            $url = 'http://localhost:9000/tfgbucket/';
            $fileName = $request->file('file')->store('files', 'minio');
            $filePath = $url . $fileName;
            $item->file_path = $filePath;
        }

        // Guardar els canvis al model
        $item->save();

        return response()->json(['message' => 'Installation information saved successfully'],

    } catch (\Exception $e) {
        Log::error('Error saving installation info', ['error' => $e->getMessage()]);

        return response()->json(['message' => 'Error saving installation information'], 500);
    }
}
```

Figura 46 *Implementació mètode saveInstallation*

En aquest mètode, quan arriba la petició, es validen les dades. Seguidament, s’obté el projecte de la base de dades i es guarda en una variable. Es comprova si el camp “installation”, que és el text de la pestanya Usage, té contingut. En cas afirmatiu, s’actualitza l’ítem.

També es comprova si hi ha un nou fitxer a la petició, en cas que n’existeixi un, s’elimina el path de l’anterior, es guarda el nou fitxer a MinIO a la carpeta de “file” i es guarda el nou path del fitxer a la variable item, que posteriorment es guarda a la base de dades.

El diagrama de seqüència d'aquesta funcionalitat és el següent:

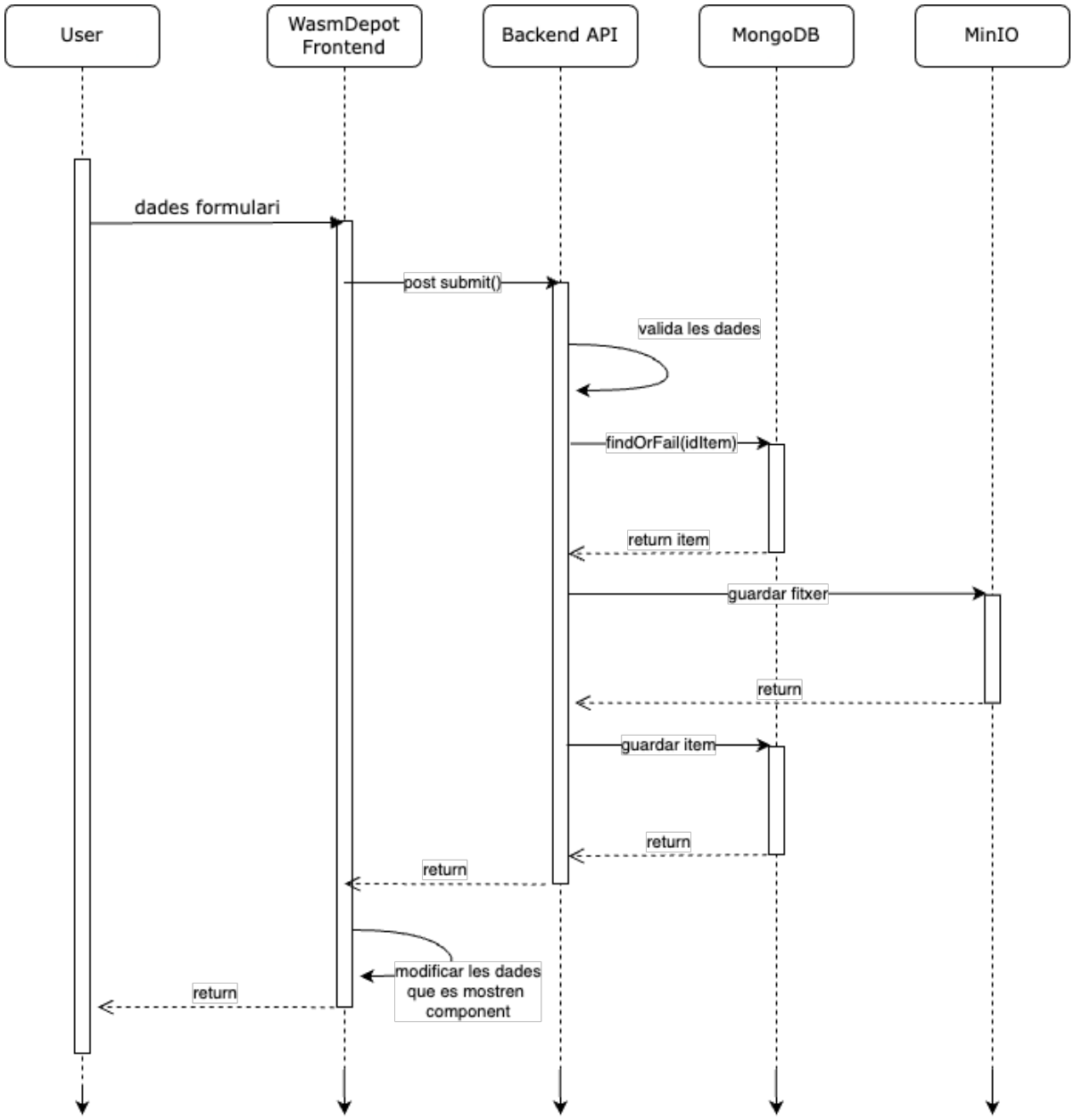


Figura 47 *Diagrama seqüència actualitzar contingut pestanya Usage*

6.2.3. Base de dades

Com s'ha explicat anteriorment, la base de dades de l'aplicació és MongoDB. Les dades, a MongoDB, es guarden en format BSON[6], una extensió binaria de JSON. Aquest format permet emmagatzemar estructures de dades complexes de manera natural en col·leccions. Cada col·lecció tindrà un conjunt de claus i valors, on els valors poden ser de tipus variat.

Per la implementació d'aquesta aplicació, han estat necessaris dos col·leccions. Una pels usuaris anomenada "users" i l'altre per els projectes de l'aplicació, anomenada "items".

Col·lecció Users

En aquesta col·lecció de la base de dades es guardarà tota la informació necessària dels usuaris.

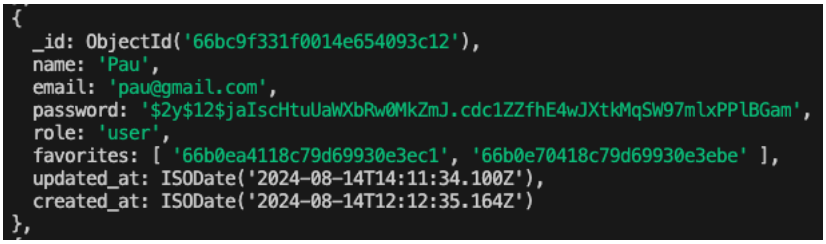


Figura 48 *Element de la col·lecció users*

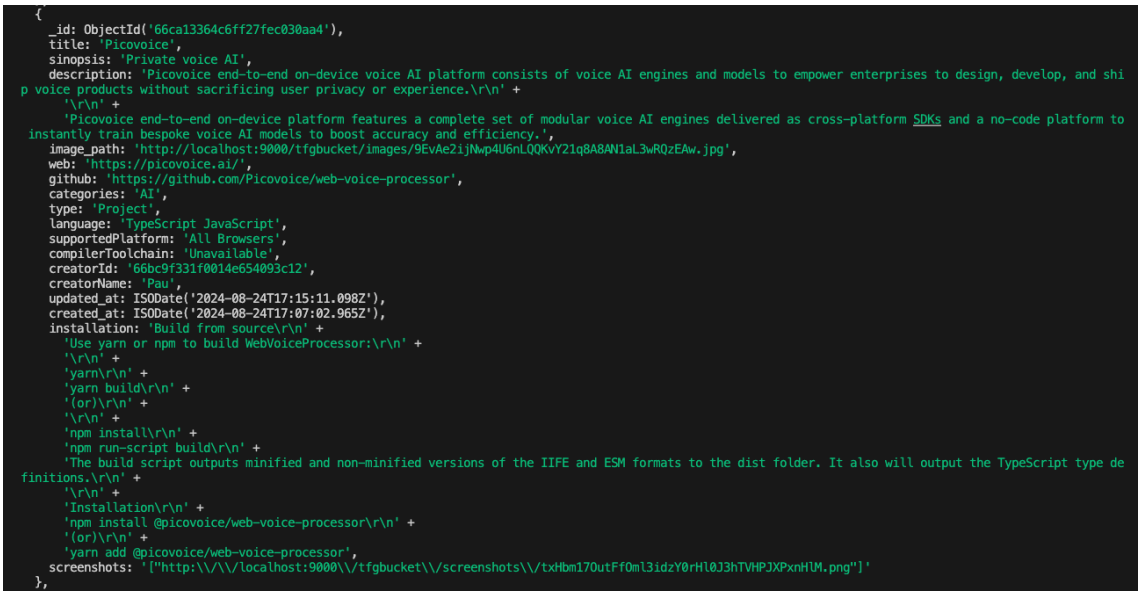
En la figura 47, podem observar el contingut d'un exemple d'usuari. Veiem els següents camps:

- id: identificador de l'usuari.
