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

---

Vicent Núñez Delgado

Directors : Eloi Puertas i Prats
Roger Angela i Gambús
Realitzat a: Departament de
Matemàtiques i Informàtica

Barcelona, 24 de Gener de 2021

Resum:

Aquest TFG es proposa resoldre una necessitat específica formulada pel CRAI Biblioteca de Matemàtiques i Informàtica de la UB: implementar un sistema de senyalització digital amb programari lliure per a una TV ubicada al vestíbul, amb l'objectiu de mantenir als usuaris informats sobre tot allò que es consideri rellevant.

Per fer-ho, s'han desenvolupat un conjunt d'aplicacions web que, mitjançant una Raspberry Pi 4 connectada a la TV, permeten gestionar remotament els continguts a reproduir (imatge estàtica i vídeo) i visualitzar-los en un navegador maximitzat. El sistema permet alternar diversos modes de reproducció en funció de les necessitats i una gestió àgil dels continguts a través de la xarxa.

Abstract:

This TFG has the purpose to solve a specific need formulated by the CRAI Biblioteca de Matemàtiques i Informática of the UB: to implement an open-source digital signage system for a TV located in the lobby, to keep users notified about everything considered relevant.

To do this, a set of web applications has been developed, to run through a Raspberry Pi 4 connected to the TV, allows content (still images and videos) to be managed remotely, and viewed in a maximized browser. The system allows to alternate between the different reproduction modes depending on the needs and agile management of the contents through the network.

Resumen:

Este TFG se propone resolver una necesidad específica formulada por el CRAI Biblioteca de Matemàtiques i Informática de la UB: implementar un sistema de señalización digital en código libre para una TV ubicada en el vestíbulo, con el objetivo de mantener a los usuarios informados sobre todo lo que se considere relevante.

Para hacerlo, se han desarrollado un conjunto de aplicaciones web que, mediante una Raspberry Pi 4 conectada a la TV, permiten gestionar remotamente los contenidos (imágenes estáticas i vídeos) y visualizarlos en un navegador maximizado. El sistema permite alternar entre los diferentes modos de reproducción en función de les necesidades y una gestión ágil de los contenidos a través de la red.

Agraïments:

En primer lloc vull agrair als meus tutors d'aquest projecte, a L'Eloi i a en Roger per la seva ajuda, gran interès i suport que m'han aportat fins a l'últim moment.

També, vull agrair a tots els meus companys, amics i a la meva família, per recolzar-me quan els meus ànims dequeien. Especialment, vull fer esment dels meus pares, que sempre van estar per donar-me paraules de suport i una abraçada reconfortant per renovar energies.

Moltes gràcies a tots!

Índex

1. Introducció

En un món digital dominat per pantalles i on predomina la sobrestimulació visual, captar l'atenció de manera tradicional, amb imatges impreses a un cartell, doncs és més difícil que mai. No importen doncs els eslògans enganxosos o el nivell de polèmica que intenten crear: la brillantor i els colors dels miralls encesos (quan deixen de ser negres) ens roben l'atenció oblidant-nos de tot.
El següent projecte però no està enfocat a un dispositiu amb finalitat de vendre productes, sinó que intenta cobrir la necessitat d'actualitzar unes dades que estan constantment canviant i que queden en alguns casos, obsoletes en el mateix moment que s'escriuen al paper. S'espera, llavors poder elaborar un sistema que desperti fàcilment l'interès dels estudiants que acudeixen al CRAI amb un contingut permanentment actualitzat.

1.1 Motivació

La realització, ha estat motivada per el gran número d'aplicacions reals que pot tenir aquest projecte i l'abundància amb que trobem aquest concepte a la nostra societat. A qualsevol lloc (com ara un cinema, al transport públic, centres comercials...) podem trobar-nos una cartellera que pot mostrar diferents anuncis o missatges. Encara hi queden moltes d'estàtiques, cartells fixats a la paret, murals de suro per anuncis... però des de fa anys es veu una clara tendència per a la digitalització en aquest àmbit.
És lògic; l'espai físic es un bé preuat i en els casos on no es pot ampliar, s'intenta reaprofitar. Les ciutats van créixer en vertical quan per raons físiques o econòmiques no ho podien fer en horitzontal. El problema en l'àmbit de la publicitat o en la comunicació de missatges no es podia aprofitar d'aquesta solució; doncs cartells uns sobre altres impedeixen la visibilitat de tots. La solució doncs, passava per poder temporitzar quan de temps havia d'estar mostrant-se cadascun ja que no cal veure tots alhora sinó focalitzar l'atenció cada cop que canvia.
Al principi, degut a les limitacions tecnològiques, el concepte va passar per màquines mecàniques que tenien els missatges emmagatzemats en format de cilíndric i que anaven estirant i canviant cada cert temps. Un dels problemes però, va ser el fet del manteniment i al gestió del contingut. Requeria adaptar cada missatge, portar-lo presencialment... I si un contingut havia de ser renovat amb una certa freqüència doncs no era una bona solució.
Tot i així, gràcies a l'evolució de les pantalles a tecnologies més barates en quant a recursos econòmics i energètics va permetre utilitzar-les en aquest camp. No només això, sinó que a diferència de l'altre sistema amb imatges impreses, permetien mostrar contingut en moviment. Ja no calia una provocació o un eslògan efectiu per poder captar l'atenció; amb una imatge il·luminada amb moviment ja era suficient.
A dia d'avui, els avenços en aquest camp ja no tracten tant en l'evolució del mecanisme de mostra (ja que ve donat per les tecnologies de les pantalles en si) sinó en la gestió del contingut i el control del sistema informàticament.

1.2 Problema a tractar

Deixant a banda tots els motius històrics de les cartelleres i de la publicitat, el problema era el següent:
Es volia utilitzar l'espai del vestíbul de la biblioteca de Matemàtiques i Informàtica per ensenyar multimèdia relacionada amb activitats que es fan a la facultat, per mostrar missatges informatius (molt necessari sobretot en una situació com la d'aquest curs 2020-2021) o promoure continguts disponibles a la biblioteca... mitjançant imatges i vídeos.
La solució més adient va ser l'ús d'una Smart tv (com a pantalla) i d'un dispositiu extern de reproducció. Aquesta decisió la va prendre el personal del CRAI de la biblioteca de Matemàtiques i Informàtica prèviament al plantejament d'aquest projecte. Va ser doncs quan va sorgir la idea de recórrer a una solució desenvolupada a mida (a nivell de software) per executar dins d'aquest dispositiu reproductor. Aquesta, havia de poder gestionar el contingut i també es clar, el control d'aquest sistema de cartellera.
L'objectiu d'aquest projecte és doncs la creació del programari per a que el client final pugui disposar d'un sistema de senyalització digital.

1.3 Estructura de la memòria

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

Treballs Previs: Recerca prèvia a la realització del sistema per tal d'orientar el projecte veient les opcions que hi ha al mercat i desenvolupar una possible alternativa.

Anàlisis: Plantejament i explicació (des del punt de vista del client com a usuari) dels requeriments en format de *user stories*.

Planificació: Distribució de les tasques en els períodes establerts de desenvolupament del producte (*sprints*).

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.

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.

2. Treballs previs

Per tal de poder dur a terme el projecte, es va fer una recerca sobre les alternatives disponibles al mercat. En aquest capítol, es troba aquest procés explicat.

2.1 Solucions completes de senyalització

S'ha de dir que no es una tasca fàcil trobar informació concreta degut a la gran quantitat de solucions que hi ha a internet. A més a més, moltes d'aquestes pertanyen a empreses poc conegudes i no destinades als "usuaris finals", sinó a donar serveis per altres empreses de manera personalitzada; fent per tant, que trobar informació sobre el *hardware*, *software* i el preu de cadascuna fos una tasca impossible en alguns casos.
De les solucions trobades, podem dividir-les en dos tipus: Les que inclouen una pantalla i les que no.
És molt possible que si es disposa d'una façana d'un edifici i es vol omplir, per exemple , amb una cartellera de 5x5 metres, es busqui una solució completa de la pantalla , el *software* i *hardware* de reproducció. Però en molts casos, com era la necessitat del projecte, només es vol cobrir una "petita" superfície, sent un monitor o televisió, candidats ideals. Petita en comparació a l'exemple de la façana, perquè amb la tendència de fer més grans les televisions, podem trobar fàcilment i a bon preu pantalles de 1,46 x 0,85 m (65 polzades de diagonal aproximadament).
Si encara calgués més superfície de pantalla, segons la recerca feta, es pot trobar pantalles de conegudes marques comercials (sense que sigui personalitzat, sinó que a qualsevol botiga es podria aconseguir) de fins a 86" de diagonal.

Solucions per a superfícies molt grans són per exemple les de Prismaflex. Aquesta empresa francesa ofereix tant serveis de *hardware* com *software*. A la seva web es pot trobar pantalles fins a 81m2 i el *software* per poder utilitzar-les. No donen cap més informació sobre quotes o preus però insisteixen molt en contactar amb el departament de ventes en cas d'estar interessats. Tot i no donar molta informació sobre les tecnologies utilitzades (a banda de dir que les seves pantalles són LED) donen un esquema conceptual del sistema:

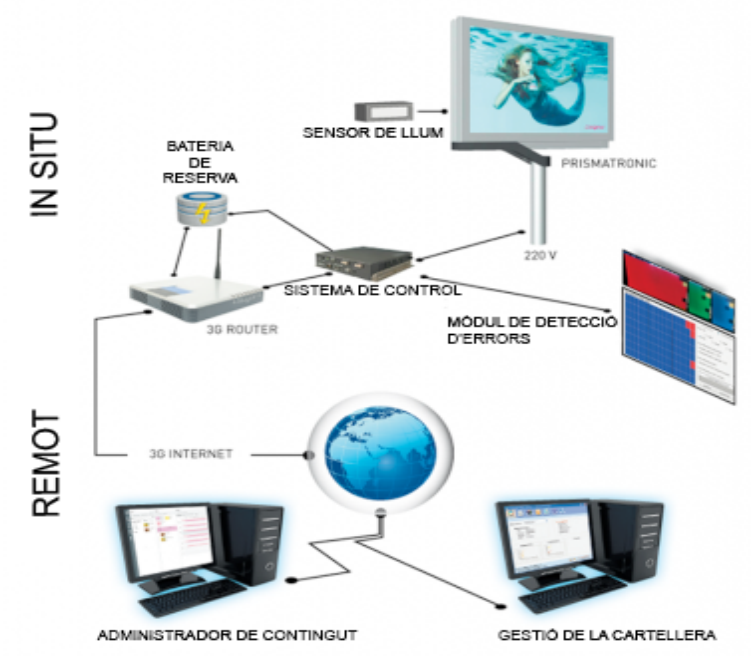


Figura 2.1: Esquema conceptual de la solució de prisma flex
(Font:https://www.prismaflex.com/files/2013/06/schemaprismatronicFR-376x400.png)

Cal destacar que aquest diagrama mostra una solució que està pensada per a llocs exteriors; ideal per exemple per un cartell interactiu a un carretera gestionant el contingut remotament requerint una connexió a internet. Per tal de funcionar en un entorn com el requerit per el projecte (cartellera connectada i gestionada localment per un ordinador) caldria connectar directament el "*systeme de controle*" al *router* local que el comunicaria amb els ordinadors de la biblioteca o l'entorn en que es trobés.

