UNIVERSITAT DE BARCELONA

Treball de Fi de Grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica

Universitat de Barcelona

Disseny i implementació d'un sistema de senyalització digital per al CRAI Biblioteca de Matemàtiques i Informàtica II

Francina Pons Llabrés

Tutors: Puertas i Prats, Eloi

Angela i Gambús, Roger

Realitzat a: Departament de Matemàtiques i Informàtica

Barcelona, 13 de juny de 2022

Resum:
Aquest TFG és una continuació d'un altre TFG i el que es proposa és millorar l'actual sistema de senyalització digital amb programari lliure per a una TV ubicada al vestíbul del CRAI. Aquest sistema és obra de Vicent Nuñez i ja està en funcionament. L'objectiu és donar-li més funcionalitats i fer-hi algunes millores perquè sigui més atractiu i el puguin utilitzar més biblioteques.
El sistema en qüestió permet alternar diversos modes de reproducció en funció de les necessitats de cada moment i facilita una gestió àgil dels continguts a través de la xarxa. També permet crear i modificar llistes de reproducció per tenir el contingut més ben organitzat.

Resumen:
Este TFG es una continuación de otro TFG i lo que se propone es mejorar el actual sistema de señalización digital con programario libre para una TV ubicada en el vestíbulo del CRAI. Este sistema es obra de Vicent Nuñez i ya está funcionando. El objetivo es darle más funcionalidades al sistema y añadir algunas mejoras para que sea más atractivo y se pueda utilizar en más bibliotecas.
El sistema en cuestión permite alternar diversos modos de reproducción en funcion de las necesidades de cada momento y también facilita una gestión ágil de los contenidos a través de la red. También permite crear y modificar listas de reproducción para tener el contenido más organizado.

Abstract:
This TFG is a continuation of another TFG and the proposal is to improve the current digital signage system with free software for a TV located in the CRAI lobby. This system is the work of Vicent Nuñez and is in operation. The goal is to give it more functionality, make some improvements, and to make it more attractive and usable for more libraries.
The system in question allows for different playback modes to be alternated depending on the needs of each moment and facilitates agile management of content through the network. It also allows for the creation and modification of playlists so that the content is better organised.

Agraïments

Vull agrair principalment a l'Eloi i el Roger per deixar-me formar part d'aquest projecte i per totes les idees i recomanacions que m'han fet al llarg de tot el procés. També vull mencionar al Vicent, que és qui va crear aquesta aplicació i va deixar les coses preparades per una ampliació futura.

A més, he d'agrair a la meva família que m'hagin acompanyat durant tota aquesta etapa, als meus amics i els meus companys de pis, que són els que més m'han motivat i animat a finalitzar aquest projecte.

Moltes gràcies també als professors perquè han fet que cada cop m'agradi més el que faig i que em vulgui dedicar a això en la meva vida professional.

Moltes gràcies a tots!!

Índex

Índex de Figures

Índex de Taules

Introducció

“Una imatge val més que mil paraules”.

Quan pensem en una biblioteca ens venen al cap llibres tradicionals, enciclopèdies, silenci, etc. Però avui en dia les tecnologies de la informació i les comunicacions estan presents a quasi tots els àmbits, incloses les biblioteques. Amb aquest moviment tan ràpid cap a la digitalització i la presència de la Covid en els últims anys, avancem a passos agegantats cap a un món de pantalles i recursos digitals. És d’aquí d’on surt la idea de crear un sistema de senyalització digital, que sigui pràctic i útil tant per al personal de la biblioteca com per a la gent que hi passa i veu el contingut.

Objectiu del projecte

El principal objectiu del projecte és obtenir un programari per reproduir arxius multimèdia al vestíbul de la biblioteca. Aquest projecte en realitat és una continuació d’un altre projecte de final de grau fet per l’estudiant Vicent Núñez (Núñez Delgado, 2021). Es tracta d’una eina que actualment es troba en funcionament a la biblioteca de la Facultat de Matemàtiques i informàtica i que s’utilitza a l’espai del vestíbul per ensenyar contingut multimèdia a la gent que hi passa. Està connectat a una pantalla situada a l’entrada del vestíbul i el personal de la biblioteca pot triar quin contingut mostrar-hi a través d’aquesta eina.

L’objectiu és millorar i afegir funcionalitats a l’aplicació segons els requeriments per tal que més biblioteques puguin gaudir d’aquest servei tan necessari.

Motivació

Triar el tema del treball de recerca no és fàcil. La llista de propostes és llarga i no sempre tens prou clar cap a on et vols encaminar. En el meu cas, quan vaig llegir el títol que el professor Eloi Puertas havia inclòs a la llista em va cridar l’atenció i vaig pensar que podria ser un bon començament. A partir d’aquí, i un cop formalitzada la sol·licitud, ell em va explicar amb més de detall de en què consistia el projecte i em va oferir l'ajuda i el suport del Roger Angela, membre del CRAI Biblioteca de Matemàtiques i Informàtica. Amb ells dos vam decidir els objectius i vam plantejar com s'hauria de continuar aquest projecte.

Quan em vaig plantejar l'orientació que havia de tenir aquest treball vaig tenir clar que havia de ser funcional, útil. Volia que fos un producte que la gent pogués fer servir i que no només estés centrat en la investigació i aspectes teòrics. Per tant, una manera era la de contribuir a la digitalització d'espais que encara se'n poguessin beneficiar. Donat que, com podem veure, els cartells publicitaris en paper són cada vegada obsolets i es van canviant per pantalles, vaig considerar que aportar eines tecnològiques per incloure aquesta informació tradicionalment impresa seria una bona solució. Un clar exemple d'aquest canvi cap a la digitalització el trobem al transport públic on abans estava ple de cartells que ara han estat substituits per elements visuals digitals (Metrópoli, 2022 i Digital AV Magazine - Underwood Comunicación SL, 2022).

En aquest cas concret, l'aplicació del meu treball està destinada, en primer lloc, als treballadors del CRAI, que disposaran d'una manera àgil per mostrar el contingut que volen fer arribar als usuaris de la biblioteca. I, en segona instància, els que es beneficiaran d'aquest producte seran els estudiants i professors que accedeixin a la biblioteca, ja que amb un cop d'ull a la pantalla podran informar-se dels esdeveniments propers, d'horaris, etc. que es duguin a terme a la Facultat.

Per altra banda, la motivació ve donada perquè de totes les aplicacions de la informàtica que hem tractat durant la carrera, la que més m'ha cridat l'atenció últimament ha estat la de Desenvolupament Web. Després de les assignatures d'Enginyeria de Software, Software Distribuït i Computació Orientada a Web vaig decidir que la temàtica del meu TFG havia de consistir en l'elaboració d'una aplicació web. A més, degut a que el meu objectiu és dedicar-me al desenvolupament de software, vaig pensar que seria un molt bon començament per aprendre i comprovar que realment és el que m'agrada. Per aquest mateix motiu també vaig decidir fer les pràctiques a una empresa que es dedica a crear i mantenir software anomenada JES Serveis Informàtics.

Un detall important que cal comentar també és que aquest treball és una continuació del TFG de Vicent Núñez, la qual cosa fa que el projecte sigui més interessant perquè no podia saber què em trobaria exactament en el moment de veure el seu codi ni quins serien els passos a seguir, però això també ofereix un avantatge

perquè la configuració de l'entorn del projecte ja quedava resolta. La veritat és que encara no m'havia trobat mai amb la situació de veure un codi d'una tercera persona (que no fossin de professors o codis d'internet bastant ben estructurats). La idea de continuar el codi d'una altra persona em va semblar un bon punt de partida i la vaig entendre com una oportunitat, ja que vaig pensar que seria un assaig d'allò que m'aniré trobant en el món laboral. Així doncs, quina millor manera d'aprendre que observant la feina d'altres professionals?

Estructura de la memòria

El contingut d'aquest document es troba organitzat en els següents capítols:

- Introducció: Context de l'aplicació, motivació i objectius d'aquest Treball de Final de Grau.
- Planificació: un breu resum de la planificació inicial i la distribució de les tasques i de la planificació final.
- Anàlisis: Plantejament i explicació. Anàlisi des del punt de vista del client com a usuari i també dels requeriments en format de *user stories*.
- Disseny: Explicació de l'arquitectura i plantejament dels diferents aspectes del sistema desenvolupat.
- Implementació: Explicació en profunditat dels algoritmes desenvolupats que conformen el sistema de reproducció.
- Manual d'instal·lació: breu explicació per a que els usuaris finals siguin capaços d'utilitzar l'aplicació.
- Resultats: Exposició de les proves realitzades al sistema finalment elaborat.
- Conclusions: Capítol explicatiu sobre les possibles ampliacions realitzables al sistema i les conclusions extretes del projecte.

Planificació

Per continuar aquest projecte de Vicent Núñez primer de tot havia d'entendre com funcionava, per a què servia exactament i com estava fet. Per això es va organitzar una reunió on l'Eloi i en Roger em van explicar els detalls del programari. L'Eloi em va guiar per entendre el codi existent i em va explicar que el projecte estava dividit

en dues parts de les quals es fa esmena més endavant: el Controlador i la Cartellera. En Roger, per altra banda, em va detallar l'ús que li estaven donant actualment a l'aplicació i com era el funcionament real.
Vaig entendre la planificació inicial com si un client m'hagués fet un encàrrec que havia de desenvolupar: l'Eloi i en Roger, que tenien moltes idees i moltes ganes, em van exposar el seu punt de vista i jo vaig escoltar-los. Després vam discutir quines eren les idees més importants i quines havíem d'incorporar. Vam fer una pluja d'idees seguint les necessitats que tenia l'aplicació i després vam prioritzar-les i vam fer-ne una estimació segons l'ús real i el temps de desenvolupament. A continuació, s'ofereix un resum d'aquestes idees que van sortir en la primera reunió i que es veuran més en detall en els requeriments i en els casos d'ús presents durant aquest document:

Bugs:

- Es vol assegurar que tot el que hi ha a la SD és en format UTF-8 perquè això assegura que el mapejat entre el nom en la base de dades i el fitxer real es fa correctament.

Idees per part del *Backend*:

- Crear i guardar llistes amb un nom. Es vol tenir una carpeta de llistes on es recuperin tant els arxius com la configuració (durada, prioritats, etc).
- Eliminar la llista desitjada i tot el seu contingut.
- Etiquetar aquestes noves llistes amb etiquetes (tags) per tal de poder classificar-les millor.
- Afegir filtre per etiquetes a l'hora de mostrar les llistes disponibles
- Marcar alguns arxius com a favorits per tal que es vagin reproduint més sovint (per exemple amb un sistema d'estrelles 1-5 o alguna cosa així)
- Millorar la classificació i la cerca de fitxers a l'hora d'afegir contingut.
- Eliminar id's de les llistes de l'aplicació de control ja que no donen informació a l'usuari.
- Afegir visualització de documents (Power Point, PDF, etc). Fins ara els tipus de fitxers suportats eren imatges i vídeos. Es volia afegir algun tipu de fitxers

i/o convertir-los en aquests formats per reproduïr-los. Per exemple, transofrmar un arxxiu en format PDF a fotos i anar passant les pàgines.

- Mirar si existeix alguna API o línia de comandes per convertir pàgines d'un document en format PDF o Power Point a imatges individuals. Preguntar a l'usuari el tipus de conversió i el temps per cada diapositiva. Mirar si és més fàcil de fer a l'hora d'afegir el contingut o a l'hora de visualitzar-lo.
- Eines per l'usuari o decisions de l'aplicació. Crides a algunes comandes perquè es puguin reiniciar o apagar tant el Docker com la Raspberry.
- Eina per modificar els fitxers: canvi de nom des de la pròpia aplicació. Quan es selecciona un arxiu, mostrar opcions: canviar nom, canviar etiqueta. No cal modificar el fitxer en local, modificar només el nom del fitxer i el nom per mostrar. És a dir, canviar els noms però mantenir el mapejat amb l'arxiu original amb el nom original.
- Veure la durada dels vídeos a la taula de la llista de reproducció; ara només la mostra pels fitxers estàtics. S'hauria doncs d'extreure aquesta informació dels vídeos amb nous algorismes combinats amb llibreries externes.
- Guardar a la base de dades la longitud del vídeos. Fins ara es reprodueixen sense saber la duració d'aquests. Buscar una llibreria Python per treure aquesta dada. Per recomanació del tutor, millor no utilitzar OpenCV. Utilitzar en el seu defecte la llibreria Ffmpeg.
- Sistema Agenda que et permeti programar continguts per un dia i hora determinats. Quan s'afegeix a la cua de reproducció, poder decidir quan afegir-lo.
- Taula a la base de dades amb arxiu multimèdia i moment de la reproducció per mantenir un historial de reproduccions.