- name: nom de l'usuari registrat.
- email: correu electrònic de l'usuari, cada usuari ha de tenir el seu email únic, ja que serà imprescindible per l'autenticació.
- password: Hash de la contrasenya de l'usuari.
- role: rol de l'usuari, hi ha dos tipus, user o admin. L'admin tindrà permisos per modificar tots els projectes sense importar el creador.
- favorites: llista de id's de projectes que l'usuari té a preferits.
- updated-at: data de l'última modificació.
- created-at: data de creació de l'usuari.

[6] Binary JavaScript Object Notation

Col·lecció Items

En aquesta col·lecció de la base de dades es guardarà tota la informació necessària dels projectes que es vagin guardant a l'aplicació.



```
  {
    _id: ObjectId('66ca13364c6ff27fec030aa4'),
    title: 'Picovoice',
    sinopsis: 'Private voice AI',
    description: 'Picovoice end-to-end on-device voice AI platform consists of voice AI engines and models to empower enterprises to design, develop, and shi
p voice products without sacrificing user privacy or experience.\r\n' +
      '\r\n' +
      'Picovoice end-to-end on-device platform features a complete set of modular voice AI engines delivered as cross-platform SDKs and a no-code platform to
 instantly train bespoke voice AI models to boost accuracy and efficiency.',
    image_path: 'http://localhost:9000/tfgbucket/images/9EvAe2ijNwp4U6nLQQKvY21q8A8AN1aL3wRQzEAw.jpg',
    web: 'https://picovoice.ai/',
    github: 'https://github.com/Picovoice/web-voice-processor',
    categories: 'AI',
    type: 'Project',
    language: 'TypeScript JavaScript',
    supportedPlatform: 'All Browsers',
    compilerToolchain: 'Unavailable',
    creatorId: '66bc9f331f0014e654093c12',
    creatorName: 'Pau',
    updated_at: ISODate('2024-08-24T17:15:11.098Z'),
    created_at: ISODate('2024-08-24T17:07:02.965Z'),
    installation: 'Build from source\r\n' +
      'Use yarn or npm to build WebVoiceProcessor:\r\n' +
      '\r\n' +
      'yarn\r\n' +
      'yarn build\r\n' +
      '(or)\r\n' +
      '\r\n' +
      'npm install\r\n' +
      'npm run-script build\r\n' +
      'The build script outputs minified and non-minified versions of the IIFE and ESM formats to the dist folder. It also will output the TypeScript type de
finitions.\r\n' +
      '\r\n' +
      'Installation\r\n' +
      'npm install @picovoice/web-voice-processor\r\n' +
      '(or)\r\n' +
      'yarn add @picovoice/web-voice-processor',
    screenshots: '["http:\\/\\/localhost:9000\\/tfgbucket\\/screenshots\\/txHbm17OutFfOml3idzY0rHl0J3hTVHPJXPxnHlM.png"]'
  },
```


Figura 49 *Element de la col·lecció items*

La col·lecció Items té els següents camps:

- id: identificador del projecte.