2.2 Reproductors per a sistemes de senyalització

2.2.1 Solucions de *Hardware + Software*

Degut a que l'espai desitjat per a col·locar-la era més petit d'això es va optar per una Smart Tv de 43" amb ports d'HDMI per a connectar un dispositiu perquè es pogués gestionar el contingut a mostrar. Les possibilitats doncs estaven reduïdes a desenvolupar el programari d'aquest dispositiu reproductor. La figura 2.2 doncs, il·lustra el concepte:

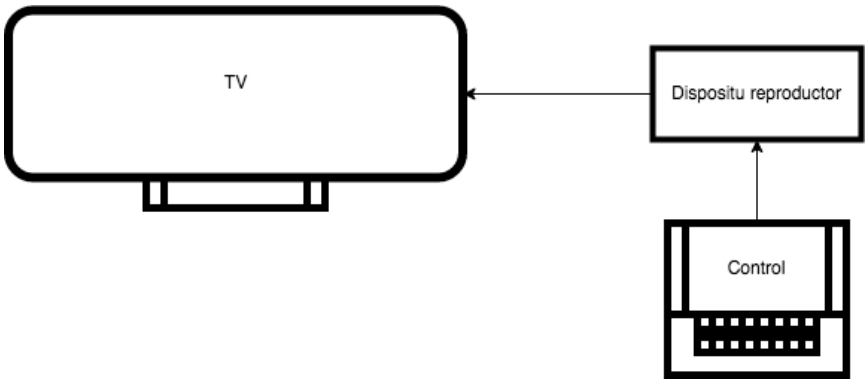


Figura 2.2: Esquema de connexions del sistema de la cartellera (Font: Elaboració pròpia)

L'alternativa més similar a la solució que s'ha realitzat finalment és un reproductor de la companyia BrightSign. Al vestíbul de la facultat de Matemàtiques i Informàtica ja n'hi ha un instal·lat a l'Smart Tv, en concret el model HD223. Es tracta d'un reproductor programable remotament i permet mostrar el contingut en una pantalla connectada directament o fent ús d'*streaming* per IP. Aquesta darrera variació comporta que a les pantalles (si s'utilitzen més d'una) tinguin un navegador amb reproductor compatible amb multimèdia codificada en h264 i h265. El *software* i *hardware* es ven conjunt per un preu de 350$ a la pàgina oficial. Es volia doncs aconseguir aproximació amb la Raspberry. A la figura 2.3 podem veure una comparació del successor del HD223 (ja que ha sigut descatalogat per introduir el HD224) amb la Raspberry a nivell de funcionalitats i connexions.

Dispositiu:	Raspberry Pi 4	HD224
Connexió a internet:	Via Ethernet i WIFI	Només disposa d'Ethernet
Sortida de vídeo:	2 ports micro-HDMI amb suport per 2 monitoris alhora. Màxima resolució: 4K 60hz. Sortida de vídeo també per el jack de 3,5mm	Sortida HDMI 2.0a , Màxima resolució suportada: 4096x2160
Connexions de sortida d'àudio:	Sortida d'àudio i vídeo per jack de 3,5mm compost	Sortida d'àudio per jack de 3,5mm
Altres connexions:	Inclou antena Bluetooth, 4 ports USB (2x 2.0, 2x 3.0)	Entrada receptor infrarojos per jack 3,5mm i un port usb 3.0
Descodificació de h264	Sí, per *hardware*(fins a 1080p60hz)	Sí, per *hardware* (fins a 1080p60hz)
Descodificació de h265	Sí, per *hardware* (fins a 4k60hz)	Sí, per *hardware* (fins a 4k60hz)
Pes:	46 grams + 39 grams de la caixa	340 grams
Mesures:	70 x 94,9 x 29.972 mm	159.9 x 22 x 144.2mm
Adaptador de corrent:	USB-c amb 5V at 3A (15 Watts)	Connector propietari amb 12V / 1.5A (18 Watts)
Preu (Aproximat):	1 GB RAM 44€, 4 GB 59€ i 8 GB 79€	350$ - 289.69€

Figura 2.3: Taula comparativa entre els dos dispositius (Font: Elaboració pròpia)

A grans trets, podem observar que la Raspberry és una bona competidora, de fet, supera en alguns d'aquests apartats a la solució de BrightSign. Més lleugera, més petita, amb més ports... Quant a compatibilitat de resolucions i descodificació, estan al mateix nivell.
El problema però es que l'HD224 és com una caixa negra de cara a la venta; s'expliquen connexions, compatibilitat però no podem saber ni el *hardware* (a nivell de CPU o memòria) ni el *software* empleats degut a

la seva llicencia privada. Això, dificulta la comparació ja que només podem basar-nos en coses com "Si suporta descodificació del *codec*, per *hardware*..." i no podem buscar *benchmarks* del processador o del sistema operatiu a Internet.
Una altra alternativa també privada amb *software* + *hardware* és mediaBOX-200 de DigitalSignage. No és tan econòmica (val uns 595$) i la resolució màxima suportada és 1080p. En aquest cas, sí que donen més informació sobre el sistema: utilitza una versió *embebed* de Windows 10 sobre un processador i3. Inclou diferents programaris: un per tal de poder mostrar el contingut i un altre per gestionar-lo. El fet de donar les especificacions es perquè a la mateixa web informen que aquesta solució (a nivell de *hardware*) fa ús d'un Intel NUC, de manera que podríem convertir qualsevol ordinador d'especificacions similars de venta en qualsevol lloc, en un producte com el que ofereixen amb la llicència del programari que es pot adquirir a la seva web. Això és degut a que el principal negoci de l'empresa és el *software* en si més que no pas el *hardware* , que de fet ni el fabriquen ells.

2.2.2 Solucions de *Software*

Si per tal de trobar una solució com les anteriors (*hardware* + *software*) ha sigut més difícil, per trobar només el *software* partint d'un NUC / Barebone PC ja comprat, existeixen molt més alternatives. Entre les quals, destaquen per número d'usuaris Xibo, RiseVision i WireSpring. Aquests programes són molt similars entre ells, però les principals diferències com podem veure a la figura 2.4 són:

	Xibo	RiseVision	WireSpring
Sistema operatiu per al programari de gestió:	Windows	Independent, és una aplicació Web	Windows, Mac
Característiques principals:	-Programació de contingut -Suport multi-pantalla	-Programació de contingut -Gestió de fitxers -Creació de panells personalitzats amb HTML -Suport multi-pantalla	-Programació de contingut -Suport multi-pantalla
Preu:	19€ llicència completa / Subscripció per 1€ al mes	Des de 121 € per 1 dispositiu a l'any fins a 8.796€ per 100 dispositius a l'any.	350€ llicència completa (1 dispositiu)
Limitacions de reproducció:	Descodificació per *software* fins a 4k 60hz Resolució màxima 4k 60hz	Descodificació per *software* fins a 4k 60hz Resolució màxima 4k 60hz	Descodificació per *software* fins a 4k 60hz Resolució màxima 4k 60hz

Figura 2.4: Taula comparativa entre Xibo, RiseVision i WireSpring
(Font: https://www.softwaresuggest.com/compare/xibo-vs-rise-vision)

Buscant una solució molt més econòmica encara, vaig trobar Screenly. Té característiques similars als *softwares* mencionats anteriorment, però aquest també es troba disponible per a plaques Raspberry pi; mentre que els altres necessiten un dispositiu amb arquitectura x86 d'ordinador "normal".
La diferència de preu entre un NUC (de x86) i la Raspberry és molt notable, sent aquesta última 4 vegades més econòmica com a mínim. El problema d'Screenly però és, que el conjunt de programari per a Raspberry té una quota de 30€ al mes segons la pàgina oficial. Si bé és cert que hi ha una versió *open source* (a diferència de les opcions anteriors que no en tenen), aquesta té limitacions en comparació amb la versió normal (a data d'avui) com ara el número de fitxers per mostrar, la resolució d'aquesta, limitacions per programar el contingut, no poder treballar sense internet, absència de suport tècnic... A banda d'haver de compilar el codi font per utilitzar-la; sent necessari tenir coneixements de programació informàtica i de sistemes Linux en general.

3. Anàlisi

En aquest capítol s'expliquen tots els requeriments que el client esperava veure al producte final i com s'esperava adaptar-los a funcionalitats més concretes mitjançant *User Stories*.

3.1 Requeriments

A partir dels requeriments parlats en les reunions sobre el projecte a principis de setembre, vaig elaborar el següent llistat:
Requeriments:
- A la cartellera (mitjançant l'aplicació del dispositiu) s'han de poder visualitzar imatges i opcionalment poder visualitzar documents.
-A la cartellera (mitjançant l'aplicació del dispositiu) s'han de poder reproduir vídeos.
- El dispositiu ha de tenir una llista de reproducció que contingui els elements a reproduir.
- Tenir emmagatzemats els fitxers multimèdia i la llista en una memòria extraïble connectada directament al dispositiu.
- Poder eliminar contingut prèviament emmagatzemat.
- Controlar la cua de reproducció sense accedir físicament al dispositiu.
- Que el sistema estigui basat en aplicació web, obtenint compatibilitat multi plataforma.
- Programar l'inici i l'aturada de la cartellera perquè funcioni amb l'horari del CRAI.

El fet de demanar expressament que la cua i els fitxers a reproduir s'emmagatzemin en un dispositiu extern com ara un disc dur (amb connexió USB) o un *pendrive*, ve donat per diversos motius: el primer de tots és per seguretat; si la Raspberry deixa de funcionar, la llista de reproducció i les altres dades (incloent els fitxers multimèdia) queden guardats.
Al restablir doncs el programari i/o el *hardware*, el contingut de la cartellera quedaria intacte. Per altra banda també el fa portable: si hi ha més d'un sistema, es poden intercanviar els *pendrives* i funcionar amb total normalitat. L'últim motiu, que també és important: l'emmagatzematge extern es pot ampliar i així no caldria dependre de la memòria disponible de la Raspberry per emmagatzemar contingut nou.

Controlar la interfície sense accedir-hi físicament, és un requisit per tal de poder modificar el comportament de la llista i els seus elements sense connectar un teclat i/o un ratolí o similar. De fet, es va acordar fer aquesta gestió des de el navegador d'un altre dispositiu connectat a la mateixa xarxa de la biblioteca. Això ajuda a futures ampliacions; si es volgués fer una gestió remota, amb configurar la xarxa per poder accedir a la Raspberry des de fora del CRAI, es podria fins i tot controlar la cartellera des de qualsevol lloc del món.

Que el sistema estigui basat en una aplicació web té també molts avantatges. El primer és la compatibilitat amb tots els sistemes operatius, no forçant a utilitzar-ne un per tal de poder controlar la cartellera. Aquesta compatibilitat però no és només amb sistemes operatius actuals sinó que està assegurada també amb els del futur (sempre i quan el navegador sigui compatible amb les tecnologies emprades). També, tot i que la interfície d'usuari no ha tingut aquest requeriment en compte i no està adaptada a versions mòbils o tauletes, no seria un problema de cara a una futura ampliació fer petites modificacions a la part de *frontend* per aconseguir -ho.

També, hi havia Requeriments relacionats amb la seguretat:

- Que cap persona externa al CRAI Biblioteca de Matemàtiques i Informàtica pugui modificar el contingut.
- Que no es pugui alterar l'ordre de reproducció sense estar autoritzat al sistema.
- Que cap persona externa puguin canviar el comportament base del dispositiu (denegar l'accés a modificar el funcionament del sistema operatiu de la placa en si).
- Que no puguin fer-la servir remotament per reproduir contingut extern (per exemple que una persona aliena mostri els seus propis fitxers a la cartellera sense permís).

Degut al lloc on es troba es un actiu valuós com a eina de difusió. Per això es van demanar tants requeriments relacionats amb la seguretat projecte; tot i que al final es poden solucionar controlant-ne l'accés.