Idees per part del *frontend*:

- Crear un calendari perquè l'usuari pugui consultar si hi ha cap contingut en un moment determinat.
- Tenir la possibilitat d'afegir, modificar i esborrar el contingut d'aquest calendari.
- Aplicació per reiniciar la Raspberry des de *frontend*. Afegir avisos per informar a l'usuari que està executant una instrucció perillosa.

Com podem observar en la llista d'idees que vam tenir, vam ser molt ambiciosos amb el projecte i el temps per fer-lo i a més vam subestimar algunes de les dificultats que presenten. És per aquest motiu que no s'han pogut dur a terme tots els objectius de la planificació.
En poc temps vam anar veient que seria pràcticament impossible afegir totes aquestes funcionalitats a l'aplicació existent i es van haver de reduir els objectius. Aquesta mesura s'ha hagut de fer principalment perquè el codi existent no estava preparat per un desenvolupament en local, ja que es feia tot a través de Dockers. Quan vaig voler canviar aquesta metodologia de treball em vaig trobar alguns problemes perquè vaig haver d'afegir algunes configuracions per fer el *hot-reload* del *frontend*. També vaig haver de crear una base de dades nova i adaptar els models i finalment a l'hora del pas a Docker es van haver de resoldre alguns errors deguts al canvi d'organització de directoris.

Calendari

Com que es tracta d'un treball de final de grau, el temps per fer tant el projecte com la memòria corresponent és de 3 mesos. S'havia previst dividir aquest temps en *sprints* de 15 dies per posar en comú la feina feta amb els tutors. Aquestes reunions s'han anat fent bastant sovint i segons la quantitat de feina s'han anat incorporant o eliminant alguns dels requisits, i s'han prioritzat els més necessaris.
Per la part de programació, s'han dividit les tasques per fer aquest projecte de forma més eficient. Com que el que més em preocupava era la part de *backend*, vaig decidir començar per aquesta part i després dedicar temps al *frontend* i a la part estètica. També cal mencionar que m'ha sigut proporcionada una Raspberry Pi per a que pogués anar provant l'aplicació a "producció" en un entorn igual que el que hi haurà a la biblioteca on s'utilitzarà.

Anàlisi

Com hem dit, aquest treball és una continuació d'un altre projecte que ja era 100% funcional i, per tant, vaig optar per intentar només afegir funcionalitats, no modificar

les que ja hi havia en la mesura del possible. A més he pogut aprofitar tota la configuració de la Raspberry i dels Dockers, exceptuant alguns canvis.

Requeriments

A partir de les primeres reunions van sortir un llistat de requeriments que amb els temps han anat variant i se n'han eliminat o se n'hi han afegit de nous.
Els principals requeriments que vam concertar pensant amb la millora de l'aplicació són:

- El Controlador ha de permetre crear llistes de reproducció i guardar-les per reproduir-les quan es desitgi, sense haver d'accedir al dispositiu físicament.
- El dispositiu ha de tenir una d'aquestes llistes de reproducció per anar reproduint el contingut amb el mode que se li ha indicat.
- S'han de poder posar prioritats als arxius de les llistes. Aquestes prioritats depenen de la llista. Un ítem pot tenir prioritats diferents en llistes diferents.
- Aquestes prioritats es determinen a l'hora d'afegir un contingut a una llista de reproducció.
- Adaptar l'aplicació de control perquè sigui més atractiva.
- Adaptar l'aplicació de control perquè sigui *responsive* i per tant es pugui visualitzar en altres dispositius de diferents mides.
- Afegir botons al controlador que permetin reiniciar o apagar la Raspberry de forma remota.
- Afegir botons al controlador que permetin reiniciar o apagar els contenidors de forma remota.

Un dels requeriments també era facilitar l'ús per als usuaris. Per això s'han afegit les instruccions relacionades amb la Raspberry a la interfície de control. A més, donat que es tracta d'una aplicació web ens assegura una compatibilitat amb tots els sistemes operatius. També s'ha procurat que la web sigui *responsive* perquè es pugui visualitzar bé a qualsevol dispositiu.

Casos d'ús

Es van dissenyar uns casos d'ús que corresponen a les funcionalitats que es volen afegir a l'aplicació (Rehkopf, s.d.). Per cada cas d'ús es va prioritzar i es va estimar.

Per descriure els casos d'ús seguiré l'estructura següent: "As a [persona], I [want to], [so that]."

- Com a Usuari, vull crear llistes de reproducció per poder reproduir-les quan vulgui.
- Com a Usuari vull determinar prioritats als fitxers a dins de llistes de reproducció per tal que es reprodueixin amb diferents freqüències.
- Com a Usuari vull poder afegir Tags a una llista de reproducció determinada.
- Com a usuari vull apagar la Raspberry per tal d'apagar tota l'aplicació.
- Com a usuari vull reiniciar la Raspberry per tal de reiniciar l'aplicació en cas de fallada del dispositiu.
- Com a usuari vull reiniciar el Docker per tal de reiniciar l'aplicació en cas de fallada de l'aplicació.
- Com a usuari vull apagar el Docker per tal d'apagar l'aplicació en cas que ja no la necessiti.
- Com a usuari vull poder filtrar llistes per etiquetes per tal de seleccionar les llistes més adients.
- Com a usuari vull poder afegir contingut a una llista de reproducció determinada.
- Com a usuari vull poder triar el fitxer intercalat a l'hora de reproduir una llista amb aquest mode o seleccionar el fitxer per defecte.

Disseny

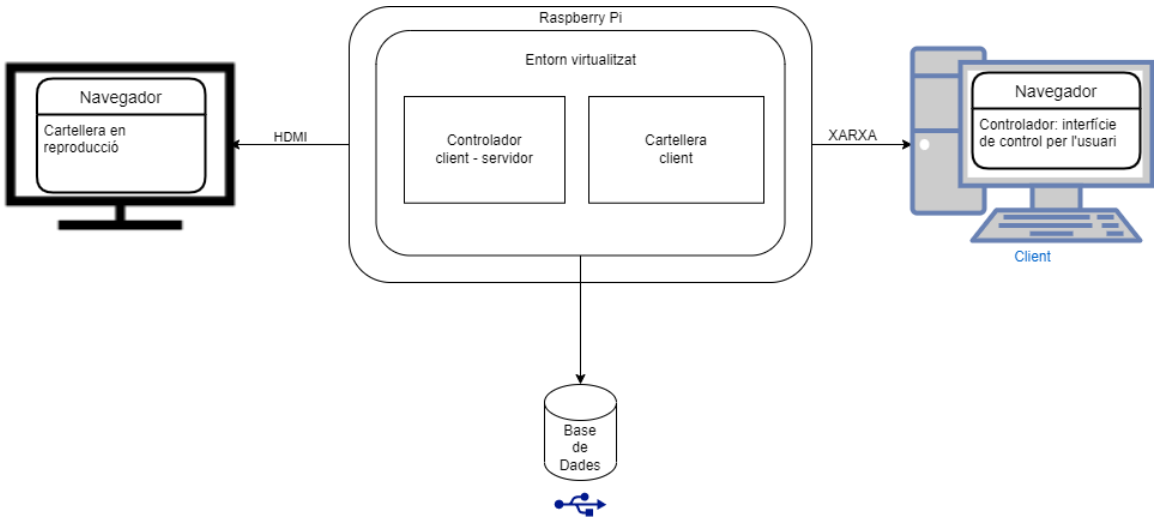
En aquest apartat de Disseny tractarem tant el disseny de l'arquitectura, les eines que s'han utilitzat per desenvolupar ambdues aplicacions i el disseny i les diferències d'aquestes. També veurem el model de la base de dades i els sistemes de seguretat utilitzats.

Arquitectura

L'esquema de l'arquitectura del sistema es podria representar de la següent forma:

Figura 1.

Arquitectura del sistema final.



Nota. Elaboració pròpia.

Com podem veure en el dibuix, el sistema es basa en dues aplicacions que anomenarem Controlador i Cartellera. Cada una d'aquestes aplicacions interactua amb l'altra a través d'unes crides REST (*get, post, delete i put*). També trobem una base de dades comuna que està connectada amb ambdues aplicacions.

El model d'arquitectura és de tipus client-servidor en ambdues aplicacions: el client proporciona una interfície a l'usuari, que permet sol·licitar serveis al servidor i mostra els resultats que el servidor retorna. El servidor espera que arribin les sol·licituds dels clients, les processa i després hi respon. En el nostre cas hi ha dos clients, un per al Controlador i un per a la Cartellera. El Controlador es pot executar a qualsevol ordinador i serveix per configurar les dades que es decideixen visualitzar. L'altre client és el que conté el reproductor i es troba a la Raspberry connectada a la pantalla a la qual volem projectar el contingut.

El servidor del sistema s'executa a la Raspberry a dins d'un Docker. Utilitza una arquitectura API RESTFUL que fa que es distribueixin les tasques a les dues aplicacions que hem comentat, depenent de quina tasca volem executar.

Cada tasca equival més o menys a un microservei, que és el que ens permet connectar el client i el servidor. Cada aplicació té els seus respectius microserveis i es troben interconnectades. Això vol dir que intercanvien informació entre elles però també tenen competències exclusives sobre un grup de dades i funcions. Més

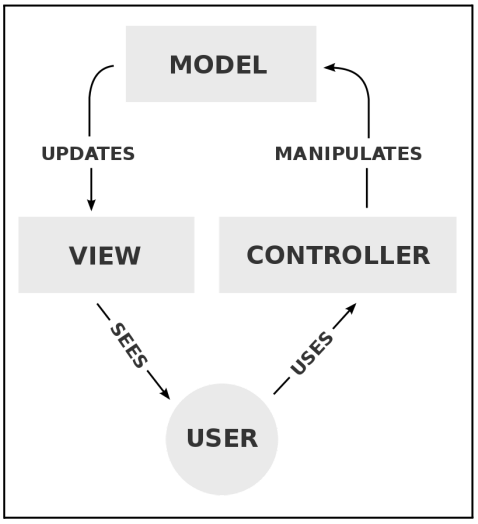
concretament, el Controlador té accés a totes les dades i la Cartellera només té accés a la taula *playlist* de la base de dades, que és qui conté els fitxers a reproduir.
Totes dues s'executen virtualitzades (dins dels seus respectius Dockers) i tenen accés a una base de dades comuna (SQLite) a través d'una ORM anomenada SQLAlchemy que explicarem en detall més endavant.
En poques paraules, l'aplicació del Controlador és la interfície gràfica que permet a l'usuari interactuar amb l'aplicació. La Cartellera en canvi és qui mostra el contingut multimèdia a la pantalla.

Estructura de fitxers

Cal comentar que ambdues aplicacions estan organitzades amb les carpetes *models* i *resources*. Aquesta divisió ens permet organitzar millor la informació i seguir un patró MVC. El patró Model Vista Controlador serveix per aïllar la informació depenent de les dades que vols mostrar a l'usuari. En el nostre cas només volem que l'usuari pugui veure i interactuar amb el contingut, però ocultant la part lògica que és on es prenen les decisions (*backend*).

Basant-nos en aquest diagrama típic per representar el patró MVC (Creative Commons Attribution-ShareAlike, 2022) podem dir que la classe app (de les dues aplicacions), que és la que defineix els serveis, i la carpeta de *resources*, que és qui desenvolupa aquests serveis, són la part del controlador. Trobem també una carpeta en cada aplicació per als models, que com el nom bé indica correspon a la part del Model del MVC. Aquesta carpeta conté els objectes i els mètodes per relacionar-los amb les corresponents taules de la base de dades.

Figura 2.
Esquema típic del patró MVC.