- title: títol del projecte.
- sinopsis: descripció curta del projecte
- description: explicació del projecte
- imatge_path: path per obtenir la imatge.
- file_path: path per obtenir el fitxer guardat.
- web: URL de la web del projecte.
- github: URL del GitHub del projecte.
- categories: categoria de projecte WebAssembly.
- type: tipus de contingut guardat.
- language: llenguatges que utilitza el projecte.
- supportedPlatform: plataformes on es pot executar.
- compilerToolchain: compilador del projecte.
- creatorId: id de l'usuari creador.
- creatorName: nom de l'usuari creador.
- installation: text que es mostra a la pestanya Usage
- screenshots: objecte JSON que conté els paths de totes les screenshots.
- benchmark: text que es mostra a la pestanya Benchmark
- benchmarks: objecte JSON que conté els paths de totes les imatges de la pestanya benchmark.
- updated-at: data de l'última modificació.
- created-at: data de creació de l'usuari.

6.2.4. Emmagatzematge de fitxers

Una de les funcions principals que té l'aplicació, es poder centralitzar el contingut WebAssembly. El contingut es guarda en un contenidor amb una imatge MinIO.

Creació del bucket

Un cop tenim configurat el contenidor, s'han de seguir uns passos per tal de crear un bucket, que és on guardarem tot el contingut de l'aplicació, ja sigui imatges o fitxers. Aquest bucket es crearà seguint els següents passos:

- Primerament, accedirem des del navegador, a la interfície web del nostre contenidor MinIO, posant la direcció "localhost:9001".
- Introduïm les credencials que hem configurat en el Docker Compose.
- Un cop accedim, podem crear un nou bucket, li posem el nom i el creem.
- Després s'ha de crear un usuari amb tots els permisos.
- Quan hagis creat l'usuari, has de crear una clau d'accés. Obtindràs dos codis, un identificador i la contrasenya de l'usuari. S'han de guardar ja que no es podran accedir posteriorment.
- Poses el bucket en estat públic.
- Finalment, les credencials del usuari creat les posem al fitxer .env del projecte Laravel

Si en algun moment, per exemple, es volgués canviar el sistema d'emmagatzematge de fitxers a Amazon S3, no hi hauria cap problema ja que funcionen de la mateixa manera i per tant, només s'hauria de configurar amb les noves credencials.

Contingut del bucket

Dins del bucket creat, s'hi guardaran tots els recursos necessaris del projecte, ja siguin fitxers, imatges, vídeos, etc. Per tenir-ho d'una manera ordenada, s'ha implementat un sistema de fitxers dividit en diverses carpetes, on cada carpeta tindrà un contingut específic. Facilitant així la labor de manteniment. Les diferents carpetes són les següents:

Figura 50 *Directori del Bucket utilitzat per l'aplicació*

A la carpeta "assets" hi ha tots els recursos que necessita la pàgina a nivell visual, ja siguin vídeo o imatges de la interfície. A les carpetes "benchmarks" i "screenshots" hi ha les imatges referents a les pestanyes respectives. La carpeta "images" conté les imatges principals. Finalment, la carpeta "files", conté els fitxers dels projectes, que més tard es podran descarregar si algun usuari ho desitja.

7. Avaluació

Per determinar si l'aplicació desenvolupada compleix els requisits funcionals i no funcionals establerts durant la fase de planificació, es procedeix a realitzar la verificació del compliment dels requisits.

Aquesta avaluació es important per assegurar la qualitat i fiabilitat del sistema, així com, identificar possibles millores. Això permetrà obtenir una visió clara del resultat obtingut i analitzar si l'aplicació satisfà les necessitats per a les quals ha estat dissenyada.