3.2 *Product Backlog*

A partir dels requeriments anteriorment llistats, vaig concretar les *User Stories* a la taula de *Product Backlog* de la figura A1 que es troba en els annexos.
Per tal d'ajudar a entendre les *Users Stories* i la relació que tenen amb els llistats de requeriments anterior, he dividit la taula en fragments visibles en les següents figures. Algunes de les *User Stories* són "èpiques"; vol dir que són molt grans, n'engloben d'altres i no són molt especifiques de cara al desenvolupament. En el món del *software* un clar exemple és el fet de dir: com a usuari d'un sistema operatiu, vull accedir a l'escriptori amb el ratolí per tal de veure, modificar, comparar... els elements. Aquest "desig" que pot semblar simple, inclou doncs també que s'ha de disposar d'un entorn d'escriptori gràfic (l'usuari demana específicament que ha de poder utilitzar el ratolí, descartant doncs un entorn basat en terminal ja que aquestes operacions no es poden fer amb aquest perifèric per consola), d'un sistema de fitxers..., requerint molts factors previs i moltes "subtasques" dins d'aquesta mateixa *User Story* per tal de completar-la.

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
TOTS	Veure el contingut per pantalla		Sí		US01
Usuari	Visualitzar documents temporalment	Veure'ls a la cartellera	No	Es mostra el document cada cert temps i després salta al següent element de la llista	US13
Usuari	Visualitzar temporalment imatges	Veure-les a la cartellera	Sí	Es mostra la imatge cada cert temps i després salta al següent element de la llista	US14

Figura 3.1:Fragment del *Product Backlog* que conté les *User Stories* 1, 13 i 14 (Font: Elaboració pròpia)

En aquest cas, com podem veure a la figura 3.1, la primera *User story* és d'aquest tipus. "Veure el contingut per pantalla" implica moltes "subtasques" per tal de completar-la. Veure el contingut segons el requeriments vol dir que l'aplicació de visualització, que s'executa dins del navegador, reprodueixi de manera maximitzada un fitxer multimèdia. Això implica que s'havia d'elaborar una aplicació que funcionés al navegador i que alhora tingués un reproductor de vídeos.
Si a la primera no ho especificava per ser massa general, en aquest cas calia doncs més concreció sobre els tipus de fitxers: "Visualitzar documents temporalment" i "Visualitzar imatges temporalment". La idea és doncs que passat cert temps predefinit, la cartellera mostri el següent element de la llista ja que per defecte aquest tipus de fitxers, són estàtics.
Aquestes US estan directament elaborades a partir del requeriment: "A la cartellera (mitjançant l'aplicació del dispositiu) s'han de poder visualitzar imatges i opcionalment poder visualitzar documents".

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Tenir una interfície de control de la cartellera al meu navegador	No haver d'accedir físicament al dispositiu per tal de modificar el comportament de reproducció	Sí		US17

Figura 3.2: Fragment del *Product Backlog* que conté la *User Story* 17 (Font: Elaboració pròpia)

La següent *User Story* de la Figura 3.2 "tenir una interfície de control de la cartellera al meu navegador" és també èpica ja que de manera similar l'usuari vol una cosa molt general, que requerirà moltes més subtasques i *Users Stories* per aconseguir-ho. Tot i així, aquesta US és un concepte important ja que en cas de no existir, s'entendria que l'usuari hauria d'estar físicament davant del dispositiu, connectat a la cartellera amb els perifèrics, per fer qualsevol modificació.

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Enviar contingut al dispositiu de la cartellera des de l'app de gestió	Poder reproduir-lo	No	L'arxiu que l'usuari vol reproduir es troba a la targeta SD de la Raspberry	US02
Usuari	Veure el contingut	Poder	Sí/No	Es veu que es corresponen els fitxers,	US03

	que hi ha al dispositiu des de el meu ordinador	gestionar-lo		accedint físicament al dispositiu, amb el llistat que tenim a l'aplicació de gestió	
Usuari	Eliminar contingut del dispositiu de la cartellera	No disposar-ne més	No	Que es pugui comprovar que l'arxiu no és físicament a l'emmagatzematge de la Raspberry	US04

Figura 3.3: Fragment del *Product Backlog* que conté les *User Stories* 2, 3 i 4 (Font: Elaboració pròpia)

La US02 de la figura 3.3 està molt relacionada amb la US17 de la figura 3.2; de fet, *en* forma part. Aquesta elimina tots els dubtes sobre on es troben els fitxers per a la reproducció i com s'agreguen (es passen manualment amb un dispositiu USB? No, s'envien per la xarxa local a través de l'ordinador). "Eliminar contingut del dispositiu de la cartellera" és realitzable només després d'haver implementat l'anterior (US02,visible a la figura 3.3) i la US03, "Veure el contingut que hi ha al dispositiu des de el meu ordinador" per raons evidents. El motiu de la seva existència és el mateix que les anteriors: no haver de fer les operacions directament al dispositiu sinó a través de la interfície de control des d'un ordinador del CRAI.
Aquestes US estan directament elaborades a partir dels requeriments: "Tenir emmagatzemat els fitxers multimèdia i la llista en una memòria extraïble connectada directament al dispositiu." i "Poder eliminar contingut prèviament emmagatzemat."

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Que no tots els elements enviats al dispositiu s'afegeixin a la cua de reproducció	Tenir més control sobre la reproducció	Sí	A l'enviar un fitxer multimèdia mitjançant l'app des del navegador, no s'afegeix directament a la llista de reproducció	US18
Usuari	Veure la llista de reproducció	Saber l'ordre de reproducció	Sí	Observar que l'ordre dels fitxers que s'estan mostrant a la pantalla coincideix amb el que mostra la llista	US05
Usuari	Treure elements de la cua de reproducció	No reproduir-los més	No	A l'eliminar el fitxer de la llista, després de reproduir-la sencera, no s'ha visualitzat l'element cap cop	US07

Figura 3.4: Fragment del *Product Backlog* que conté les *User Stories* 5, 7 i 18 (Font: Elaboració pròpia)

La US18 de la figura 3.4 ("Que no tots els elements enviats al dispositiu s'afegeixin a la cua de reproducció"), és un punt clau, ja que no sempre es volen tenir tots els elements al reproductor a la cua de reproducció. Hi ha d'haver doncs una distinció entre el contingut i la llista de reproducció. No tindria sentit unificar-ho i que cada cop que es volgués treure de la llista un element, s'hagués d'eliminar físicament de la memòria de la Raspberry, obligant a enviar-lo de nou cada cop que es volgués tornar a veure. Seria una feina innecessària.
Aquesta *User Story* dona pas doncs a dues més: "Veure la llista de reproducció" (US05) per tal de saber quins elements multimèdia es troben a la cua i "Treure elements de la cua de reproducció"(US07). Feta la distinció entre cua i contingut, s'han de poder treure elements de la reproducció però sense eliminar-los del dispositiu: potser es volen fer servir més endavant.

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Seleccionar un contingut per reproduir a continuació del que s'està reproduint	Imposar un fitxer immediatament després	No	Es reprodueix, seguit del fitxer actual, un altre fitxer seleccionat prèviament	US06
Usuari	Programar la reproducció d'un element donada una data	Reproduir un fitxer en un dia i hora concrets	No	Que es pugui visualitzar el contingut desitjat en la data prèviament especificada.	US08

Figura 3.5: Fragment del *Product Backlog* que conté les *User Stories* 6 i 8 (Font: Elaboració pròpia)

Hi ha casos en què l'usuari vol reproduir un contingut immediatament després de l'actual, encara que no fos el següent de la llista. Pot ser degut a un esdeveniment imprevist o simplement per fer una prova i veure com

queda un element determinat en pantalla. Per això, existeix doncs la *User Story* 06 (visible a la figura 3.5) "Seleccionar un contingut per reproduir a continuació del que s'està reproduint". Per contra doncs, si es coneix amb antelació un esdeveniment i s'ha de mostrar contingut (per exemple, un missatge sobre activitats a la Facultat) en una data concreta, cal una altra *User Story* (la nº08): "Programar la reproducció d'un element donada una data".
Tant les *User Stories* reflectides a la figura 3.4 com a la 3.5, formen part del mateix requeriment: "El dispositiu ha de tenir una llista de reproducció que contingui els elements a reproduir."

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Iniciar sessió	Accedir a l'aplicació	Sí	L'usuari inicï sessió amb unes credencials i només així pugui veure la interfície de control	US15
Usuari	Tancar la sessió	Sortir de l'aplicació	Sí	L'usuari tanqui la sessió i deixi de veure la interfície de control	US16

Figura 3.6: Fragment del *Product Backlog* que conté les *User Stories* 15 i 16 (Font: Elaboració pròpia)

Quant a la seguretat, a nivell d'usuari només he plantejat dues *User Story*; "Com a usuari vull iniciar sessió" i "Com a usuari vull tancar la sessió". Visibles a la figura 3.6 amb identificador 15 i 16 respectivament.
Podem limitar l'ús de la interfície de control perquè només hi tingui accés el personal del CRAI amb un control d'accés mitjançant credencials. Aquesta *User Story* també es èpica ja que per tal de desenvolupar-la es dividiran en més concretes durant la fase d'implementació.

Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació	US
Usuari	Apagar el dispositiu de la cartellera remotament	No haver d'apagar el dispositiu amb perifèrics connectats	No	Que el dispositiu s'apagui des de l'ordinador de control	US09
Usuari	Reiniciar el dispositiu de la cartellera remotament	Tornar a tenir el sistema en funcionament en cas d'aturada	No	Que el dispositiu s'apagui des de l'ordinador de control i torni a iniciar l'app de la cartellera automàticament	US10
Usuari	Programar l'apagada del dispositiu de la cartellera	No haver d'apagar el dispositiu manualment	No	Que a l'hora indicada el dispositiu deixi de funcionar	US11
Usuari	Programar l'inici del dispositiu de la cartellera	No haver d'iniciar el dispositiu manualment	No	Que a l'hora indicada el dispositiu s'engegui i automàticament obri l'app de la cartellera, començant la reproducció	US12

Figura 3.7: Fragment del *Product Backlog* que conté les *User Stories* 9, 10, 11 i 12 (Font: Elaboració pròpia)

Finalment, trobem a la figura 3.7 les *User Stories*: "Apagar el dispositiu de la cartellera remotament" ,"Reiniciar el dispositiu de la cartellera remotament" ,"Programar l'apagada del dispositiu de la cartellera" i "Programar l'inici del dispositiu de la cartellera". Totes aquestes, reflecteixen el requeriment: "Programar l'inici i l'aturada de la cartellera perquè funcioni amb l'horari del CRAI."
Això és degut a que també es vol controlar l'inici i l'aturada del dispositiu de la cartellera de manera telemàtica, sense haver de connectar perifèrics per fer aquesta operació que es duu a terme diàriament. Es vol aconseguir aquest propòsit de dues maneres: de manera temporitzada, ja que el CRAI té un horari definit; i també de manera "manual" per tal d'apagar i tornar a tenir el sistema en funcionament en cas d'una aturada molt greu. En el cas d'engegar, potser poder utilitzar-la fora de l'horari establert, de manera puntual.

4. Planificació

En aquest capítol es pot veure la previsió de la distribució del treball per a dur a terme el seu desenvolupament.

4.1 En context sobre la distribució

Per elaborar el projecte, he disposat d'un període de 3 mesos aproximadament (des de la primera reunió el 25 de Setembre fins al 18 de Desembre). Per distribuir-la vaig planificar la realització de les diferents *user stories* en 6 s*prints*. Aquestes divisions de temps constaven de 14 dies i en arribar l'últim, es feia una reunió amb dues parts: Una part anomenada "*Sprint Review*" on els tutors veien una demostració en viu del projecte i valoraven la feina feta: com havia desenvolupat les *Users Stories* a nivell de disseny i implementació, i de totes les escollides per realitzar, si s'havien acabat o no segons els criteris d'acceptació prèviament establerts. També, però és parlava sobre els problemes sorgits, possibles millores sobre el realitzat i d'altres observacions.
L'altre part de les reunions anomenada "*Sprint Planning*" consistia, (en funció del parlat anteriorment a l'altre part) en decidir quines serien les *User Stories* escollides per desenvolupar en *l'Sprint* a començar i en cas de ser necessari, la manera de realitzar-les tant a nivell de disseny, com d'implementació. Ja que tot i estar planificats des de un principi els *sprints*, aquesta cronologia s'acaba alterant al llarg del temps degut a que sorgeixen imprevistos o noves *User Stories* al desenvolupament.
Per exemple, fins que no s'ha realitzat "veure un llistat de fitxers" , l'usuari no veu que potser seria útil tenir un cercador; per això, s'elabora una nova *User Story* i s'afegeix a la llista d'US (tenint en compte sempre la prioritat)per al pròxim *Sprint*.