Nota. Recuperat de Creative Commons Attribution-ShareAlike. (2022, Juny 1). *Model–view–controller*. Wikipedia. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Eines

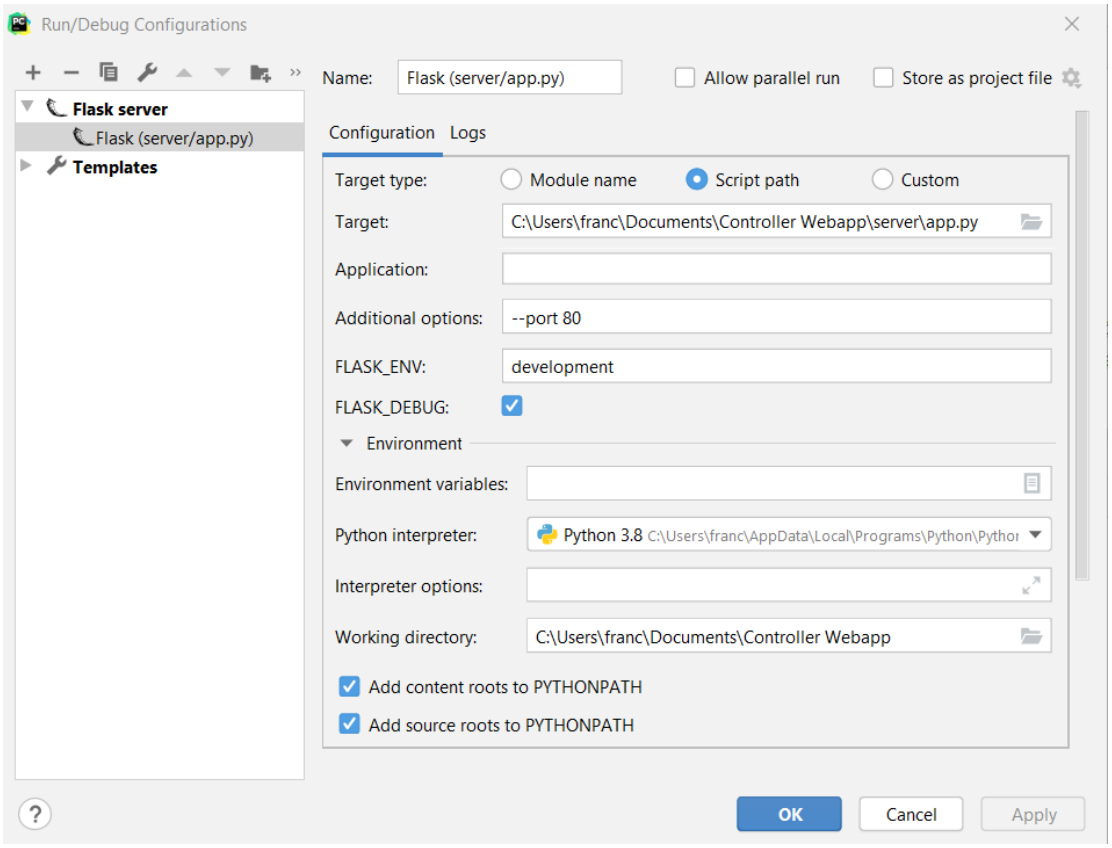
En local

Per treballar amb aquest projecte en local he utilitzat la IDE de programació PyCharm. Aquesta IDE permet introduir configuracions per executar les aplicacions de forma més senzilla. També permet afegir plugins per facilitar l'escriptura de codi i és una eina molt potent a l'hora de depurar codi.

Aquí podem veure la configuració que he emprat per executar l'aplicació del Controlador. La del Billboard és gairebé igual però canviant el port per 8000 i els paths dels directoris corresponents.

Figura 3.

Captura de pantalla per mostrar un exemple de configuració utilitzada al PyCharm per executar el Controlador en entorn local.



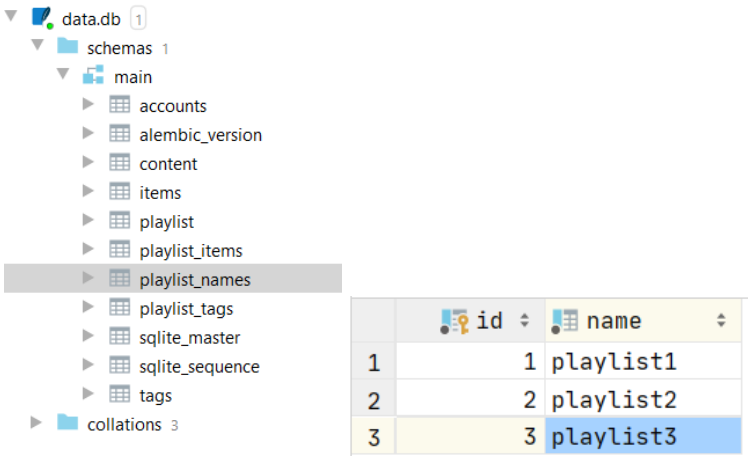
Nota. Elaboració pròpia.

També cal destacar l'eina de Database que ofereix el PyCharm. Amb aquesta eina es pot visualitzar un arxiu de base de dades. En el nostre cas, tenim un arxiu anomenat data.db que conté totes les taules i els elements en elles. Aquesta eina també permet fer modificacions a les taules de la base de dades corresponent, sigui a través de comandes SQL o a través de la interfície gràfica que proporciona.

Així és com es visualitzen les bases de dades i les taules corresponents amb aquesta eina:

Figura 4.

Captura de pantalla per mostrar com es veuen les taules de la base de dades amb l'eina de PyCharm.



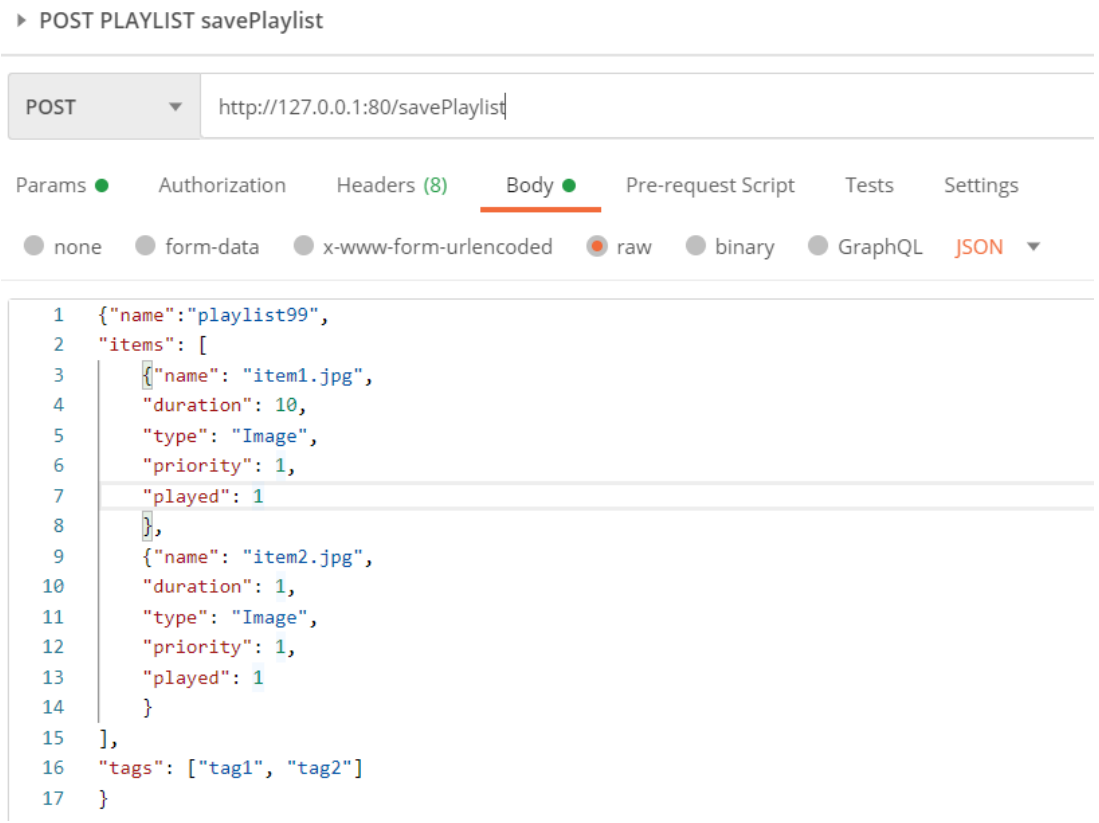
Nota. Elaboració pròpia.

Per altra banda, he utilitzat una eina anomenada Postman que serveix per fer crides a APIs i testejar-les. Aquesta eina serveix per anar testejant els microserveis amb els seus respectius paràmetres d'entrada i de sortida.

Aquest és un exemple de les proves que he anat fent amb aquesta eina. En concret per provar el microservei /*savePlaylist* del Controlador:

Figura 5.

Captura de pantalla per executar un microservei amb el programari Postman.



Nota. Elaboració pròpia.

Per anar guardant els canvis i tenir-ne un seguiment, he penjat el codi a un repositori de Github i he utilitzat l'aplicació de Github Desktop per gestionar les versions i els canvis. D'aquesta manera tinc un historial de canvis i també una còpia del codi penjada al núvol per poder-hi accedir-hi en tot moment.

Cal comentar breument que he usat Chrome com el navegador per defecte per obrir les aplicacions en *localhost* i algunes vegades per fer *debug* d'alguns mètodes pertanyents al *frontend* del controlador. També mencionar el senzill, però útil Notepadd++ que he utilitzat per editar alguns dels Dockerfiles.

Finalment, he fet servir l'eina de Docker i Docker Compose per crear i executar els contenidors on està emmagatzemat el codi i els quals farem servir per tenir una imatge de les aplicacions i que es puguin desplegar en qualsevol entorn. Explicarem en detall a l'apartat de Dockers a dins de l'apartat d'Implementació.

En producció

El sistema de senyalització ja està muntat perquè com hem dit es tracta d'una aplicació en funcionament. Està format a nivell de hardware per:

- Una Smart TV LG, model 43UM7100PLB
- Una placa Raspberry Pi 4 amb 8GB de RAM
- Una targeta SD per al sistema operatiu
- Una memòria USB (on s'emmagatzema el contingut multimèdia i la base de dades)
- Un temporitzador digital programable

A nivell de software:

- Dos containers Docker comunicats mitjançant docker compose
- Dues aplicacions Flask desenvolupades en llenguatge Python conjuntament amb el Framework de Vue.js (per al controlador) i HTML + Javascript (per la cartellera)
- Una base de dades relacional de tipus SQLite que conté diverses taules
- Scripts en Bash

Aplicació 1: El reproductor

Funcionament

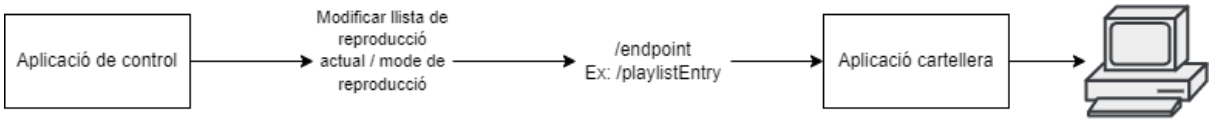
El reproductor és l'aplicació client que s'ocupa de reproduir el contingut en un navegador. El servidor s'encarrega d'ordenar i gestionar el contingut a mostrar mentre que el client només el mostra.

Aquesta aplicació no té cap mena de control sobre cap arxiu ni té accés a cap servei del controlador. El que fa és consultar la llista de reproducció determinada per l'usuari, el mode en què s'ha de reproduir i l'estat en què es troba la llista (amb alguns matisos que explicaré més endavant).

El funcionament d'aquest reproductor es podria resumir amb aquest diagrama:

Figura 6.

Diagrama de funcionament del reproductor.



Nota. Elaboració pròpia.

Frontend

Per aquesta aplicació no ens ha fet falta utilitzar cap Framework de *frontend* ja que es tracta només d'una pantalla. Aquesta pantalla està feta amb HTML i CSS i és l'encarregada de rebre un arxiu i reproduir-lo els segons indicats.

Backend

Tant aquesta aplicació de cartellera com l'aplicació de control estan dividides en serveis. Aquests serveis estan dividits en classes per tenir una millor organització, com hem mencionat abans. Podem trobar els serveis corresponents a aquesta aplicació a la classe app, que és el servidor en si. Aquesta classe delega les crides mitjançant les APIs que hi definim. També és l'encarregada de retornar la vista corresponent al reproductor.

Les dues classes *'Resources'* principals en aquesta aplicació passen a ser només tres: Playlist, setPlaylist i Scripts. Un cop l'aplicació (app.py) agafa l'*endpoint* corresponent, deriva la gestió de la seva crida en les classes mencionades. Aquestes classes no modifiquen directament les dades sinó que deriven la consulta o modificació a la classe del model corresponent.

Com que ara en comptes d'anar reproduint arxius reproduïm llistes senceres, trobem el Resource setPlaylist que és l'encarregat de rebre la llista de reproducció, el mode en què es vol reproduir i el fitxer intercalat que l'usuari ha escollit (si escau).

Per altra banda trobem Playlist que és la que s'encarrega de la lògica de la reproducció depenent del mode fixat per l'usuari. Aquesta lògica es du a terme concretament dins del resource anomenat NextEntry que és el que va determinant el següent fitxer a reproduir.

Finalment tenim la classe Scripts que serveix perquè l'usuari pugui manipular tant els dockers com la Raspberry des del controlador sense necessitat d'accedir-hi

físicament. Trobem també unes quantes classes auxiliars que serveixen per consultar informació i que fins ara eren les que donaven sentit a l'aplicació però algunes d'elles ja no s'utilitzen.

Mètodes RESTful

Veiem la llista d'*endpoints* de l'aplicació de la Cartellera:

Taula 1.

Col·lecció d'endpoints de l'aplicació de la Cartellera.