7.1. Requisits funcionals

Requisits usuari no registrat

Requisit	Descripció	Resultat
00.Registre-se	L'usuari s'ha de poder registrar per primera vegada	✅ Usuari registrat correctament
00.Registre-se	Intentar registrar-se un cop ja s'ha registrat	✅ No permet el registre
00.Registre-se	Registrar-se amb algun camp buit	✅ Notifica que els camps són obligatoris
00.Registre-se	Intentar registrar-se sense complir les condicions de cada camp	✅ Indica el camp del formulari que no compleix les condicions
01. Iniciar Sessió	Iniciar sessió amb les dades correctes	✅ Inici de sessió exitós
01. Iniciar Sessió	Iniciar sessió amb les dades incorrectes	✅ No permet el inici de sessió, missatge credencials incorrectes
01. Iniciar Sessió	Iniciar sessió amb camps buits	✅ Notifica que els camps són obligatoris
01. Iniciar Sessió	Iniciar sessió sense complir les condicions del camp	✅ Notifica que no compleix les condicions

Taula 1 *Requisits mètodes per iniciar sessió*

Requisit	Descripció	Resultat
02.Navegació	L'usuari ha de poder navegar per l'aplicació	✅ Navegació correcte
02.Navegació	L'usuari ha de poder veure les opcions de Login i Sign up	✅ Els botons de Login i Sign Up es mostren quan l'usuari no està iniciat
03. Visualització de projectes	L'usuari ha de poder veure tots els projectes de l'aplicació	✅ Es veuen tots els projectes independentment de si estàs logejat
03. Visualització de projectes	L'usuari ha de poder utilitzar tots els tipus de filtres per navegar	✅ Els filtres funcionen tots correctament independentment de si estàs logejat
04. Visualització detalls del projecte	Al clicar a una targeta es mostren els detalls d'aquell projecte	✅ Permet visualitzar els detalls del projecte
04. Visualització detalls del projecte	Dins del component detalls es pot navegar per les diferents pestanyes	✅ Permet navegar per les pestanyes
05. Descarregar fitxer	Si el projecte té un fitxer, permet descarregar-lo	✅ Permet la descarrega independentment de si estàs logejat
05. Descarregar fitxer	Si el projecte no té fitxer, no mostrarà l'opció de descarregar	✅ No mostra el botó

Taula 2 *Requisits mètodes usuari no logejat*

Requisits usuari registrat

Requisit	Descripció	Resultat
06. Tancar Sessió	L'usuari ha de poder tancar sessió	✅ Final de sessió exitós
07. Creació de Projecte Nou	S'ha de mostrar el botó per crear projecte nou a la toolbar	✅ El botó es mostra correctament quan hi ha un usuari actiu
07. Creació de Projecte Nou	Crear un projecte amb els camps correctament	✅ El projecte es crea correctament
07. Creació de Projecte Nou	Crear projecte sense omplir tots els camps	✅ No permet la creació del projecte amb camps buits
08. Navegació Usuari Registrat	Es mostra la pestanya byMe al toolbar quan hi ha un usuari registrat	✅ La pestanya es accessible quan hi ha un usuari registrat
08. Navegació Usuari Registrat	No es mostra la pestanya byMe al toolbar quan no hi ha un usuari registrat	✅ La pestanya no es accessible quan no hi ha un usuari registrat
09. Llista preferits	S'ha de poder afegir els projectes a la llista de preferits	✅ S'afegeix el projecte correctament al clicar el botó
09. Llista preferits	S'ha de poder treure els projectes a la llista de preferits	✅ S'elimina el projecte de la llista al clicar el botó
09. Llista preferits	Un cop tanques sessió i tornes a entrar has de tenir la llista de preferits igual	✅ Els preferits del usuari es mantenen tot i tancar sessió
09. Llista preferits	Cada usuari té la seva llista personalitzada	✅ Cada usuari té la seva pròpia llista
10. Modificar contingut pestanya	Si l'usuari que accedeix al projecte és el creador, ha de poder modificar el contingut de les pestanyes	✅ L'usuari creador pot modificar el contingut