4.2 Distribució temporal de les *User Stories*

Per poder dividir en *sprints* les *user stories*, primer es van ordenar en funció de la seva prioritat, ja que el codi de numeració USXX correspon a l'ordre amb què les vaig redactar a partir dels requeriments analitzats, sense tenir en compte l'ordre d'implementació. No és possible, per exemple, implementar que l'usuari pugui "iniciar sessió" (US15) sense "tenir una interfície de control" (US17) primer, entre altres.

User Story	Com a usuari vull...	Estimació en hores
US01	Veure el contingut per pantalla	30
US17	Tenir una interfície de control de la cartellera al meu navegador	13
US02	Enviar contingut al dispositiu de la cartellera des de l'aplicació de gestió	17
US03	Veure el contingut que hi ha al dispositiu des de el meu ordinador	8
US04	Eliminar contingut del dispositiu de la cartellera	4
US14	Visualitzar temporalment imatges	10
US13	Visualitzar documents temporalment	6
US18	Que no tots els elements enviats al dispositiu s'afegeixin a la cua de reproducció	21
US05	Veure la llista de reproducció	9
US07	Treure elements de la cua de reproducció	2
US06	Seleccionar un contingut per reproduir-lo a continuació del que s'està reproduint	6
US15	Iniciar sessió	10
US16	Tancar la sessió	2
US08	Programar la reproducció d'un element donada una data	6
US09	Apagar el dispositiu de la cartellera remotament	2
US10	Reiniciar el dispositiu de la cartellera remotament	1
US11	Programar l'apagada del dispositiu de la cartellera	1
US12	Programar l'inici del dispositiu de la cartellera	1

Figura 4.1: Fragment de la Taula de les *User Stories Product Backlog* (Font: Elaboració pròpia)

En aquesta Figura 4.1, podem veure dues columnes extretes de la taula del *product backlog* pertanyent als annexos com a Figura A2. Com es pot veure, ara en comptes de ser ordenada per la columna *user story* com es troba a la figura de l'Annex, ja s'ha tingut en compte la prioritat de cadascuna.

L'agrupació per a cada *Sprint* d'aquestes US es pot veure en el següent diagrama de Gantt de la Figura 4.2:

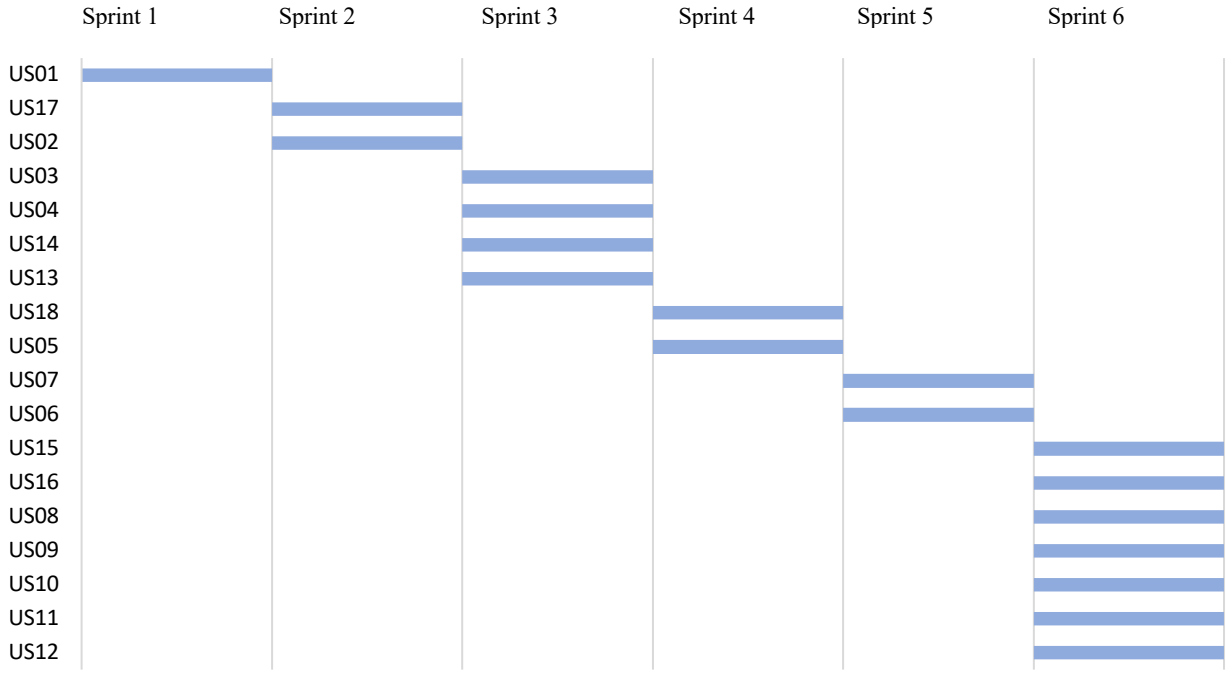


Figura 4.2: Diagrama de Gantt de les User Stories per Sprint (Font: Elaboració pròpia)

Dividint el període del 25 de setembre fins al 18 de desembre en grups de 14 dies van sortir els següents *sprints*:

- *Sprint* 1:Del 25 de setembre al 9 d'octubre.
  Estimació de 30 hores.
- *Sprint* 2: Del 9 d'octubre al 23 d'octubre.
  Estimació de 17 + 13 = 30 hores.
- *Sprint* 3: Del 23 d'octubre al 6 de novembre.
  Estimació de 8 + 10 + 4 + 6 = 30 hores.
- *Sprint* 4: Del 6 de novembre al 20 de novembre.
  Estimació de 21 + 9 = 30 hores.
- *Sprint* 5: Del 20 de novembre al 4 de desembre.
  Estimació de 2 + 6 = 8 hores. La resta, de marge per solucionar els errors trobats en la part de *testeig* amb el client.
- *Sprint* 6: Del 4 de desembre al 18 de desembre.
  Estimació de 10 + 6 + 2 + 2 +1 +1 +1 = 23 hores.

El número de *user stories* seleccionades per a desenvolupar varia molt a cada *sprint* com es pot apreciar al diagrama de la Figura 4.2. La xifra depèn del temps estimat de cadascuna segons les seves tasques associades. Per exemple, al primer *sprint* només es va seleccionar una única *User Story*: "Veure el contingut per pantalla". Tenint en compte que no hi havia cap codi previ, les tasques associades, doncs, al primer *sprint* eren moltes: s'havia de crear l'entorn de *Docker*, crear l'aplicació i l'HTML que permet la reproducció... En canvi, n'hi ha d'altres que contenen *user stories* com ara "Eliminar contingut del dispositiu de la cartellera" que no tenen tantes tasques associades i necessiten menys temps de desenvolupament. Durant *l'sprint* 5 i 6 es preveia realitzar un *testeig* per part del client amb una versió de prova del projecte; s'esperava doncs treball de reparació de funcionalitats existents i per això només es contemplaven *user stories* molt curtes com ara "Reiniciar el dispositiu de la cartellera remotament".

4.3 Objectius tècnics per a cada iteració

A partir de l'estimació vista al diagrama de Gantt de l'apartat anterior de les *user stories*, es va fer una aproximació sobre com hauria d'estar el projecte a nivell tècnic en acabar cada *sprint*. La programació era la següent:

Sprint 1: Una aplicació virtualitzada que pogués mostrar els vídeos al navegador.

Sprint 2: Tenir una segona aplicació virtualitzada que futurament servís per fer la gestió. Només havia de poder enviar fitxers entre l'ordinador amb el navegador que tenia la interfície i la Raspberry on s'executava el servidor de l'aplicació 2.

Sprint 3: La segona aplicació havia de poder fer un llistat del fitxers (amb crides a sistema), mostrar-lo i poder escollir quins eliminar. L'altra aplicació , havia de poder mostrar documents i imatges.

Sprint 4: Base de dades creada i sistema de la llista de reproducció. El reproductor havia de poder funcionar amb ella i l'usuari havia de tenir-hi accés mitjançant la interfície de la segona aplicació. El llistat de contingut havia d'estar a la base de dades amb una taula relacionada amb les entrades de la llista de reproducció.

Sprint 5: Control d'accés implementat i protecció dels *end points*.

Sprint 6: Aplicacions modificades segons les parts negatives del testeig realitzat durant *l'sprint*. Sistema d'apagada temporitzada i manual a distància implementats. Possibilitat de programar la visualització de fitxers en el temps per l'usuari també funcionals.

4.4 Pressupost del sistema de senyalització digital

Una altra part important de la planificació del projecte és el seu cost. En aquest apartat s'ha realitzat el següent pressupost:

Raspberry Pi 4

Concepte	**Preu**
Raspberry Pi 4 Modelo B - 8GB	83,99
Font d'alimentació oficial 15.3W USB-C per a Raspberry Pi 4	8,95
Caixa oficial per a Raspberry Pi 4	5,95
Cable oficial Micro-HDMI a HDMI (Tipus A) 1M	5,90
Tarjeta oficial micro-SD NOOBS 64 GB Classe 10	19,90
Total	124,69
21 % IVA	26,19
Total	150,88 €

Cablejat

Concepte	**Preu**
Cablejat elèctric i endoll	120,32 €

Xarxa

Concepte	**Preu**
Cablejat de xarxa i roseta	257,38 €

TV

Concepte	**Preu**
LG Ultra HD TV 4K, 108cm/43" + suport	554,18 €

Altres

Concepte	**Preu**
Pendrive SanDisk Ultra Fit USB 3,1 64 GB	12,46
Cable ethernet (1 metre) classe 6	2,55
Programador digital	19,90
Velcro	5,50
Total	40,41 €

Cost total de Hardware i materials:

Concepte	**Preu**
Cost total de *Hardware* i materials	1023,17 €

Software

Concepte	**Preu**
Preu per hora de desenvolupament	17,00 €
Total (153 hores estimades)	2601,00 €

Cost total de *hardware* i *software*:

Concepte	**Preu**
Cost total de *hardware* i *software*	3114,17 €

En aquest pressupost podem veure dues parts: El cost del *hardware* (i materials) i el cost aproximat del desenvolupament del *software*.
Aquest últim està inclòs per poder fer un càlcul teòric del cost total del projecte si aquest hagués sigut un treball remunerat. Com no ha sigut el cas, el cost real del projecte ha sigut només el de la primer part.

4.5 Adaptació de la planificació

Degut a la metodologia de treball, la planificació es va anar adaptant als diferents canvis i a les noves idees. La part de testeig es va enfocar més a *l'sprint* 6, originant un canvi en la planificació, allargant-la un *sprint* més, que finalitzà el 7 de gener. Entre aquests dos períodes, es van afegir més de 14 *Users Stories* a les plantejades inicialment. Totes elles molt concretes i en la majoria de casos sorgides per haver redissenyat la interfície de control a *l'sprint* 5.
A la reunió d'abans d'implementar la visualització de la llista de reproducció (per exemple) es va decidir afegir user stories com "Tenir paginat el llistat " o "Poder utilitzar un cercador en la llista de reproducció". Algunes d'aquestes però van resultar ser més prioritàries que d'altres, descartant per a la implementació algunes de les plantejades inicialment. La figura A3 dels annexos mostra la taula total de les *Users Stories* després d'haver finalitzat el projecte. Tot i així, hi ha molts canvis de codi que vaig fer i que no queden reflectits en aquest *backlog* degut a no ser percebuts directament per l'usuari.