Mètode	Endpoint	Descripció
GET	/mode	Permet saber el mode de reproducció seleccionat en aquell moment
PUT	/mode	Permet canviar el mode de reproducció seleccionat
POST	/playlistEntry (obsolet)	Permet afegir una nova entrada a la llista de reproducció
DELETE	/playlistEntry/<string:name>	Permet eliminar l'entrada corresponent al nom de la llista de reproducció
GET	/playlist	Permet veure tota la llista de reproducció i els seus ítems
GET	/nextEntry	Permet saber les metadades del següent fitxer a reproduir
POST	/nextEntry	Permet determinar el següent fitxer a reproduir
GET	/status	Permet saber l'estat del reproductor
POST	/status	Permet canviar l'estat del reproductor
GET	/intercalatedEntry	Permet saber quina és l'entrada intercalada
POST	/intercalatedEntry	Permet afegir una entrada per utilitzar-la de fitxer intercalat

DELETE	/intercalatedEntry	Permet eliminar l'entrada intercalada
POST	/setPlaylist	Permet guardar la llista de reproducció a reproduir i el mode corresponent
POST	/scipts	Permet executar un script referent al docker o la Raspberry

Nota. Elaboració pròpia.

Aplicació 2: El controlador

Funcionament

L'aplicació del controlador té la mateixa estructura que l'altra, com hem comentat, però aquesta serveix per a unes funcions totalment diferents. El seu client és la interfície de control que permet a l'usuari configurar el contingut a reproduir. Es tracta d'una interfície que l'usuari pot veure al navegador i pot interactuar per tal de definir les tasques que necessiti dur a terme. És també qui s'encarrega del control d'usuaris i dels fitxers multimèdia. Per fer-ho també compta amb uns quants microserveis que connecten amb un servidor i la base de dades.

Frontend

Aquesta aplicació està formada com hem dit per una interfície gràfica per tal que l'usuari pugui manipular la informació desitjada. Aquesta interfície està creada amb un framework de JavaScript anomenat Vue.js. Aquest framework permet dividir els elements a mostrar en components. El que fem és crear un component "pare" anomenat App.vue que és el que conté tots els altres. Llavors l'únic que fem és anar mostrant o ocultant els components a mesura que l'usuari vagi interaccionant amb la interfície. Aquest component, però, també es divideix alhora en dos components que són els components més globals. Un que és el del login i un que és el panell de control. Aquests dos components també estan formats per components més petits. Com explicarem més en detall en l'apartat de *Seguretat*, cal tenir en compte que sense iniciar sessió no podrem veure cap més component que el del login.

A dins del panell de control (Playlist.vue) trobem gairebé tots els altres components que es van mostrant depèn de la pestanya seleccionada. La primera pestanya

correspon al component Content.vue. Aquest component serveix per afegir contingut a l'aplicació per després ficar-lo a alguna llista de reproducció. En aquest component utilitzem també el ContentUpload.vue que és el que ens permet arrossegar arxius o seleccionar-los de l'ordinador per pujar-los.

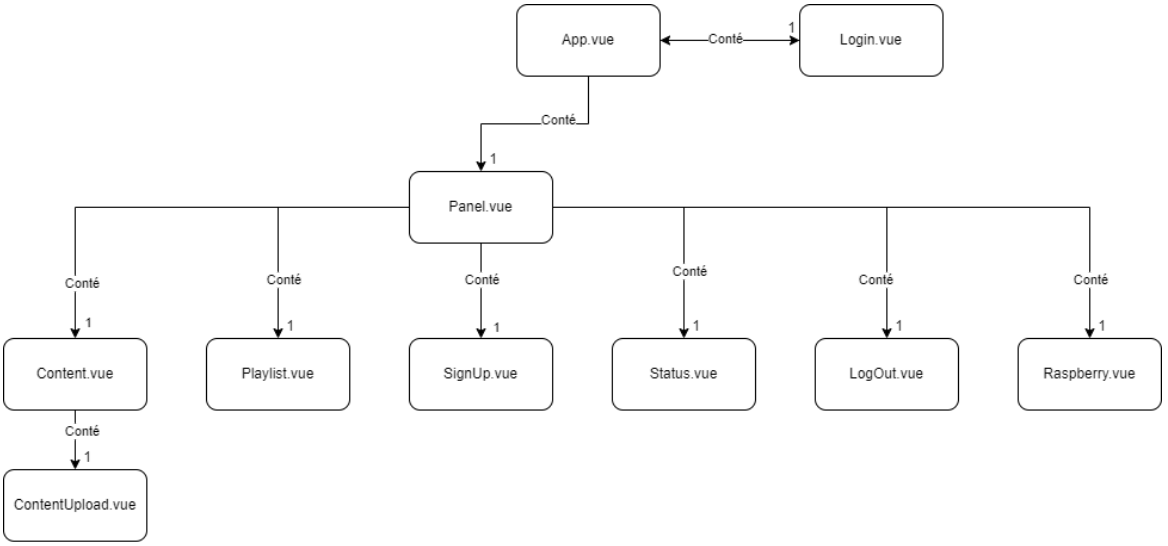
A la següent pestanya trobem el component principal que és el Playlist.vue. Aquest component és el que ens permet seleccionar una llista de reproducció de les ja creades i reproduir-la. També permet seleccionar el mode de reproducció i el fitxer intercalat.

La part de la vista de la gestió d'usuaris està al component SignUp.vue. En quant a la vista per poder canviar l'estat del reproductor i entrar o sortir del mode manteniment, es troba al component SwitchStatus.vue. Trobem també el component Raspberry.vue que conté tots els botons per interactuar amb la Raspberry i amb els Dockers. I finalment per tancar la sessió, la vista està implementada al component LogOut.vue.

Aquí tenim un esquema d'aquests components:

Figura 7.

Esquema dels components de l'aplicació del Controlador.



Nota. Elaboració pròpia.

Backend

Aquest apartat s'assembla molt al de l'Aplicació 1, ja que, ambdues aplicacions tenen una organització i estructura bastant semblants. Cal comentar que algunes de les classes de les aplicacions hi són presents a les dues.

Com passava amb l'altra aplicació, trobem una estructura MVC formada per la classe App, que és la principal, els *Resources* i els *Models*. La classe app és la que defineix els endpoints i delega les crides als *Resources* corresponents. Podríem dir que aquesta classe és el servidor en si. Llavors en cada Resource hi ha uns mètodes HTTP que fan les operacions i retornen la informació necessària per mostrar-la. Els models són els encarregats de fer la gestió de dades (modificar, guardar, etc.) en la taula de la base de dades corresponent.

De tots aquests endpoints disponibles a l'aplicació, cal que en detallem un d'ells perquè és la que s'utilitza més i la més difícil de definir:

- Playlists: És l'encarregada de guardar les llistes de reproducció: el nom, els tags relacionats, els ítems. Això s'ha fet amb l'eina de SQLAlchemy de Flask per tractar les relacions Many-to-Many (Sigils, 2014). Per això té una relació entre models utilitzant de la forma:

```
tags = relació("TagsModel", secondary=playlist_tags)

items= relació("ItemsModel", secondary=playlist_items)
```

Mètodes RESTful

Aquí la llista d'endpoints de l'aplicació del Controlador:

Taula 2.

Col·lecció d'endpoints de l'aplicació del Reproductor.

Mètode	Endpoint	Descripció
GET	/account/<string:username>	Permet saber la informació d'un compte d'usuari donat un nom
POST	/account	Permet afegir un nou compte d'usuari
DELETE	/account/<string:username>	Permet eliminar un compte d'usuari ja existent
PUT	/account/<string:username>	Permet modificar la informació d'un compte d'usuari ja existent
GET	/accounts	Permet veure una llista amb tots els coptes d'usuari

POST	/login	Permet iniciar sessió
POST	/content	Permet afegir un nou fitxer
PUT	/content<string:filename>	Permet modificar la informació d'un fitxer existent (no la informació en el disc)
DELETE	/content<string:filename>	Permet eliminar les metadades i un fitxer (també del disc)
GET	/content	Permet veure una llista amb tots els continguts
GET	/playlistlist	Permet veure una llista dels noms de llistes de reproducció
GET	/playlists/<string:name>	Permet veure la informació d'una llista de reproducció existent
POST	/savePlaylist	Permet afegir una nova llista de reproducció amb els seus items i tags corresponents
GET	/items	Permet veure una llista dels items existents
POST	/item	Permet afegir un item a una llista de reproducció existent
GET	/tags	Permet veure una llista dels tags existents
POST	/tag	Permet afegir un tag a una llista de reproducció existent

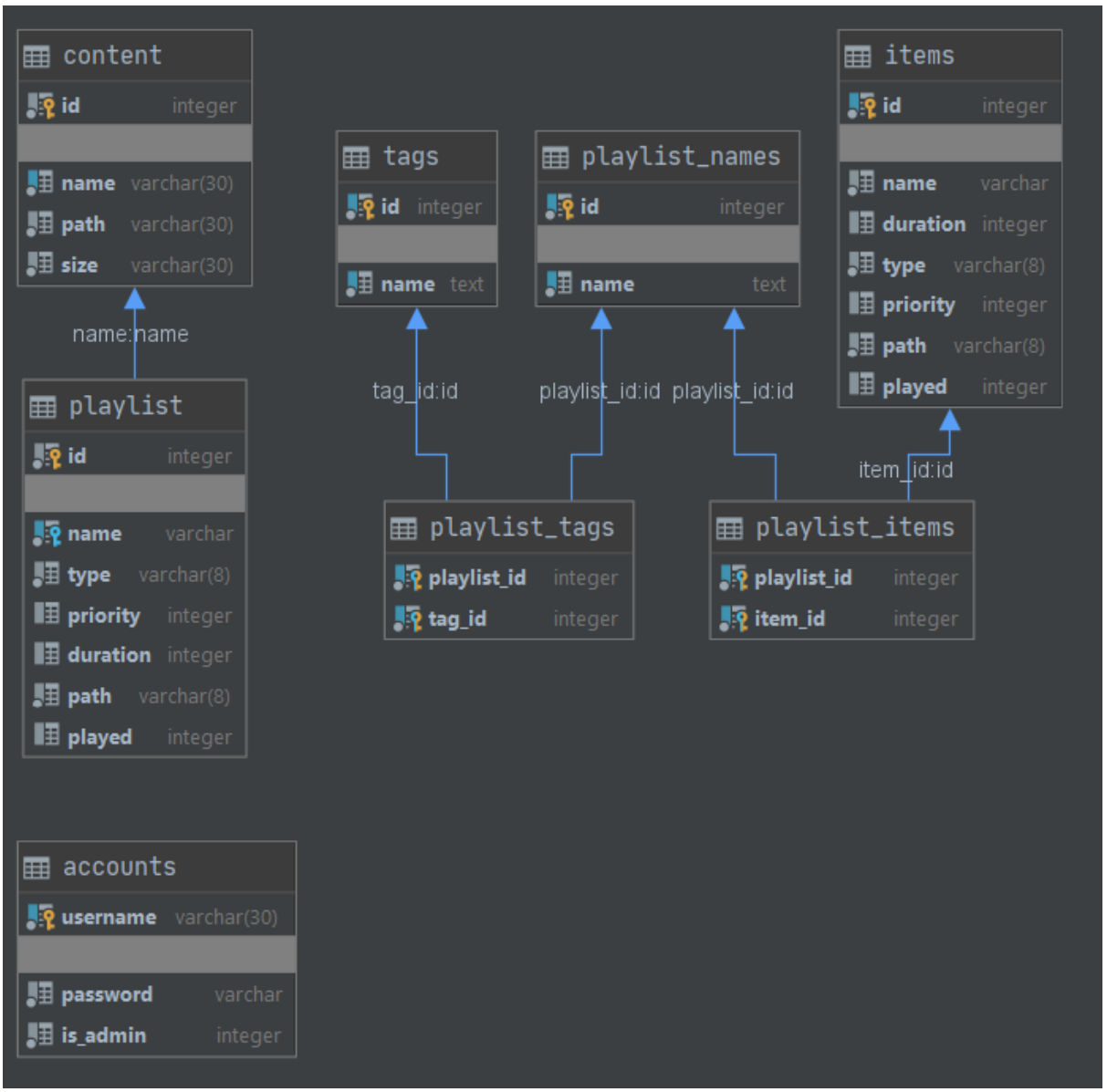
Nota. Elaboració pròpia.

Model relacional

Aquí veiem el model de dades, que és el mateix per les dues aplicacions amb l'única diferència que des del controlador accedim a totes les taules i des de la cartellera només accedim a la taula *playlist* que és on tenim guardada la llista de reproducció que s'està reproduint en aquell moment.

Figura 8.

Model entitat relació.



Nota. Elaboració pròpia a través del Plugin UML de PyCharm.

Totes les taules de la base de dades i les columnes corresponen a un objecte del Model i els seus atributs corresponents, que s'anomenen igual que a les taules.

Les relacions que trobem són entre la taula *playlist_names* i items i també amb la taula tags. Això ho vam fer perquè cada llista de reproducció pot tenir uns quants tags associats i conté una llista d'ítems.

Aquestes relacions es guarden a *playlist_items* i *playlist_tags*. Llavors, quan fem una consulta a una taula podem obtenir també els elements relacionats amb aquesta.