10. Modificar contingut pestanya	Si l'usuari que accedeix al projecte no és el creador, no pot modificar	✅ No es pot modificar dades, només visualitzar-les i descarregar fitxer
10. Modificar contingut pestanya	El contingut modificat es mostra amb el mateix format	✅ Es respecten els espais, salts de línia, etc. a l'editar el contingut

Requisit	Descripció	Resultat
10. Modificar contingut pestanya	Es poden afegir imatges il·limitades a les pestanyes que ho permeten	✅ Les imatges es guarden correctament
11. Eliminar Projecte	Si l'usuari és el creador del projecte ha de poder eliminar-lo	✅ El projecte s'elimina correctament
11. Eliminar Projecte	Si l'usuari no és el creador del projecte no el podrà eliminar	✅ No es mostra l'opció d'eliminar projecte

Taula 3 *Requisits mètodes usuari logejat*

Requisits Usuari Admin

Requisit	Descripció	Resultat
12. Modificar projectes Admin	L'usuari amb rol d'admin ha de poder modificar tots els projectes com si fos el creador	✅ L'usuari admin pot modificar tots els projectes
13. Eliminar projectes Admin	L'usuari amb rol d'admin ha de poder eliminar qualsevol projecte	✅ L'usuari admin pot eliminar qualsevol projecte

Taula 4 *Requisits mètodes usuari admin*

7.2. Requisits no funcionals

Un cop s'ha verificat que els requisits funcionals estan complerts, també s'han de comprovar els requisits no funcionals. Per avaluar-ho, analitzarem els requisits establerts anteriorment i comprovarem si s'han complert

Requisits d'usabilitat

Els requisits d'usabilitat bàsicament es centraven en que l'aplicació fos intuïtiva per tots els usuaris, fàcil d'aprendre on buscar cada cosa i amb un feedback al fer accions per tal de mostrar al client si tot havia anat correcte o hi havia algun error.

Els requisits d'usabilitat han estat coberts satisfactòriament. Ho podem afirmar ja que, tant amb el disseny del frontend, fent l'aplicació atractiva pels usuaris, com les implementacions dels menús, filtres i les respostes al fer accions, l'usuari en poc temps pot tenir un gran domini del funcionament de l'aplicació. S'ha testat amb diferents perfils d'usuari i les valoracions han estat positives.

Requisits de rendiment

Els requisits de rendiment fan referència a l'eficiència de la pàgina web.

Analitzant una mica l'experiència d'ús com a usuari i sabent l'arquitectura aplicada al projecte, podem confirmar que els requisits de rendiment estan superats.

Ho podem afirmar ja que els temps de resposta de l'aplicació són insignificants i l'usuari no tindrà una mala experiència a nivell de temps d'espera.

list	204	preflight	Preflight	0 B	121 ms
list	200	xhr	index.js:62	33.0 kB	137 ms
9EvAe2ijNwp4U6nLQQKvY21q8A8AN1aL3w...	200	jpeg	VImg.js:200	(disk cache)	1 ms
66ca13364c6ff27fec030aa4	204	preflight	Preflight	0 B	150 ms
66ca13364c6ff27fec030aa4	200	xhr	index.js:62	1.9 kB	152 ms
txHbm17OutFfOml3idzY0rHl0J3hTVHPJXPx...	200	png	VImg.js:200	104 kB	10 ms
list	204	preflight	Preflight	0 B	172 ms
list	200	xhr	index.js:62	33.0 kB	166 ms
ID2GQzdY2O019tCQnVJgNcoMaQleEaSamT...	200	png	VImg.js:200	(disk cache)	1 ms
19GkkeD85NoDVWn6n8u2LlvK4AO7Jirhw3...	200	png	VImg.js:200	(disk cache)	2 ms
list	204	preflight	Preflight	0 B	157 ms
list	200	xhr	index.js:62	33.0 kB	165 ms
ovWZQRb4aEhaqky50yW2f5c78tLI4g4bcDt...	200	png	VImg.js:200	(disk cache)	1 ms
j4Gq1mOwBgCnV9VcgpLoQJA47GWSbJqz...	200	png	VImg.js:200	(disk cache)	1 ms

Figura 51 *Temps d'espera navegació per l'aplicació*

Mitjançant l'arquitectura implementada també ens permet l'escalabilitat fàcilment si ho necessitéssim.