5. Disseny

En aquest capítol, es tracten els temes referents al disseny del sistema en els diferents subapartats: disseny de l'arquitectura, de classes, del model de la base de dades i dels sistemes de seguretat.

5.1 Arquitectura de la solució

L'arquitectura del sistema es troba representada a la següent figura 5.1:

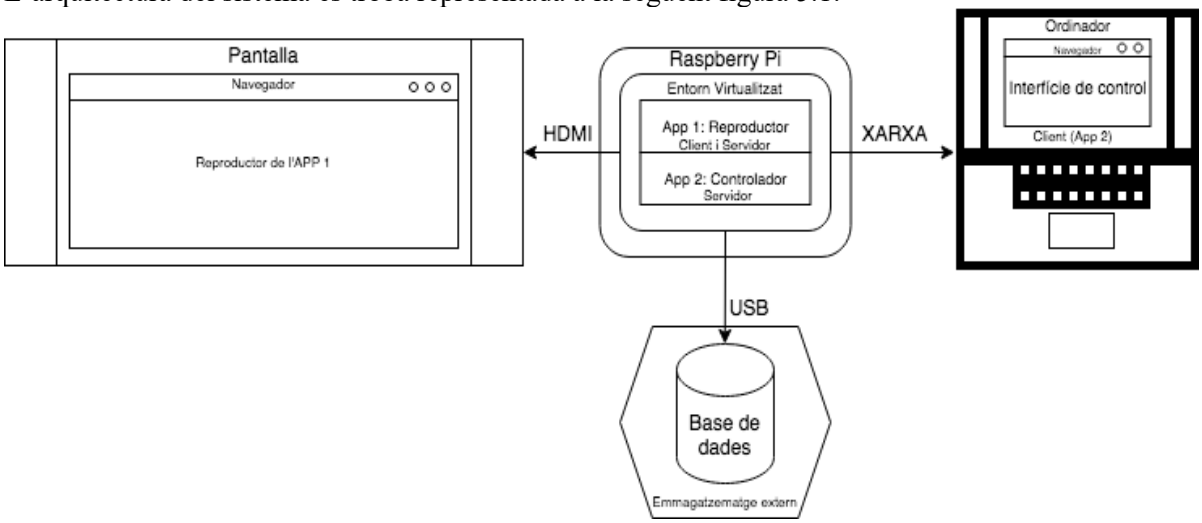


Figura 5.1: Diagrama de l'arquitectura (Font: Elaboració pròpia).

El model d'arquitectura és de tipus client-servidor: 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 aquest cas, un client s'executa al navegador de l'ordinador connectat a la xarxa del CRAI (el que es vulgui fer servir per a la gestió del sistema) i l'altre, que conté el reproductor, a la mateixa Raspberry.

El servidor del sistema s'executa a la Raspberry. Utilitza una arquitectura API RESTFUL; distribuint les tasques del sistema en dues aplicacions: el reproductor i el controlador.
Cadascuna funciona com un microservei; contenen un client i un servidor, es troben interconnectades, intercanvien informació entre elles però tenen competències exclusives sobre un grup de dades i funcions. Totes dues s'executen virtualitzades i tenen accés a una base de dades comuna.

5.2 Eines i tecnologies del projecte

El sistema de senyalització 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
- Vue.js (per a la interfície de control que veu l'usuari) i HTML + Javascript (per a el reproductor)
- Una base de dades relacional de tipus SQLite
- Scripts en Bash

Aquests, funcionen a la Raspberry sobre Raspbian OS (distribució de Debian que ve per defecte a la Raspberry). L'aplicació del reproductor es mostra sobre el navegador Chromium (que ve instal·lat per defecte al S.O)

5.3 Aplicació 1: El reproductor

5.3.1 Funcionament

El seu client és un reproductor (visible al navegador) i el seu servidor és qui s'encarrega de gestionar-lo. Aquesta aplicació té control exclusiu de la llista de reproducció i del sistema de reproducció en si mateix (modes, estat...). És a dir: per fer qualsevol operació amb la llista de reproducció (afegir, eliminar una entrada,...) l'usuari (des de el client de l'aplicació de control), fa una crida, per exemple, a l'*endpoint* de */playlistEntry* (n'hi ha més i estan explicats a l'annex d'*endpoints*), que forma part de *l'API* (del servidor) de l'aplicació del reproductor i aquest últim la processa.

5.3.2 *Back end*

El diagrama de classes del *back end* correspon a la figura A4 dels annexos. Degut a la seva naturalesa de microserveis, tant l'aplicació de reproducció com la de control, comparteixen classes. Els mètodes no es troben representats a la figura del diagrama ja que són operacions HTTP, però al següent subapartat sí que se'n parla. L'explicació del diagrama, classe per classe, es la següent:
Classe *app*: Aquesta classe és el servidor en si. Delega les crides d'*end points* mitjançant les APIs que es troben a *resources* i també quan és el cas, retorna la vista (el reproductor) cap al client. Aquesta classe conté també la configuració de cada servidor respectivament (ports, etc...).

Nivell de clases '*Resources*': Un cop l'aplicació agafa l'*end point*, deriva la gestió de la seva crida en les classes d'aquest nivell. Aquestes classes, **no modifiquen directament les dades** sinó que deriven la consulta o modificació a la classe del model corresponent. Formen part d'aquest nivell:

-*PlaylistEntry*: Permet consultar o modificar per entrades la llista de reproducció.
-*Playlist*: Permet consultar la llista de reproducció sencera.
-*NextEntry*: Permet consultar o modificar el següent element a reproduir-se. Fa ús de la llibreria pròpia *nextByMode.*
-*Status*: Permet consultar o modificar l'estat de l'aplicació del reproductor.
-*Mode*: Permet consultar o modificar el mode de reproducció.
-*IntercaledEntry*: Permet consultar o modificar l'entrada intercalada de la llista de reproducció.

Llibreries pròpies amb mètodes de suport:

-*nextByMode.py*: Conté els algoritmes de cada mode de reproducció. Quan l'aplicació 1 demana el següent element a reproduir des de *NextEntry.py*, segons el mode seleccionat, aquest retorna la informació del següent element a reproduir.

Playlist(Model), Content(Model), Account(Model): Aquestes classes pertanyen al nivell de "Model" de l'aplicació. Interaccionen amb la base de dades mitjançant *sqlalchemy* per gestionar el seu contingut. Contenen el model de les taules i els mètodes per realitzar operacions (de cada taula, respectivament). Aquests models, es troben detallats al següent punt.

En quant a patrons, l' aplicació, està realitzada amb el patró "model vista controlador" aplicat entre el *front end* i el *back end*.
El reproductor només visualitza la part de l'aplicació i "interactua" amb l'usuari mostrant-li el contingut però la lògica de la reproducció es troba a la part de *back end,* al servidor.
El nivell de la classe *app* conjunt amb el *resources* es tractaria doncs de controlador (segons el patró de disseny). Això és degut a que la classe *app* només gestiona les crides i les classes que trobem a *resources,* són el desacoblament d'aquestes funcions.
Finalment trobaríem el nivell de model que, com bé indica el seu nom, apliquen la part de "model" del patró de disseny. Aquestes classes són en essència les que tenen l'esquema de les dades. De fet, són les taules "traduïdes" per poder treballar en Python amb la base de dades SQL.

5.3.3 Mètodes *RESTFUL*

Els mètodes de les classes *Resource* de l'apartat anterior es troben amb una breu descripció a la següent figura 5.2:

Mètode	***End point***	**Descripció**
GET	/mode	Permet saber el mode de reproducció seleccionat
PUT	/mode	Permet canviar el mode de reproducció seleccionat
POST	/playlistEntry	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
GET	/nextEntry	Permet saber la informació del següent element a reproduir
POST	/nextEntry	Permet saber el mode de reproducció seleccionat
GET	/status	Permet saber l'estat del reproductor
PUT	/status	Permet canviar l'estat del reproductor
GET	/intercalatedEntry	Permet saber quina és l'entrada intercalada
POST	/intercalatedEntry	Permet afegir una entrada per ser utilitzada com a intercalada
DELETE	/intercalatedEntry	Permet eliminar l'entrada intercalada

Figura 5.2: Taula de l'API de l'aplicació del reproductor (Font: Elaboració pròpia)

Aquests mètodes pertanyen a l'API d'aquesta aplicació, però també són accessibles des de l'altra aplicació del sistema.

5.4 Aplicació 2: Control

5.4.1 Funcionament

El seu client és la interfície de control que l'usuari veu al seu navegador i el seu servidor és qui s'encarrega de gestionar-lo. El servidor d'aquesta aplicació també té el control dels fitxers multimèdia i de la gestió dels usuaris, de manera que per fer qualsevol operació amb la llista de fitxers o el sistema d'usuaris, l'usuari fa una crida des de el client (el panell de control) a l'*endpoint* de */account* o */content*, que es troba al seu propi servidor.

5.4.2 *Front end*

A diferència de l'aplicació del reproductor, aquesta té un client més sofisticat, ja que ha de proporcionar la interfície de control per a l'usuari. Tota la part de *front end* forma part d'una mateixa pàgina de cara a l'usuari. Internament, no està dividit en classes ja que el *framework* no ho permet. "L'equivalent" en aquest cas són els *components*. Per això he elaborat aquest esquema de la Figura 5.3, per veure la relació.

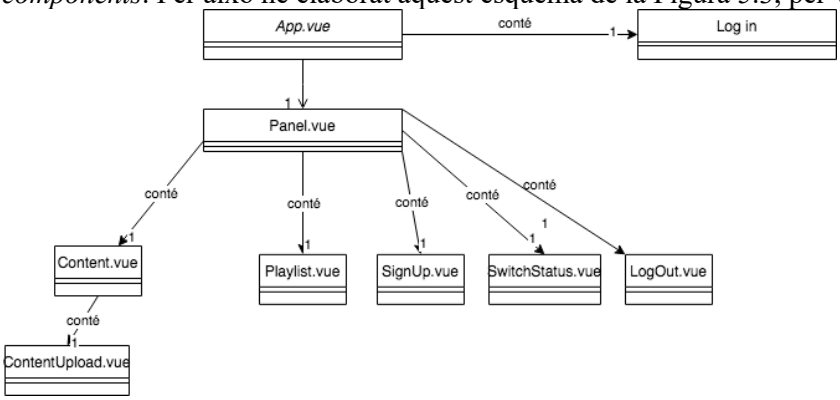


Figura 5.3: Diagrama de *components* del *front end* de l'aplicació de control (Font: Elaboració pròpia)

Com es pot veure, la base de tot és l'*App.vue*. Aquesta, a banda de contenir el peu de pàgina i el color o imatge de fons, conté un altre component: el panell de *log in* i el panell de control. Aquests dos components no es visualitzen mai alhora ja que sense iniciar sessió no es pot accedir a veure el panell de control.

El panell de control, està format per una agrupació de pestanyes. Cada pestanya conté una part diferent de la interfície, organitzada a la seva vegada en *components*. Aquest disseny aporta modularitat, ja que per tal d'afegir

nous “controls”, només s’hauria de crear un nou component i enllaçar-lo amb el panell, sense afectar a la resta de codi.
La part (que gestiona el *front end*) d’enviar arxius, és un component anomenat *ContentUpload.vue* que es troba dins del *component Content*. Aquesta distinció és necessària ja que en cas de futures iteracions, els arxius podrien estar al núvol i no fer necessari enviar cap contingut: es podria prescindir d’aquest component sense modificar el *component* de Content.
La vista de *front end* del control de la reproducció (canviar els modes, canviar els arxius...) es troba al component de *Playlist.vue*. 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*. I finalment per tancar la sessió, la vista està implementada al component *LogOut.vue*.
Tots aquests components estan formats per *divs*, *scripts* i altres elements, al igual que el *front end* del reproductor de l’altra aplicació (aquest darrer no té cap *framework*, és tot HTML+CSS+Javascript). Degut a la naturalesa d’aquestes tecnologies, el codi no està organitzat en classes sinó que es tracta d’elements dins d’un arxiu HTML, no explicats aquí per no pertànyer a la part estricta de disseny.