A part d'aquesta base de dades i dels models, hem fet ús de la llibreria Pickle de Python per guardar en local algunes de les variables necessàries a l'hora de reproduir. Per exemple, hem guardat la variable que ens indica si és el torn del fitxer intercalat o si és torn del següent fitxer seqüencial. Aquesta llibreria el que fa és guardar una variable en forma de bits en un fitxer i ens proporciona una manera molt senzilla de guardar i accedir a n'aquestes dades (*Pickle — Python Object Serialization*, s.d.).

Seguretat

Per l'aplicació de controlador hi ha una pantalla per fer iniciar sessió. Aquest inici de sessió està relacionat amb la taula *accounts*. En el moment del registre d'un usuari, aquest introdueix el nom i la contrasenya, que s'encripta i es guarda a la base de dades. Després en el moment del login el que fem és comparar les credencials amb les dades de la taula *accounts*. Aquesta encriptació es fa amb el protocol OAuth (*OAuth 2.0*, s.d.).
Això ho fem per tenir un protocol de seguretat a l'hora de mostrar informació a l'usuari. Per això a gairebé tots els mètodes del controlador hem de demostrar que hi tenim accés. Això es fa passant un header amb el token generat a l'inici de la sessió per demostrar que som un usuari vàlid i garantir la seguretat de l'aplicació i de les dades: `auth: { username: token }`
Indiquem que els mètodes són restringits amb la següent etiqueta que col·loquem damunt de cada Resource corresponent: `@auth.login_required()`

Implementació

Estructura de la base de dades

La base de dades per ambdues aplicacions és la mateixa, la diferència és que la cartellera només interacciona amb la taula "playlist" mentre que el controlador gestiona totes les altres taules.
La base de dades utilitzada és del tipus SQLite perquè es tracta de dues "embed applications" i SQLite és una gran opció de base de dades per a aquestes aplicacions que necessiten portabilitat i no requereixen una gran expansió futura.

Com hem dit prèviament, les dades estan organitzades en models i cada un d'ells correspon a una taula de la base de dades amb les seves respectives columnes. Cada model representa un objecte i utilitzem SQLAlchemy de Python per fer-ho.
SQLAlchemy és un conjunt d'eines d'accés a la base de dades per a Python, amb el seu *object-relational mapper* (ORM). Aquest ORM proporcionat per SQLAlchemy és el que ens permet pensar en les dades com objectes (Farrell, s.d.). A més utilitzem aquesta eina per crear totes les queries de l'aplicació, en comptes de fer-les en format SQL. Un exemple n'és aquesta query que utilitzem per filtrar un item de la playlist per el seu id en l'aplicació del reproductor:

```
PlaylistModel.query.filter_by(id=id).first()
```

Aquesta query amb sintaxi SQL seria equivalent a:

```
SELECT * FROM playlist WHERE id = id LIMIT 1
```

Modes de reproducció

En aquesta aplicació disposem de quatre modes de reproducció diferents. Això vol dir que donada una llista es pot reproduir de quatre maneres diferents: seqüencial, intercalat, aleatori i aleatori-intercalat.
L'usuari pot triar el mode de reproducció a l'aplicació del controlador a l'hora de decidir quina llista de reproducció es vol reproduir.
Abans d'explicar en detall els diferents modes cal que fem una aclaració. Existeix un fitxer anomenat *default.mp4* que el veurem aparèixer en aquests algoritmes. Aquest fitxer és un fitxer per defecte que es troba físicament a l'aplicació i serveix com a comodí en cas que alguna cosa falli. Per exemple, si hem seleccionat el mode intercalat i no hem seleccionat cap fitxer, apareixerà aquest fitxer per defecte. També sortirà en el cas que afegim una llista buida.
Dit això, expliquem una mica els diferents modes en detall i els corresponents algoritmes.

Seqüencial

Com el seu nom indica, es van reproduint els fitxers un després de l'altre, sense importar la prioritat. Quan un fitxer es reprodueix, es canvia la columna de "played" de la taula playlist per indicar que ja s'ha reproduït i poder passar al següent. Quan

tots estan “played”, es reprodueix el primer de tots i es modifiquen tots com NO “played”.

Algoritme en pseudocodi:

```
Si Item['played'] = 0:
     Item['played'] = 1
     retorna Item
sinó:
     PlaylistModel.makeAllUnplayed()
     First_Item['played'] = 1
     retorna First_Item
```

Intercalat

El mode intercalat reprodueix de forma seqüencial i a més a més va intercalant un fitxer (el fitxer intercalat) entre cada fitxer. L’usuari a l’hora de decidir quina llista de reproducció a reproduir i el mode, el sistema demana a l’usuari quin dels fitxers de la llista vol que sigui el fitxer a intercalar. L’usuari pot seleccionar un dels fitxers de la llsita o pot decidir que sigui el fitxer per defecte, com hem comentat. Per triar el següent fitxer a reproduir es segueix el mateix algoritme que al mode seqüencial però tenint en compte una variable que indica si toca reproduir el fitxer intercalat o el fitxer que toca segons la seqüencialitat. Aquesta variable es guarda en un fitxer en forma de bits i es consulta i es modifica per anar actualitzant el valor i donar sentit a n’aquest algoritme.

Algoritme en pseudocodi:

```
Si TornFitxer:
     TornFitxer = Fals
     Item = Find_Next_Item()
     retorna Item
sinó:
     TornFitxer = Vertader
     Item = FitxerIntercalat
     retorna Item
```

Aleatori

Aquest mode el que fa és reproduir els elements de la llista de forma pseudo-aleatoria. Això és perquè a part de ser aleatori, té en compte les prioritats dels arxius. L'usuari a l'hora d'introduir un contingut a una llista de reproducció, pot seleccionar la durada però també la seva prioritat (de l'1 al 5). Aquesta prioritat és la que s'utilitza per determinar el nivell de aleatorietat a l'hora de reproduir.

Algoritme en pseudocodi:

```
    Item = Find_Random()
    retorna Item
```

Aleatori - Intercalat

L'últim mode que trobem és l'aleatori - intercalat, que és una mescla dels altres dos modes. Té un component aleatori però també va intercalant el fitxer seleccionat per l'usuari.

Algoritme en pseudocodi:

```
Si TornFitxer:
    TornFitxer = Fals
    Item = Find_Random()
    retorna Item
sinó:
    TornFitxer = Vertader
    Item = FitxerIntercalat
    retorna Item
```

Dockers

Com que aquestes aplicacions les ha de poder utilitzar gent que no té perquè tenir idea d'informàtica, la intenció és posar-lis lo més fàcil possible. Per això en el projecte original ja es van virtualitzar les aplicacions i es van distribuir una sèrie d'instruccions perquè qualsevol persona pugui executar-les. Per fer aquesta virtualització hem utilitzat l'eina anomenada Docker. Aquesta eina el que fa és encapsular el codi dins d'una imatge i permetre que l'usuari l'executi sense idea del que hi ha a dins. Com ja es feia prèviament, trobem dos Dockerfiles que són fitxers

que serveixen per crear el contenidor de cada aplicació. Llavors, el controlador té el seu Dockerfile corresponent i la cartellera també. Després, utilitzem l'eina anomenada Docker Compose i un fitxer docker-compose.yml. Aquesta eina s'utilitza per executar diversos contenidors com un sol servei. Llavors, el que fem és indicar-li bàsicament on pot trobar els Dockerfiles de les aplicacions que volem i que es faci tot en un sol procés.

Manual d'instal·lació

Per executar les aplicacions ho podem fer de dues maneres: en local o virtualitzades.

Local

Si el que volem és canviar alguna cosa de codi llavors ho hem de fer a través d'algun IDE de programació. Per això primer de tot haurem d'instal·lar els requeriments necessaris. Es pot fer a través del fitxer requirements.txt i amb la instrucció: `pip install -r requirements.txt`. També cal tenir instal·lat el Node.js perquè utilitzarem npm per executar el *frontend*.
Quan ja tenim les eines instal·lades i preparades, seguim els passos següents:

1. Hi ha dues opcions: obrir les dues carpetes en la mateixa pestanya de l'IDE i tenir dues configuracions de Flask per engegar-les o bé obrir-les en dues pestanyes diferents. Jo ho he fet amb dues pestanyes diferents, per tant, els passos que explicaré es basen en aquesta metodologia.
2. En la carpeta Controller webapp desplaçar-se a la carpeta *client* i executar el servidor Flask. Es pot fer des de l'IDE amb una configuració com hem explicat prèviament o per línia de comandes amb la següent comanda: `flask run --port 80`
3. Llavors si volem fer crides al sistema ja podem fer-ho. Ja sigui amb el programa Postman com hem comentat, amb la terminal o amb qualsevol altra eina.
4. Per executar el *frontend* i veure la interfície el que hem de fer és desplaçar-nos a la carpeta server i escriure les comandes:
   Si és el primer cop cal que executem: npm install; npm build

Després només cal fer la instrucció npm run serve

5. Amb aquests passos ja tindrem el controlador preparat per la seva utilització.

Per executar el Billboard webapp és bastant més senzill. Notem que també s'han de tenir instal·lats els requeriments inicials. Després només cal engegar un servidor Flask amb una configuració de l'IDE o amb una línia de comandes com aquesta: `flask run -p 8000`

Com veiem els servidors de Flask estan a dos ports diferents (80 i 8000) i els hem determinat nosaltres. Això és molt important perquè a l'hora de comunicar-se ho fan per aquests ports en concret. Per altra banda el *frontend* s'executa al port per defecte que en aquest cas ens és indiferent.

Entorn virtualitzat

Com hem comentat abans a l'apartat d'Implementació, suposem que tenim ja els Dockers creats i totes les eines necessàries instal·lades. Llavors només hem d'executar el procés per "aixecar" les aplicacions. El que fa aquest procés és crear el contenedor de la cartellera i executar-lo en el port 8000 i el mateix amb la cartellera però en el port 80. D'aquesta manera ja tenim les aplicacions llestes per utilitzar.

Per fer aquest procés només cal obrir un terminal i accedim a la carpeta on tenim el codi font. Un cop allà, introduïm: `docker-compose up`

Finalment, per accedir a les aplicacions, hem d'obrir un navegador d'internet i introduir les següents URL en diferents finestres o pestanyes :

- http://127.0.0.1:8000 (per veure la cartellera)
- http://127.0.0.1:80/controlPanel (per veure la interfície de control)

Noteu que no és necessari tenir obertes les dues apps alhora per comprovar el funcionament només d'una.

Proves i resultats

Per fer aquestes proves, en alguns casos s'ha fet en local i d'altres en l'entorn virtualitzat. Com que els entorns són exactament iguals, en principi els resultats han de ser els mateixos, amb una excepció, que és a l'hora d'introduir contingut: aquest només es pot fer quan l'entorn està virtualitzat ja que està fet de tal manera que

l'aplicació verifica l'existència de l'arxiu a la memòria USB connectada a la Raspberry. Això es fa per assegurar la consistència de dades.

Algunes de les proves que s'han fet han estat les següents:

Taula 3.

Proves realitzades a les dues aplicacions.

Prova	OK / KO	Solució
Iniciar sessió amb credencials incorrectes	OK	
Iniciar sessió amb credencials correctes	OK	
Afegir nou contingut a l'aplicació en local	KO	Això és degut a que el contingut només es pot afegir quan estem connectats a la Raspberry i hi tenim un USB connectat. No disponible en local
Afegir nou contingut a l'aplicació virtualitzada	OK	
Afegir contingut a una llista de reproducció ja creada	OK	
Afegir contingut a una llista de reproducció nova (crear llista de reproducció)	OK	
Reproduir llista de reproducció amb mode seqüencial	OK	
Reproduir llista de reproducció amb mode intercalat seleccionant un arxiu intercalat	OK	
Reproduir llista de reproducció amb mode intercalat seleccionant l'arxiu per defecte	OK	
Reproduir llista de reproducció amb mode aleatori	OK	
Reproduir llista de reproducció amb mode aleatori - intercalat	OK	
Eliminar llista de reproducció	OK	

Eliminar un arxiu de contingut	KO	No disponible perquè si aquest contingut és a cap llista de reproducció, aquesta quedaria corrompuda
Reiniciar la Raspberry	OK	
Reiniciar Docker	OK	
Apagar Raspberry	OK	
Apagar Docker	OK	
Filtrar llistes de reproducció per tags	KO	No implementat
Provar la mateixa llista amb mode intercalat i diferents prioritats	OK	

Nota. Elaboració pròpia.