Requisits de seguretat

S'ha aplicat metodologies de autenticació i autorització. També, s'han protegit les dades sensibles com ara contrasenyes utilitzant codificació amb hash.

En aquest sentit, donem per completats els requisits de seguretat, tot i que sabem que en aquest camp tenim molt per millorar, aplicant més metodologies de seguretat.

Requisits de manteniment

Els requisits de manteniment, són les característiques que facilitaran la feina posterior.

Confirmar que els requisits establerts han estat complerts ja que s'ha documentat les parts importants del codi, i mitjançant l'arquitectura de microserveis aconseguim tenir totes les parts de l'aplicació funcionant independentment. També, es facilita molt l'escalabilitat en un futur si es necessari.

La utilització de MinIO, per exemple, ens permet canvia a Amazon S3 sense modificar codi.

7.3. Resultats finals

El resultat final obtingut del desenvolupament d'aquesta aplicació, en general ha estat molt bo. Podem dir que s'ha aconseguit l'objectiu de crear una aplicació per centralitzar l'ecosistema del WebAssembly, sent atractiva visualment , intuïtiva i amb un bon rendiment.

Aquí adjunto un link on es pot veure una demo explicada de l'aplicació:

- https://drive.google.com/file/d/1u07PiwwmRT4cckAjRbeyZwSiYCQXDdq4/view?usp=sharing

8. Conclusions

El principal objectiu d'aquest projecte era crear una aplicació web que permetés centralitzar el contingut de WebAssembly, estandarditzar la informació al respecte i mostrar aquesta informació d'una forma visualment agradable pels usuaris.

El procés de creació de l'aplicació, se m'ha fet molt enriquidor a nivell de coneixement en diferents camps.

Primerament, he après sobre que és el WebAssembly, el seu funcionament i les aplicacions útils que té i que pot tenir en un futur.

També, he trobat molt enriquidor els coneixements que he obtingut sobre l'arquitectura de microserveis, barallant-me amb les configuracions per aconseguir que funcionés tot correctament i descobrint els grans avantatges que aporta la seva implementació.

Per acabar, he pogut adquirir coneixements durant la implementació del codi en els diferents llenguatges, funcionalitats de paquets i la utilització de MinIO i la base de dades.

Un cop he passat per tot aquest procés i veig el resultat final de l'aplicació, puc confirmar que s'ha aconseguit l'objectiu del projecte i espero que sigui d'ajuda per la comunitat de WebAssembly i el grup de recerca CloudLab, que van ser ells qui em van informar de la carència d'una eina d'aquest tipus.

Aspectes a millorar en un futur

Durant la implementació de tota l'aplicació m'he adonat que es podrien incorporar nous aspectes per perfeccionar l'experiència d'ús, la seguretat o afegir noves funcionalitats a l'aplicació.

Destacar a nivell de seguretat, la possibilitat d'implementació de sol·licituds HTTPS, encriptació de dades o implementació de més mètodes de seguretat com ara verificació en dos passos.

Per altre banda, es podrien incorporar nous apartats a la web on es pretengués tindre una finalitat més formativa, per ajudar als usuaris a introduir-se en l'ecosistema WebAssembly.

A nivell de disseny i navegació per la web, es podria estudiar noves implementacions segons les necessitats que vagin sorgint dels usuaris.

9. Annex

Posada en marxa de l'aplicació

Per desplegar l'aplicació en un ordinador s'han de seguir una sèrie de passos molt senzills. Aquesta facilitat de desplegament es gràcies a la dockerització de l'aplicació. Aquest passos són:

- Instal·lar Docker i Docker Compose al ordinador.
- Descarregar el codi de l'aplicació.
- Executa "docker-compose up -d" per desplegar l'aplicació.
- Verificar que tot funcioni bé.
- Accedir des del navegador a la direcció "localhost"

El codi de l'aplicació està en aquest repositori de GitHub

- https://github.com/Saluchoooo/WasmDepot