5.4.3 Interfície d’usuari

Després de veure els diferents apartats de disseny relacionats amb parts invisibles per a l’usuari, queda veure doncs, el disseny de la part que directament veu l’usuari al client d’aquesta aplicació. La maqueta que mostraré a continuació va ser el resultat d’adaptar les *User Stories* inicials a una interfície més tangible. Per exemple, quan el client parlava de la llista, ho feia com un concepte; al moment de planificar la vista, va afegir requeriments nous com ara poder veure miniatures de cada element. També es tracta del resultat d’incloure idees noves sorgides en els *sprints* previs a la realització d’aquesta part, i a un extens diàleg per tal d’assegurar la satisfacció del client.
En tot moment i d’acord amb el tractat, he tingut en compte més criteris d’usabilitat que d’experiència d’usuari ja que el projecte té la finalitat gestionar el sistema de la cartellera i no pas generar una experiència que generi un atractiu sobre la competència per tal d’escollir-lo, ja que no hi ha finalitat comercial darrere. Una resum de la justificació d’aquests criteris:

- És fàcil d’aprendre; no cal tenir coneixements previs, ni llegir cap manual per tal d’entendre el seu funcionament. Els botons es troben etiquetats amb claredat i combinats amb colors que indiquen de quin tipus d’operació es tracta, basats en dos tipus: destructius (botons de color vermell), per indicar a l’usuari que aquella acció comportarà eliminació (per reforçar aquest missatge) i constructius (botons amb la combinació de colors de la interfície, blanc i blau) per reforçar la idea de les etiquetes “afegir elements”, “afegir usuari”, etc...

- És fàcil de recordar i d’entendre; l’usuari pot estar molt de temps sense fer-la servir que gràcies a la descripció senzilla de cada funció condensada en el nom de les etiquetes recordarà com utilitzar-la.

- És eficient en realitzar les tasques; totes les accions es duen a terme en el mínim de clics possibles.

- Algunes d’aquestes s’han minimitzat al màxim amb la selecció múltiple. No cal per exemple seleccionar un a un i esborrar tot el contingut; sinó que tenim un botó a la part superior del llistat de fitxers, que permet seleccionar tots els elements alhora per tal de deixar operacions com “eliminar els elements” o “afegir a la cua de reproducció” a un clic.

El *mockup* o esbós elaborat amb la web Balsamiq el podem veure a la següents figures:

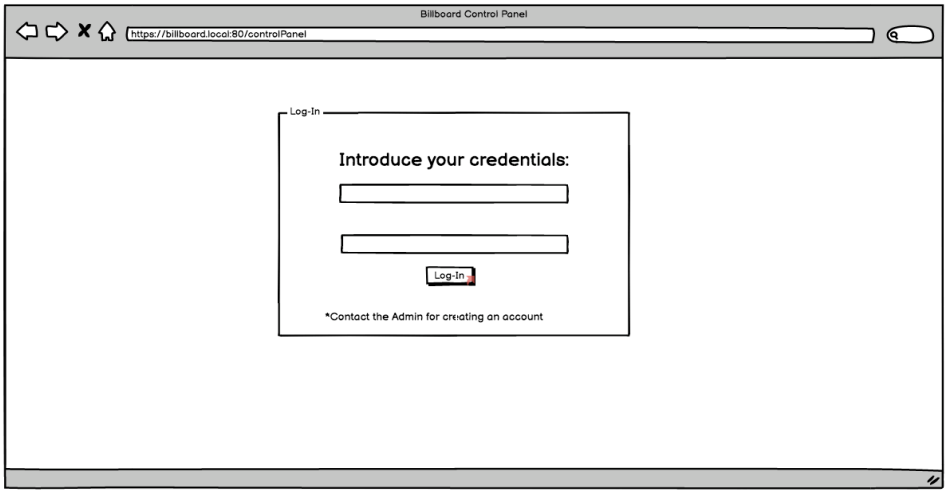


Figura 5.4: Esbós inici de sessió (Font: Elaboració pròpia)

El primer que veu l'usuari quan vol accedir al sistema de control és la finestra de la figura 5.4, per tal de limitar l'accés només al personal autoritzat. Com es pot veure no permet la creació d'usuaris ja que no tindria sentit al ser un servei privat. Un cop s'inicia sessió, l'usuari podrà veure el panell de control, que en aquest document es troba dividit en les figures següents:

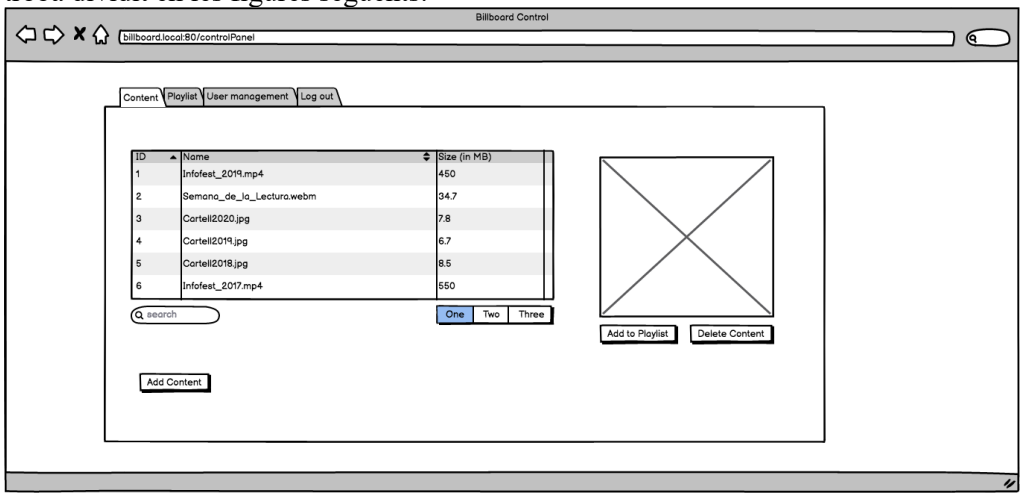


Figura 5.5: Esbós pestanya gestió de contingut (Font: Elaboració pròpia)

Com podem veure a la figura 5.5, la interfície d'usuari està dissenyada en una única pantalla i distribuïda en diferents pestanyes del mateix panell principal. D'aquesta manera es dona la sensació que tots els aspectes de control del sistema són un conjunt, alhora que en facilita molt la navegabilitat. La pestanya de gestió dels fitxers multimèdia mostra el llistat de fitxers de la taula de la base de dades, que es troben físicament a la carpeta designada de la Raspberry. Per facilitar la visualització, tal com es va acordar a les reunions prèvies al disseny d'aquesta UI, vaig afegir un cercador de fitxers, un *scroll* de la taula i opcions de paginació per limitar aquest desplaçament vertical. També tenim diferents botons que realitzen les funcions, com ara afegir a la cua de reproducció, esborrar del sistema o afegir més fitxers.

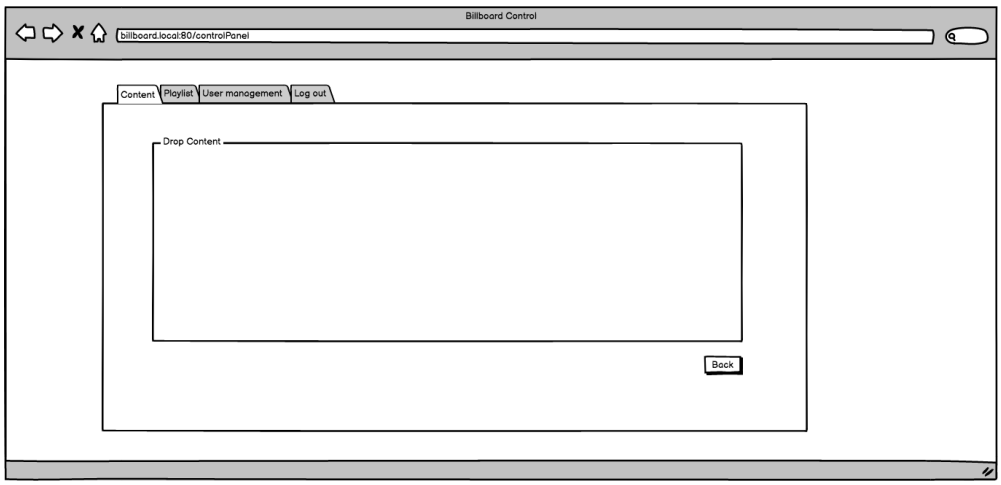


Figura 5.6: Esbós de la part afegir contingut dins de la pestanya de contingut(Font: Elaboració pròpia)

A la figura 5.6 es pot veure el panell que permet enviar fitxers des de l'ordinador amb què accedim a la interfície de la Raspberry. Per facilitar el procés, s'hi poden arrossegar múltiples fitxers.

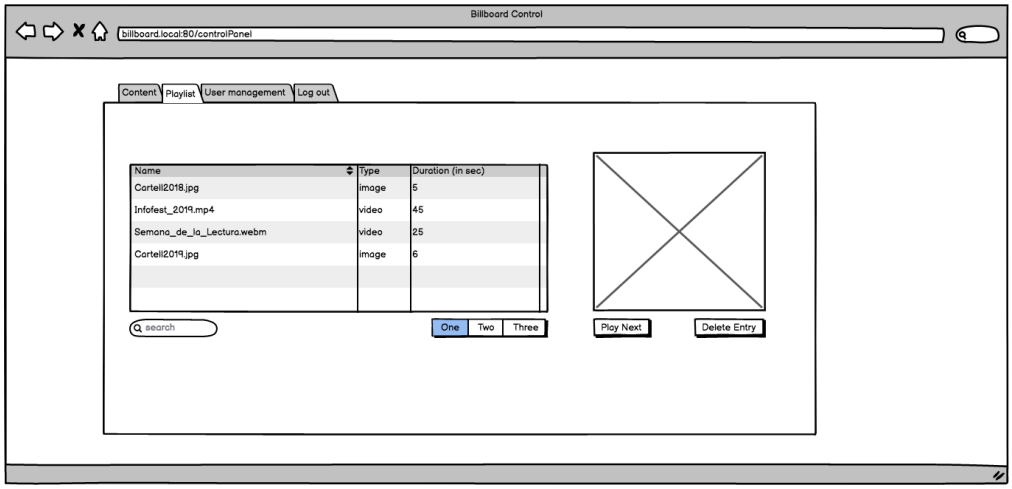


Figura 5.7: Esbós de la pestanya gestió de de la cua de reproducció (Font: Elaboració pròpia)

El disseny de la pestanya per visualitzar la cua de reproducció, que es pot veure a la figura 5.7, mostra la similitud amb la distribució dels elements de la interfície de gestió dels fitxers. Amb això, es facilita l'aprenentatge, ja que l'usuari només haurà d'acostumar-se a un únic disseny, ja de per si enfocat a la senzillesa.

Amb el mateix concepte en ment, vaig dissenyar la part de gestió dels comptes d'usuari, que es pot veure a la figura 5.8. Conté una taula per poder visualitzar els noms i el rol, un botó per afegir nous usuaris i un altre per esborrar els seleccionats.