Tenint en consideració que algunes de les pantalles ja existien, l'aspecte de l'aplicació final és aquest:

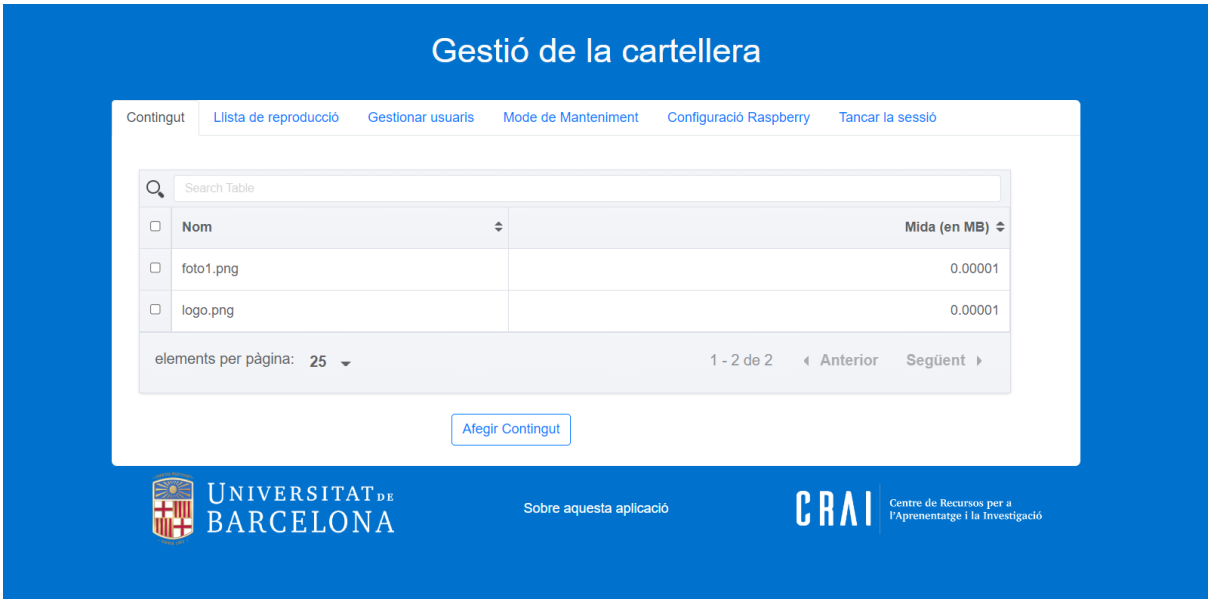
Figura 9.

Captura de pantalla de la interfície d'usuari final. Finestra d'inici de sessió.

Nota. Elaboració pròpia.

Figura 10.

Captura de pantalla de la interfície d'usuari final. Pestanya "contingut".



Nota. Elaboració pròpia.

Figura 11.

Captura de pantalla de la interfície d'usuari final. Secció "afegir contingut" pertanyent a la pestanya "contingut".

Nota. Elaboració pròpia.

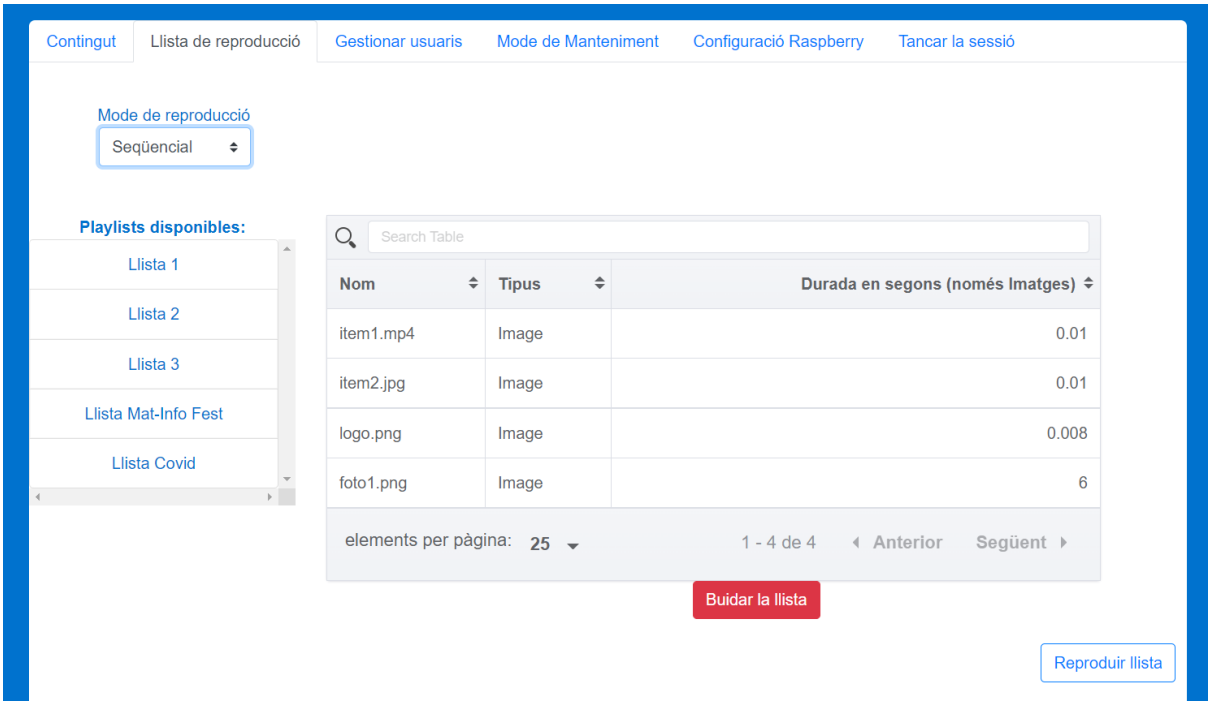
Figura 12.

Captura de pantalla de la interfície d'usuari final. Secció "Afegir a llista de reproducció" pertanyent a la pestanya "contingut".

Nota. Elaboració pròpia.

Figura 13.

Captura de pantalla de la interfície d'usuari final. Secció "Reproduir llista seleccionada" pertanyent a la pestanya "llista de reproducció".



Nota. Elaboració pròpia.

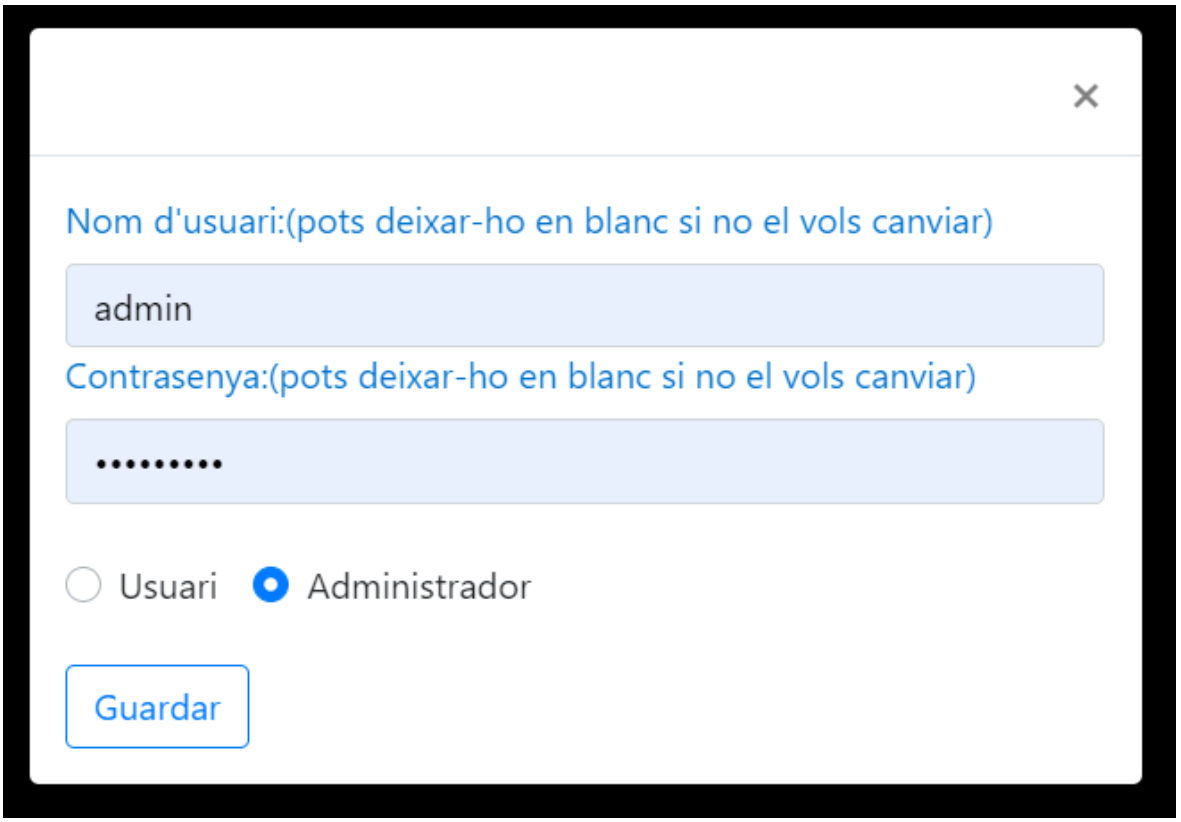
Figura 14.

Captura de pantalla de la interfície d'usuari final. Pestanya "Gestionar usuaris".

Nota. Elaboració pròpia.

Figura 15.

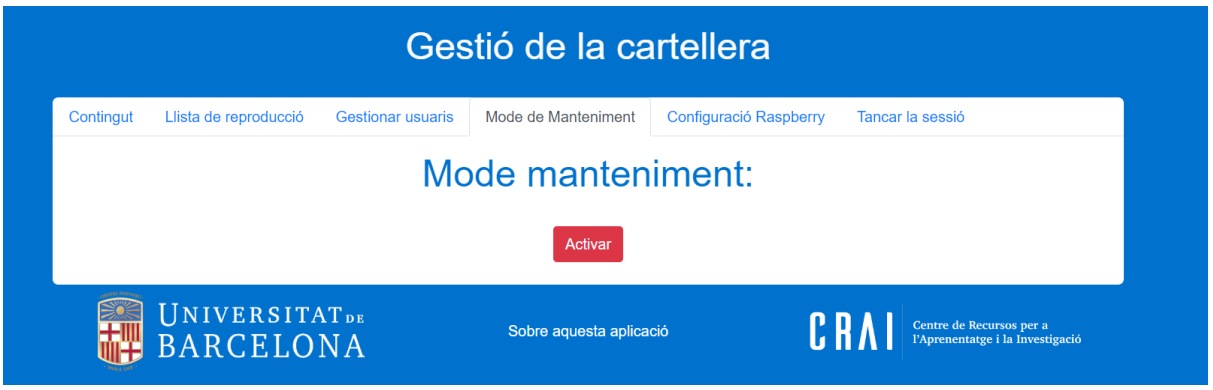
Captura de pantalla de la interfície d'usuari final. Pestanya "Gestionar usuaris". Secció "Modificar usuari seleccionat" pertanyent a la pestanya "Gestionar usuaris".



Nota. Elaboració pròpia.

Figura 16.

Captura de pantalla de la interfície d'usuari final. Pestanya "Mode manteniment".



Nota. Elaboració pròpia.

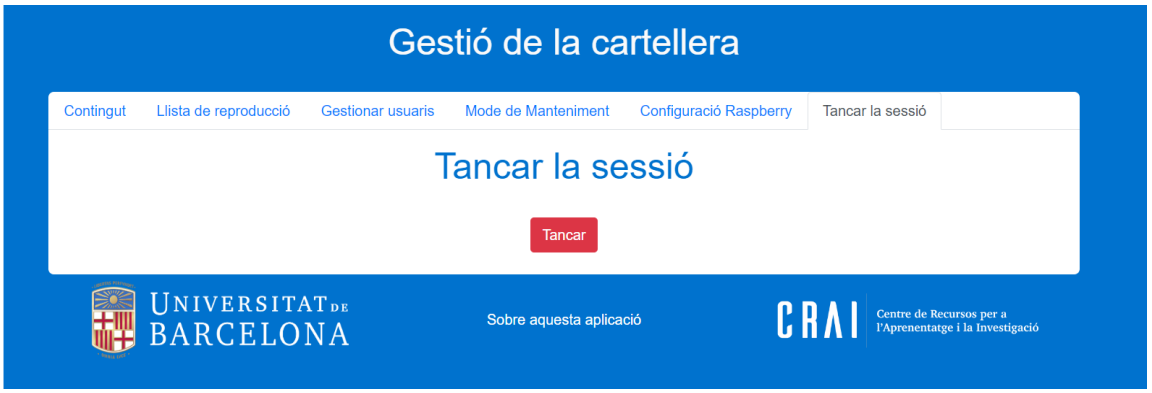
Figura 17.

Captura de pantalla de la interfície d'usuari final. Pestanya "Configuració Raspberry".

Nota. Elaboració pròpia.

Figura 18.

Captura de pantalla de la interfície d'usuari final. Pestanya "Tancar la sessió".



Nota. Elaboració pròpia.

Conclusions i treball futur

Per concloure aquest treball, cal comentar que el projecte és molt escalable i es pot arribar a aconseguir una aplicació que serveixi per més biblioteques i fins i tot per altres entorns.

De totes les idees que teníem al principi jo i els meus tutors, al final la limitació de temps i algunes dificultats afegides pel camí -sobretot a l'hora de desenvolupar l'aplicació en local, que m'ha obligat a canviar considerablement la configuració inicial- han fet que només n'hagi pogut dur a terme unes quantes.

Sigui com sigui, crec que amb la meva aportació la interfície d'usuari ha quedat molt més clara i que la funcionalitat de les llistes de reproducció era molt necessària. I, en aquest sentit, estic contenta del resultat. També ,'agrada veure que he reforçat molts coneixements que havia adquirit a la universitat i que he après moltes altres coses noves força interessants, com ara treballar amb un entorn virtualitzat i poder-hi accedir només des de la xarxa de la biblioteca.

Finalment voldria deixar constància d'algunes de les idees que se'ns havien acudit i da'ltres que han anat apareixent al llarg del desenvolupament, perquè qualsevol persona pugui seguir amb aquest projecte en el futur:

- Implementar un sistema d'agenda que permeti programar la reproducció de contingut en determinats moments.
- Desenvolupar una tercera aplicació web per controlar l'apagada i reinici de la placa i del Docker.
- Veure la durada dels vídeos a la taula de la llista de reproducció.
- Afegir controls en directe per gestionar la reproducció: pausa, següent, anterior, etc.
- Afegir suport per altres tipus d'arxius com per exemple Power Point, PDF, etc). Fins ara els tipus de fitxers suportats eren Imatges i Vídeos.
- Canviar el mode d'emmagatzematge de fitxers. En comptes d'utilitzar un USB, tenir els fitxers al núvol i permetre reproduir-los des d'allà.
- Permetre afegir múltiples fitxers a la vegada a una llista de reproducció ja que ara es fa un per un.
- Permetre crear una còpia d'una llista de reproducció.

Finalment, de la llista de possibles funcionalitats i canvis en l'aplicació explicades a l'apartat d'Introducció, al final s'han acabat desenvolupant les següents:

- Podem crear i guardar llistes amb un nom. Tenim una llista amb les llistes disponibles on podem seleccionar la que ens interessa. Quan la seleccionem es recupera tant els arxius com la configuració (durada, prioritats, etc).
- Podem eliminar la llista desitjada i tot el seu contingut.
- Podem etiquetar aquestes noves llistes amb etiquetes (tags) per tal de poder classificar-les millor.
- Podem afegir filtre per etiquetes a l'hora de mostrar les llistes disponibles
- Podem determinar la prioritat de cada arxiu quan l'afegim a una llista. Ho fem amb un sistema d'estrelles de 1-5. Per tant la prioritat de l'arxiu depèn plenament de la llista, no de l'arxiu en si.
- S'ha millorat classificació i cerca de fitxers a l'hora d'afegir contingut.
- S'han eliminat els id's de les llistes de l'aplicació de control ja que no donen informació a l'usuari.
- S'ha permès eliminar un element d'una llista de reproducció.
- S'han creat eines per facilitar el funcionament a l'usuari. S'han creat crides a algunes comandes perquè es puguin reiniciar o apagar tant el Docker com la Raspberry.
- Disposem d'una eina per modificar els fitxers: canvi de nom des de la pròpia aplicació. Quan es selecciona un arxiu, mostrar opcions: canviar nom, canviar etiqueta. No cal modificar el fitxer en local, modificar només el nom del fitxer i el nom per mostrar. O sigui, canviar els noms però mantenir el mapejat amb l'arxiu original amb el nom original.
- Podem veure la durada dels vídeos a la taula de la llista de reproducció; ara només la mostra pels fitxers estàtics. Guardar a la base de dades la longitud del vídeos. Fins ara es reproduien sense saber la duració d'aquests.
- Disposem d'una aplicació per reiniciar la Raspberry des de *frontend*. Hem afegit avisos per fer notar a l'usuari que està executant una instrucció perillosa.
- Hem creat una pestanya a l'hora de crear una llista de reproducció per afegir-hi les etiquetes desitjades.

- Podem veure una pestanya que apareix si hem seleccionat mode intercalat perquè aparegui una pantalla que demani: “quin fitxer vols que sigui intercalat?” En pot seleccionar un de la llista o podem deixar que surti l’arxiu per defecte.

Bibliografia

Creative Commons Attribution-ShareAlike. (1 Juny, 2022). *Model–view–controller*. Wikipedia. https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Digital AV Magazine - Underwood Comunicación SL. (13 Juny, 2022). *Barcelona Metro bets again on dynamic advertising on a stretch of the L1*. Digital AV Magazine. https://www.digitalavmagazine.com/en/2020/03/10/metro-de-barcelona-apuesta-nuevo-publicidad-dinamica-en-tramo-l1/

Farrell, D. (s.d.). *Data Management With Python, SQLite, and SQLAlchemy*. Real Python. https://realpython.com/python-sqlite-sqlalchemy/

Metrópoli. (20 Febrer, 2022). *Aparece un mural de la candidatura olímpica del '92 en el metro de Barcelona*. Metropoli. https://www.metropoliabierta.com/vivir-en-barcelona/aparece-mural-candidatura-olimpica-92-metro-barcelona_50230_102.html

Núñez Delgado, V. (24 Gener, 2021). *Disseny i implementació d'un sistema de senyalització digital per al CRAI Biblioteca de Matemàtiques i Informàtica*. Dipòsit Digital de la Universitat de Barcelona. http://hdl.handle.net/2445/174208

OAuth 2.0. (s.d.). OAuth. https://oauth.net/2/

Pickle — Python object serialization. (s.d.). The Python Standard Library. https://docs.python.org/3/library/pickle.html

Rehkopf, M. (s.d.). *User stories with examples and a template*. ATLASSIAN Agile Coach. https://www.atlassian.com/agile/project-management/user-stories

Sigils. (4 Setembre, 2014). *Flask sqlalchemy many-to-many insert data*. Stack Overflow. https://stackoverflow.com/questions/25668092/flask-sqlalchemy-many-to-many-insert-data

Annexos

Endpoints Controlador

/account Accés limitat amb credencials: Si

GET: **/account/<string:username>**

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:

```
{
 "account": {
 "username": "user1",
 "is_admin": 0,
 }
}
```

400 COS DE RESPOSTA:

```
{
 "message": “Usuari no trobat ”
}
```

POST:

EXEMPLE COS DE PETICIÓ:

```
{
 "account": {
 "username": "user1",
 "password": s%GT/8jZG_J:3ac3,
 }
 auth:”eyJhbGciOiJIUzUxMiIsImlhdC...”
}
```

200 COS DE RESPOSTA:

```
{
 "account": {
 "username": "user1",
```

"is_admin": 0,
}
}
400 COS DE RESPOSTA:
{
"message": “Usuari no trobat ”
}
PUT:
EXEMPLE COS DE PETICIÓ:
{
"account": {
"username": "user1",
"password": s%GT/8jZac3,
}
auth:”eyJhbGciOiJIUzUxMiIsImlhdC...”
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament"
}
400 COS DE RESPOSTA:
{
"message": “Usari amb ['username': user1] no modificat "
}
DELETE: /account/<string:username>
200 COS DE RESPOSTA:
{
"message": “Usuari amb ['username': user1] esborrat correctament"
}
400 COS DE RESPOSTA:
{
"message": “Usuari amb ['username': user1] no trobat"
}
/**accounts** Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:

{ "accounts": [
{"username": "admin", "is_admin": 1},
{"username": "user3", "is_admin": 0},
{"username": "user6", "is_admin": 0}
]
}

400 COS DE RESPOSTA:

{
"message": “Llista d’usuaris buida”
}

/login

POST:

EXEMPLE COS DE PETICIÓ:

{"username":"admin","password":"s%GT/jZac3624hY2453"}

200 COS DE RESPOSTA:

{ "token": “eyJhbGci…….”
}

400 COS DE RESPOSTA:

{
"message": “Contrasenya incorrecta"
}

/**contents** Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:

{
"content": [
{

"id": 1, "name": "foto1.png", "path": "/assets/logo.png", "size": "10", "priority": 0, "duration": 0

}

]

}

400 COS DE RESPOSTA:

{

"message": “[] ”

}

POST:

EXEMPLE COS DE PETICIÓ:

{

"chunk": "45457fdsg566787909uiuyi8978056ugfef34gfs4rt45y7658578",

auth:”eyJhbGciOiJIUzUxMiIsImlhdC...”

}

200 COS DE RESPOSTA:

{

"message": “El fitxer {file.filename} guardat correctament”

}

400 COS DE RESPOSTA:

{

"message": “Hi ha hagut un problema: + exception.message”

}

DELETE:**/content/<int:id>**

EXEMPLE COS DE PETICIÓ:

{

"account": {

"username": "user1",

"password": s%GT/8jZac3,

}

auth:”eyJhbGciOiJIUzUxMiIsImlhdC...”

}

200 COS DE RESPOSTA:

{

"message": “El fitxer amb nom:" + name + "s'ha eliminat correctament"

}

400 COS DE RESPOSTA:

{

"message": “"El fitxer amb nom" + name + "no s'ha eliminat"

}

/thumbnail/<string:file> Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:

{<thumbnail file>}

400 COS DE RESPOSTA:

{

"message": “Thumbnail no trobat ”

}

/playlistlist Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:


```
"playlists": [
    {
        "id": 1,
        "name": "Llista 1",
        "items": [
            {
                "id": 1,
                "name": "item1.mp4",
                "type": "Image",
                "priority": null,
                "duration": 10,
                "path": "./static/item1.mp4",
                "played": 0
```

```
            },"played": 0
        ],
        "tags": [
            {
                "id": 1,
                "name": "covid"
            },
            {
                "id": 3,
                "name": "tag1"
            },
            {
                "id": 4,
                "name": "tag2"
            }
        ]
    },
}
```


400 COS DE RESPOSTA:


```
{
 "playlists": “[]”
}
```


/savePlaylist Accés limitat amb credencials: Si

POST:

EXEMPLE COS DE PETICIÓ:


```
{
    "name": "playlist99",
    "items": [
        {
            "name": "item1.jpg",
            "duration": 10,
            "type": "Image",
            "priority": 1,
            "played": 1
```

```
        },
        {
            "name": "item2.jpg",
            "duration": 1,
            "type": "Image",
            "priority": 1,
            "played": 1
        }
    ],
    "tags": [
        "tag1",
        "tag2"
    ]
}
```

200 COS DE RESPOSTA:

```
{
    "playlist": {
        "id": 36,
        "name": "playlist9",
        "items": [
            {
                "id": 2,
                "name": "item2.jpg",
                "type": "Image",
                "priority": null,
                "duration": 10,
                "path": "./static/item2.jpg",
                "played": 0
            },
            {
                "id": 14,
                "name": "item1.jpg",
                "type": "Image",
                "priority": null,
```

```
            "duration": 10,
            "path": "./static/item1.jpg",
            "played": 0
        }
    ],
    "tags": [
        {
            "id": 3,
            "name": "tag1"
        },
        {
            "id": 4,
            "name": "tag2"
        }
    ]
  }
}
```


400 COS DE RESPOSTA:


```
{
 "message": “Hi ha hagut un problema amb la petició ”
}
```


/items Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:


```
{
   "items": [
      {
         "id": 1,
         "name": "item1.mp4",
         "type": "Image",
         "priority": null,
         "duration": 10,
```

```
            "path": "./static/item1.mp4",
            "played": 0
        },
}
```


400 COS DE RESPOSTA:


```
{
 "items": “[] ”
}
```


/tags Accés limitat amb credencials: Si

GET:

EXEMPLE COS DE PETICIÓ:

BUIT

200 COS DE RESPOSTA:


```
{
    "tags": [
        {
            "id": 1,
            "name": "covid"
        },
        {
            "id": 2,
            "name": "estiu"
        },
}
```


400 COS DE RESPOSTA:


```
{
 "tags": “[] ”
}
```


Endpoints Reproductor

Docker Compose

```
version: '3'
services:
  billboard:
    build: ./Billboard WebApp
    volumes:
      - ./Billboard WebApp:/app
      - ./media:/app/static
      - ./models:/app/models
      - ./resources:/app/resources
      - ./database:/app/database
    ports:
      - "8000:8000"
    restart: "no"

  control:
    build: ./Controller WebApp
    volumes:
      - ./Controller WebApp:/app
      - ./media:/app/media
      - ./models:/app/models
      - ./resources:/app/resources
      - ./database:/app/database
    ports:
      - "80:80"
    restart:  "no"
```

Dockerfile Controlador

```
FROM python:3.7-alpine
RUN apk --update add --no-cache \
    lapack-dev \
    python3-dev gcc \
    freetype-dev \
```

```
    jpeg-dev zlib-dev libjpeg \
    gfortran musl-dev \
    ffmpeg
RUN apk add make automake gcc g++ subversion python3-dev
RUN mkdir /app
WORKDIR /app
ADD requirements.txt /app
ADD /server/app.py /app
RUN pip3 install -r requirements.txt
CMD  ["gunicorn",  "-w  4",  "-b",  "0.0.0.0:80",  "app:app",
"--timeout 20000"]
```

Dockerfile Reproductor

```
FROM python:3.7-alpine
RUN mkdir /app
WORKDIR /app
ADD requirements.txt /app
ADD app.py /app
RUN pip3 install -r requirements.txt
CMD ["gunicorn", "-w 4", "-b", "0.0.0.0:8000", "app:app"]
```

Manual de funcionament

Aclariments previs

Unitats d'emmagatzematge

- La targeta micro SD conté el sistema operatiu Raspbian i el programari fruit del TFG, que fa funcionar el sistema de senyalització. 6 4.1.2 La memòria USB El nom de la unitat ha de ser necessàriament PENDRIVE i ha d'estar formatada amb FAT32.
- La memòria USB conté dos directoris:
    - /database: conté la base de dades.