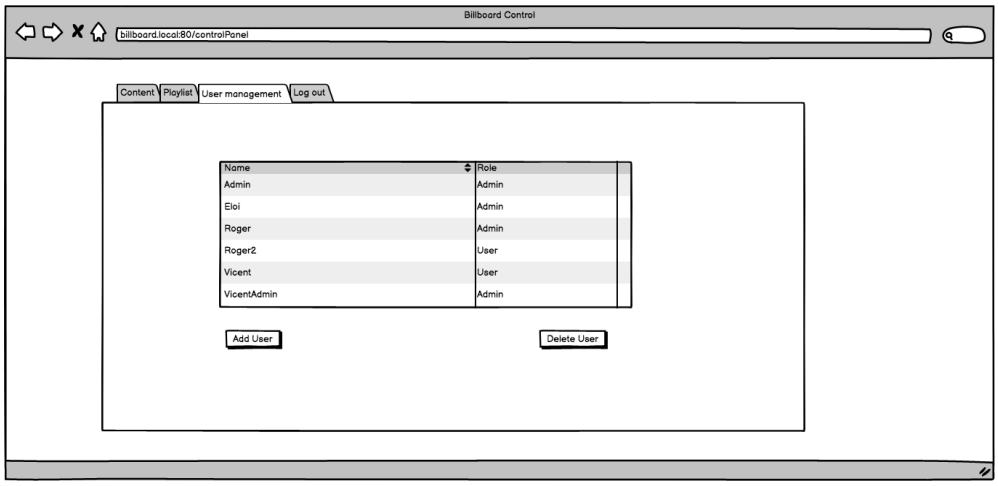


Figura 5.8: Esbós pestanya gestió d'usuaris (Font: Elaboració pròpia)

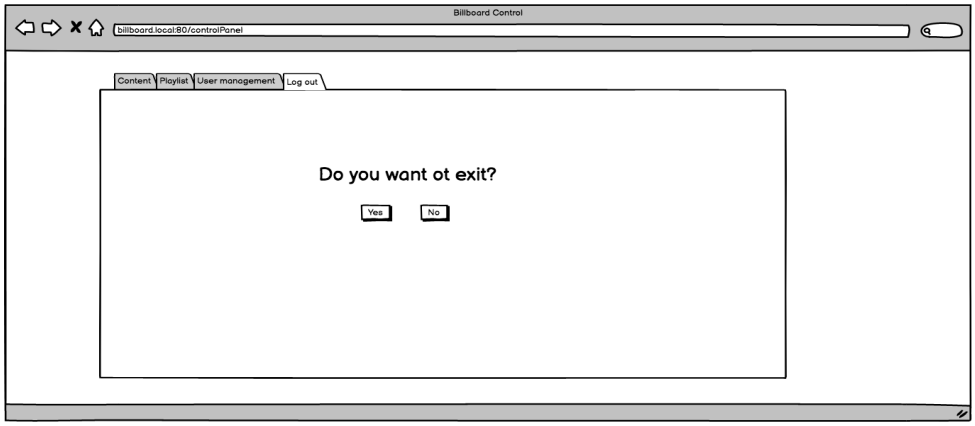


Figura 5.9: Esbós pestanya per tancar la sessió (Font: Elaboració pròpia)
Finalment, ens trobem amb la part de la vista de tancar la sessió a la Figura 5.9 que ens retornarà (en cas de confirmar) al punt inicial: l'inici de sessió mostrat a la Figura 5.4

5.3.4 Back end

El diagrama de classes del *back end* correspon a la figura A5 dels annexos. Degut a la seva naturalesa de microserveis, tant l'aplicació de reproducció com la de control, comparteixen classes. Els mètodes no es troben representats a la figura del diagrama ja que són operacions HTTP, però al següent subapartat sí que se'n parla.
Classe *app*: Aquesta classe és el servidor en si. Delega les crides d'*end points* mitjançant les APIs que es troben a *resources* i també quan és el cas, retorna la vista cap al client. Aquesta, és el *front end* de la interfície de control tractats al subapartats anteriors.
Aquestes classes contenen també la configuració de cada servidor respectivament (ports, etc...).

Nivell de clases '*Resources*': Un cop l'aplicació agafa l'*end point*, deriva la gestió de la seva crida en les classes d'aquest nivell. Aquestes classes, **no modifiquen directament les dades** sinó que deriven la consulta o modificació a la classe del model corresponent. Formen part d'aquest nivell:

-*Content*: Permet consultar o modificar la informació dels fitxers multimèdia.
-*Account*: Permet consultar o modificar la informació relacionada amb els comptes d'usuari.
-*Login*: Permet iniciar sessió.

Llibreries pròpies amb mètodes de suport:

-*thumbnailGenerator*: utilitza *VideoFileClip* de Python per tal de generar les miniatures dels fitxers multimèdia que s'afegeixen.

Content(Model), Account(Model): Aquestes classes pertanyen al nivell de "Model" de l'aplicació. Interaccionen amb la base de dades mitjançant *sqlalchemy* per gestionar el seu contingut. Contenen el model de les taules i els mètodes per realitzar operacions (de cada taula, respectivament). Aquests models es troben detallats al següent apartat.

En quant a patrons, l'aplicació, està realitzada amb el patró "model vista controlador" aplicat entre el *front end* i el *back end*.
La part amb què interactua l'usuari doncs, feta amb *Vue js*, conté només la lògica dels botons i la traducció d'aquestes funcions desitjades en peticions al servidor. El nivell de la classe *app* conjunt amb el *resources* es tractaria doncs de controlador (segons el patró de disseny). Això és degut a que la classe *app* només gestiona les crides i les classes que trobem a *resources* són el desacoblament d'aquestes funcions. En cap moment interactua amb l'usuari directament.
Finalment trobaríem el nivell de model que com bé indica el seu nom, aplica la part de "model" del patró de disseny. Aquestes classes són en essència les que tenen l'esquema de les dades. De fet, són les taules "traduïdes" per poder treballar en Python amb la base de dades SQL.

5.4.5 Mètodes RESTFUL

Els mètodes de les classes *Resource* de l'apartat anterior, es troben amb una breu descripció a la següent figura 5.2:

Mètode	***End point***	**Descripció**
GET	/account/<string:username>	Permet saber la informació d'un compte d'usuari
POST	/account	Permet afegir un nou compte d'usuari
PUT	/account/<string:username>	Permet modificar la informació d'un compte d'usuari existent
DELETE	/account/<string:username>	Permet eliminar un compte d'usuari existent
GET	/accounts	Permet veure un llistat amb tots els comptes d'usuari
POST	/login	Permet iniciar sessió
GET	/content/<string:filename>	Permet saber les metadades d'un fitxer
POST	/content	Permet la recepció d'un fitxer per afegir-lo
PUT	/content/<string:filename>	Permet modificar la informació d'un fitxer existent
DELETE	/content/<string:filename>	Permet eliminar les metadades i el fitxer físicament
GET	/contents	Permet veure un llistat amb totes les metadades dels fitxers

Figura 5.10: Taula de l'API de l'aplicació Control (Font: Elaboració pròpia)

5.5 Model relacional

A diferencia del disseny de la part de client-servidor o el de classes, el disseny de la base de dades és molt simple. Per elaborar el model de dades només vaig fer ús de 3 taules: *Accounts, Content* i *Playlist*.

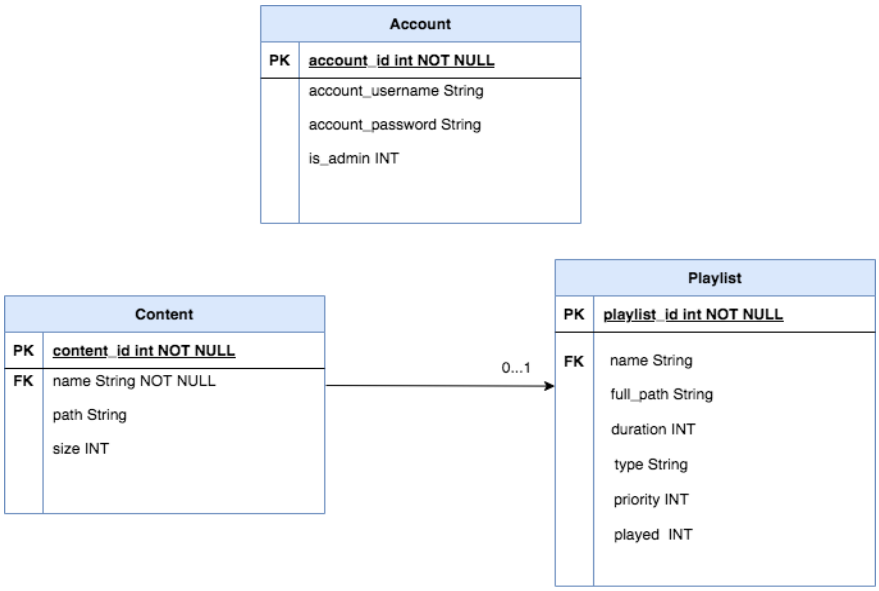


Figura 5.10: Model entitat relació (Font: Elaboració pròpia)

A la figura 5.10 podem veure que l'única relació es troba entre *Content* i *Playlist:* el contingut pot tenir (o no) només una entrada a la llista, però una entrada a la llista sempre tindrà un contingut (un fitxer) associat, ja que no pot existir una entrada a la llista buida. L'únic cas doncs seria en una futura implementació amb arxius al núvol, on no caldria aquesta relació, però amb el sistema actual és necessari.
La clau que els relaciona (clau forana) doncs és el nom del fitxer que serà el mateix nom que tindrà l'entrada a la llista. Aquesta decisió la vaig basar en l'observació de diferents programes de reproducció, on com és d'esperar, els noms dels elements de les llistes de reproducció tenen el propi nom del fitxer. Podria ser diferent i deixar que l'usuari posés el nom que volgués com a entrada (amb finalitat descriptiva), per exemple: Recopilatori.Videos.Matefest.2019.ver.3840x2160.mp4 podria ser guardat a la llista com: "Matefest 2019 4k" però el client no va considerar aquesta possibilitat.
Taula per taula, trobem doncs:

-*Account*: On es guarda la informació de registre dels usuaris. Els seus camps són: l'identificador d'usuari guardat com un enter, el propi nom d'usuari i la seva contrasenya, guardats com a cadenes de caràcters respectivament, i un camp anomenat *is_admin* en format d'enter que indica els privilegis (0 si es usuari normal, 1 si és administrador).

-*Content*: Taula on es guarda la informació bàsica dels fitxers. Permet disposar d'un llistat simple per facilitar operacions com ara afegir-los a la llista per reproduir o obtenir les seves metadades i no haver de fer crides al sistema per obtenir aquesta informació. Els seus camps són: l'identificador de contingut, guardat com un enter, el propi nom de fitxer i la ruta absoluta del fitxer, en format de cadenes de caràcters (respectivament) i la mida (d'ordre de *Mega Bytes*) en format d'enter.

Playlist: Taula que funciona com a cua de reproducció. L'algorisme de reproducció agafa les metadades de les entrades a la llista d'aquesta taula. A diferència de les altres, els camps són una mica més complexos i alguns necessiten una petita explicació:

-L'identificador d'entrada a la llista, guardat com a enter.

-El nom de fitxer, guardat com a cadenes de caràcters.

-La ruta del fitxer: aquest camp conté la ruta en virtual (en format de cadena de caràcters), per tal que el reconegui el reproductor. És necessari degut a que no coincideix amb el camp de ruta de la taula de contingut i per tant, per poder eliminar o modificar els fitxers en si, aquesta no es pot utilitzar.

-*Duration*: camp de duració, en segons, per fitxers estàtics. L'usuari definirà aquest valor al moment d'afegir una entrada a la llista. Els segons definits seran el temps que es mostraran els fitxers estàtics. En cas de deixar-se buit, el sistema afegirà el valor per defecte de 6 segons al camp de la taula. Es guarda com un enter.

- *Type*: No es tracta de l'extensió del fitxer sinó de si és un vídeo o imatge. Abans d'afegir l'entrada a la taula, el sistema escull l'opció més adient de manera automàtica. El valor es guarda en format de cadenes de caràcters.