- /media: conté els fitxers multimèdia que es mostraran en pantalla. És convenient no accedir-hi mai directament: tot allò que calgui reproduir s'ha de gestionar mitjançant l'aplicació.

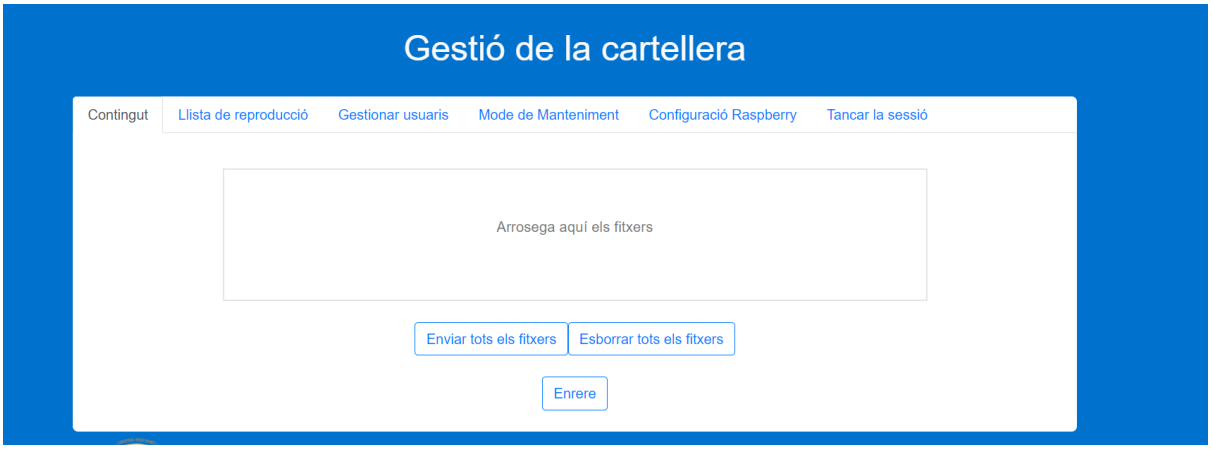
- **El fitxer default.mp4** Només hi ha una excepció a la norma anterior: el fitxer default.mp4, que s'ha de copiar directament a la memòria USB, dins del directori /media. Aquest fitxer no es mostrarà ni a la pestanya de contingut de l'aplicació ni a la llista de reproducció, però és el fitxer que es reprodueix a la pantalla en els següents supòsits:
    - Comença un cicle de reproducció.
    - Activem el mode de manteniment.
    - La llista de reproducció és buida.
    - Seleccionem un mode de reproducció intercalat però no hem denit el fitxer a intercalar. Aquest fitxer ha de ser un vídeo, en mp4 i amb el nom default.mp4. Convé que sigui representatiu de la Biblioteca perquè es mostrarà sovint; i que sigui més aviat permanent, perquè substituir-lo obliga a accedir a la memòria USB.
- El nom dels fitxers El nom dels fitxers a reproduir no pot contenir ni espais ni caràcters estranys. Si els conté, el navegador no els reproduirà: mostrarà una pantalla buida, en blanc.

Posada en marxa i apagada

- La Raspberry s'encén automàticament al connectar-la a la corrent. Per apagar-la o reiniciar-la trobem una pestanya que permet fer aquestes operacions sense necessitat d'accedir-hi físicament. En aquesta pestanya podem Apagar la Raspberry, Reiniciar la Raspberry, Apagar el Docker i Reiniciar el Docker. **Aquestes operacions només s'han de dur a terme en cas de fallada.**

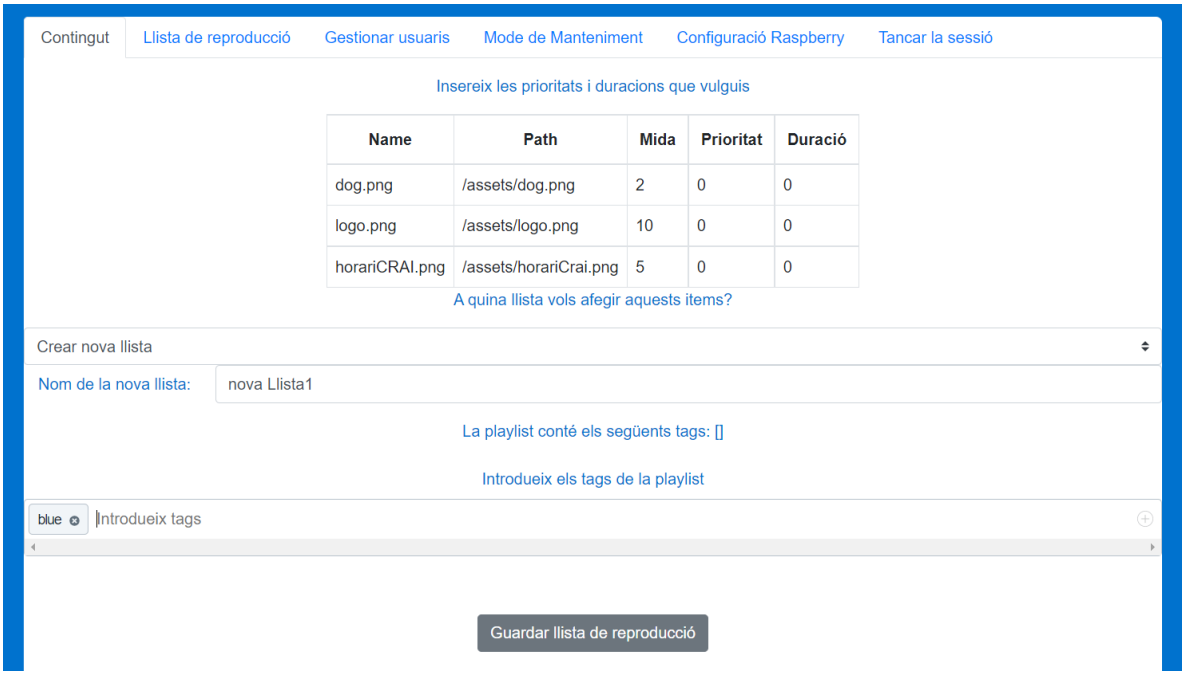
Un cop tenim la Raspberry connectada a la corrent, el programa es mostrarà automàticament la pantalla per iniciar sessió. Aquí només hem d'introduir un usuari i una contrasenya correctes i ja estem dins!

Quan hem introduit les credencials correctes, ja podem interactuar amb el progamari. Des d'aquí, podem accedir a la pestanya de "Contingut" i afegir contingut amb el botó "Afegir Contingut". Per afegir contingut només hem d'arrossegar o seleccionar un fitxer de la memòria interna de l'ordinador, com es mostra a la imatge:

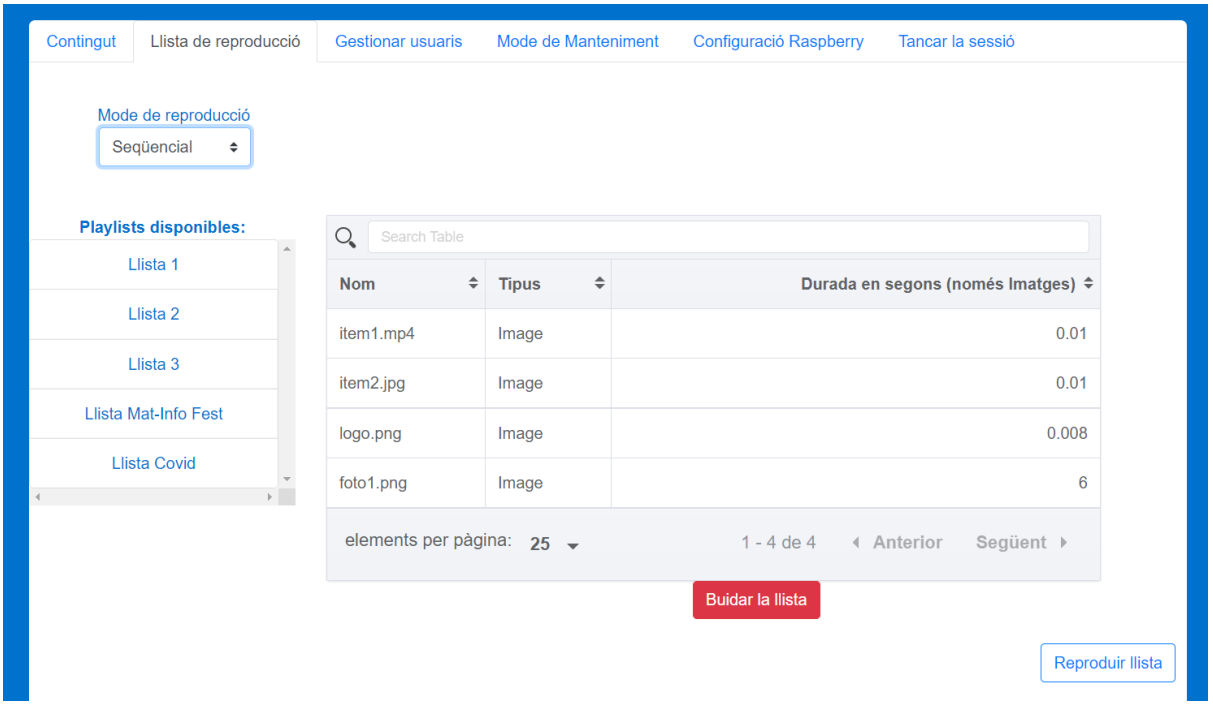


També podem seleccionem el fitxer o fitxers desitjats i clicar el botó “Afegir a una llista de reproducció” per afegir-los a la llista de reproducció desitjada.

Un cop tenim els fitxers, podem posar una llista ja existent o també en podem crear una de nova afegint el nom d’aquesta.



Si accedim ara a la pestanya Llista de reproducció, trobem a l'esquerra un selector on podem seleccionar qualsevol de les llistes creades. Quan en seleccionem una, se'ns actualitza la vista i se'ns mostren el Nom, el Tipus i la Durada de l'ítem. Llavors, si volem reproduir aquesta llista només hem de seleccionar el mode amb el selector de dalt a la dreta i clicar el botó "Reproduir llista de reproducció".



Si mirem la pantalla, veurem que ja s'ha començat a reproduir el contingut de la llista en funció del mode seleccionat.

Glossari

- **Back end**: Capa del software no visible per a l'usuari final que interactua amb les dades i operacions d'un sistema.
- **Front end**: Capa que veu directament l'usuari i amb la qual interactua.
- **User Story:** Descripció d'una funció de programari des de la perspectiva de l'usuari final plantejada prèviament al desenvolupament.
- **Sprint**: Al món del desenvolupament de software, és un període curt i limitat en el temps en què un equip (o individu) treballa per completar una quantitat determinada de treball.
- **Endpoint**: És el "punt de connexió" d'un servei, una eina o una aplicació a la qual s'accedeix a través d'una xarxa.
- **Bug**: Comportament inesperat, no desitjat o incorrecte produït per un sistema o programa informàtic.
- **Llibreria**: és un conjunt d'implementacions funcionals, codificades en un llenguatge de programació (normalment compilades), que ofereix una interfície definida per a la funcionalitat que s'invoca.
- **API** (Application Programming Interface): Interfície que permet comunicar diferents aplicacions perquè interactuïn entre elles.
- **Microservices**: Els serveis i funcions individuals (o blocs de construcció) que formen una aplicació més gran basada en microserveis.
- **Restful**: És un estil arquitectònic (al món del desenvolupament de software) i una visió de les comunicacions sovint utilitzat en el desenvolupament de serveis web individuals basat en la transferència d'estats representacionals (REST).
- **ORM**: És una tècnica de programació per convertir dades entre sistemes de tipus utilitzant llenguatges de programació orientats a objectes. Això crea, en efecte, una "base de dades d'objectes virtuals" que es pot utilitzar des del llenguatge de programació.
- **Docker**: És un conjunt de productes de plataforma com a servei que utilitzen la virtualització a nivell del sistema operatiu per oferir programari en paquets anomenats contenidors.