- *Priority*: Camp on s'indica la prioritat de reproducció, en valor d'enter. El sistema l'utilitza per tal de saber quin fitxer és prioritari i determinar l'ordre de la reproducció (en els modes seqüencial i intercalat), assignant-lo automàticament a cada entrada. La intenció també era poder utilitzar-lo amb futures ampliacions on hi hagi valors propis de l'usuari, per tal de tenir un mode de reproducció aleatori amb prioritats. El valor és un enter.

-*Played*: S'utilitza per saber quins fitxers s'han reproduït. El valor es guarda en format d'enter (0: En Cua, 1: reproduït).

En tot moment he prioritzat l'eficiència i velocitat de les operacions de la cartellera, sobretot en temps d'agafar les dades per a la reproducció. Per això hi ha, per exemple, un camp per tal d'indicar directament al client de l'aplicació 1 (el reproductor), si ha de mostrar el visualitzador d'imatges o el reproductor de vídeos, sense que necessiti d'un algoritme de detecció del tipus de fitxer, ja que la transició entre fitxers a reproduir ha de ser el més ràpida possible. Per fer lleugeres també les *queries* a la base de dades, vaig simplificar el número de taules: podria haver-hi una taula per a cada rol d'usuari o diferenciar el contingut, però tenint en compte les limitades prestacions del dispositiu sobre el que s'executen les aplicacions amb virtualització i el seu ús constant de la base de dades, vaig considerar que l'havia de reduir al màxim.

5.6 Aspectes de seguretat

Aquest apartat va ser molt important en el desenvolupament del projecte. De fet, un dels requeriments inicials del projecte va ser que s'havia de tenir molt en compte la seguretat, ja que la cartellera com a actiu és valuosa. No monetàriament sinó com a eina de difusió: algú podria comprometre la seguretat i canviar el contingut per un altre no desitjat.
Per aconseguir mitigar aquest tipus d'amenaces s'ha optat per la utilització de tècniques com ara la protecció *d'endpoints* mitjançant privilegis d'usuari. El nivell d'accés als *endpoints* de cada usuari està detallat en el llistat *d'endpoints* als annexes.
Per poder accedir a l'aplicació web del controlador (o a les seves dades mitjançant *endpoints*) es requereixen unes credencials d'usuari prèviament creades per un usuari amb rol d'administrador. No és possible registrar-se com a qualsevol servei d'internet conegut (*Google*, *Facebook*, *Outlook*...) ja que un atacant podria crear-se un usuari i per tant totes les altres mesures no serviria de res.
Hi trobem doncs els usuaris normals i administradors com a tipus d'usuaris. Algunes de les accions que poden fer els de primer tipus són:
-Afegir contingut multimèdia des de un navegador a la memòria de la cartellera.
-Afegir i esborrar el contingut desitjat a la llista de reproducció que es troba dins de la cartellera.
-Modificar el comportament de la reproducció.
En canvi, els administradors, a banda de poder realitzar tot l'anterior, poden gestionar tots els usuaris:
-Crear usuaris nous
-Modificar els noms, contrasenyes o el rol de cada usuari
-Esborrar-los

El sistema té un disseny d'una única llista de reproducció a la qual podran accedir tots els usuaris. Això té sentit ja que s'utilitzen les credencials per restringir l'accés, però el contingut és únic, ja que pertany a la cartellera en si, no pas a cap usuari.

A banda de la protecció de les aplicacions, per protegir l'accés remot a la *Raspberry*, s'ha canviat el port d'accés *ssh*. Per defecte, era el mateix que en altres sistemes operatius de la família *Unix* (el port 22) però ara s'utilitza el port 1998. L'elecció d'aquest port ve donada pel fet de que cap altra tasca o programa al sistema l'utilitza.
També s'ha canviat la contrasenya que venia configurada amb la instal·lació (*raspberrypi*) per una altra amb els següents criteris de seguretat:

- Llarga: més de 8 caràcters
- Inclou minúscules, majúscules i números
- No utilitza la substitució de números per lletres.
  (Com ara *dorbell*->d00rb311)
- No conté seqüencies del teclat (Per exemple 789 o zxcv)

A banda d'això, també s'ha tingut en compte que la contrasenya havia d'estar protegida a atacs més sofisticats que la força bruta normal (protegida amb els criteris anteriors), com ara l'ús de diccionaris. És un tipus d'atac molt comú, en què l'atacant introdueix un conjunt de possibles paraules relacionades amb un tema i mitjançant un *software*, genera possibles combinacions fins a trobar el resultat. En aquest cas, de no haver-ho tingut en compte, seria senzill poder obtenir-la. L'atacant, com és lògic, introduiria al diccionari contrasenyes com ara: biblioteca, ub, crai, 2020, 2021, cartellera… per tal d'obtenir una contrasenya del tipus bibliotecaUb2021, CraiUniveritat2020 o més sofisticades que aquest exemple. A la mateixa distribució *Kali Linux* (molt coneguda a Internet i de fàcil accés) trobem eines com *Hydra* que són molt fàcils d'utilitzar inclús per a gent amb nivell baix de coneixements d'informàtica i que per tant serien un greu problema.
Per tant, la contrasenya va ser generada per un *software* de manera aleatòria. En aquest cas, es va utilitzar el servei web: https://passwordsgenerator.net/

6. Implementació

En aquest capítol es troben detallats els diferents algoritmes que he desenvolupat per tal d'aconseguir el funcionament de les característiques del sistema.
Un aspecte important per tal d'entendre'ls, és l'ús de l'element multimèdia default.mp4.
Aquest vídeo es troba físicament a la carpeta del contingut i com el seu nom indica, és el fitxer per defecte. És tracta d'un recurs que serà utilitzat pels algoritmes en casos on no hi ha contingut disponible (com ara quan la llista de reproducció està buida o quan s'ha seleccionat un mode intercalat i no s'ha definit prèviament un fitxer multimèdia per dur-lo a terme).

6.1 Algorisme principal del reproductor

Algorisme 1 : algorisme principal

```
Funció: motor_de_reproducció()
Entrada :  buit
Sortida   : buit
Mentre noAturar :
        haAcabat = espera_a_acabar()
        si haAcabat:
                resposta = preguntar_al_servidor()
                si resposta == Ok:
                        següent = preguntar_al_servidor()
                        si següent == vídeo:
                                reproductor = vídeo
                                vídeoActual = següent
                        si següent == estàtic:
                                reproductor = estàtic
                                contingutActual = següent
                sinó
                        vídeoActual = videoPerDefecte

        haAcabat = Fals
```

Figura 6.1: Algorisme principal del reproductor representat amb pseudocodi (Font: Elaboració pròpia)

En rebre al navegador l'HTML amb el reproductor, comença el procés que podem veure a la figura 6.1: reprodueix automàticament el fitxer *per defecte* del sistema. En acabar, crida als mètodes de *javascript* propis per demanar la informació al servidor de l'estat de reproducció. Si aquest retorna un *online*, es continua amb el flux de l'algorisme. Sinó, s'inicia la reproducció del fitxer per defecte un altre cop.
En el primer cas, es demana la informació del següent element mitjançant les crides al servidor de l'aplicació 1. Aquest darrer comprovarà que l'usuari no hagi seleccionat cap fitxer multimèdia per reproduir seguidament. Això es degut a que aquest element té prioritat màxima i independentment del mode, l'usuari el vol reproduir immediatament després del que estava veient per pantalla, inclús si és abans d'obrir el propi reproductor. És a dir: si l'usuari primer ha obert la interfície de control, ha seleccionat el fitxer per reproduir-lo a continuació i després ha obert el reproductor, quan aquest acabi de reproduir el fitxer per defecte, el reproductor començarà el seleccionat prèviament.
Un cop descartat aquest cas (no hi ha cap fitxer amb aquesta prioritat) en funció del mode de reproducció seleccionat, escollirà l'element a reproduir. Si des de la interfície l'usuari no ha canviat prèviament el mode, per defecte, reproduirà els elements de la llista de forma seqüencial. Els algoritmes de reproducció (segons el mode) es troben detallats als apartats següents. Un cop el servidor ha retornat la informació, el reproductor mostra aquest arxiu i el visualitza fins que acabi (en cas de ser estàtic, el mostra durant el temps seleccionat) i comença l'algorisme de nou.

6.2 Algorisme de reproducció seqüencial

Algorisme 2 : algorisme de reproducció seqüencial

```
Funció: reproducció_seqüencial(llista):
Entrada :  (Objecte de classe) llista
Sortida   : element de la classe llista
si no esPrimerElement:
        auxiliar = llista.primerElement
        llista.esborra_primer_element()
        llista.afegir_element(auxiliar)
        llista.actualitza_les_posicions()

sinó:
        retorna llista.primerElement
```

Figura 6.2: Algorisme de reproducció seqüencial representat amb pseudocodi (Font: Elaboració pròpia)

Tal com es pot veure a la figura 6.2, l'algorisme fa una distinció de dos casos al començament. L'explicació d'aquesta distinció de casos és la següent: si és just el primer fitxer (després de mostrar el vídeo per defecte), aquest encara s'ha de reproduir abans de posar-lo al final de la cua. En canvi, si ja s'ha reproduït el primer element de la llista, es vol veure el següent. El que era el primer abans, ara ha d'anar al final per tal de seguir l'ordre d'una cua.
Un cop s'ha fet la comprovació, si estem en aquest darrer cas, el següent pas és guardar l'entrada en una variable, ja que s'esborra la seva posició a la llista per tal d'afegir-la al final de tot. Cada cop que es fa això, la base de dades incrementa en una unitat la posició de l'entrada, ja que detecta que es tracta d'un element nou a la taula. Per no acabar tenint un número de posició elevat i superior al número d'elements, s'aplica la disminució a tota la columna de posició a la taula de la llista. Finalment, es compleixi o no la condició inicial, es retorna el primer element de la llista com a resposta de la petició de següent element a reproduir per part del reproductor.
A la figura 6.3, es pot veure un petit exemple:

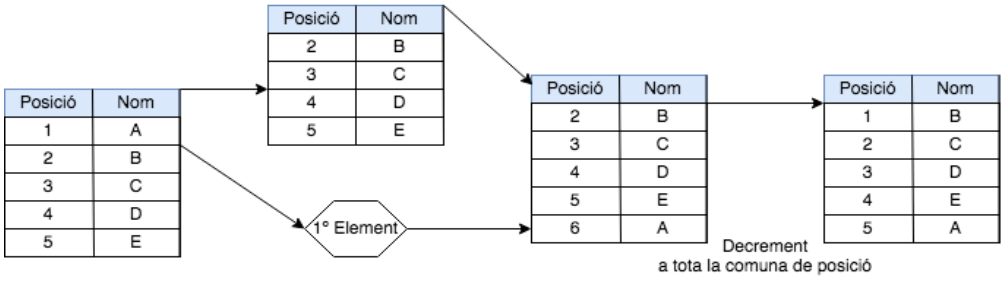


Figura 6.3: Exemple de l'algorisme de reproducció seqüencial (Font: Elaboració pròpia)

6.3 Algorisme de reproducció intercalada seqüencial

Algorisme 3 : algorisme de reproducció intercalada

```
Funció: reproducció_intercalada (llista):
Entrada :  (Objecte de classe) llista, booleà torn
Sortida   : element de la classe llista
si torn == seqüencial:
        torn = intercalat
        retorna  reproducció_seqüencial(llista)

sinó:
        torn = seqüencial
        retorna servidor.fitxerIntercalat
```

Figura 6.4: Algorisme de reproducció intercalada representat amb pseudocodi (Font: Elaboració pròpia)
L'algorisme de la figura 6.4 realitza els següents passos:

Primer de tot, comprova el torn que toca: si ha de reproduir el fitxer intercalat o el següent de la llista. Sigui el torn que sigui, després de fer la comprovació, es permuta el valor de la variable que indica el torn (de cara al següent) pel seu valor oposat, ja que es tracta d'un tipus booleà.

Si toca el torn de reproducció "normal", s'executa l'algorisme vist prèviament a l'apartat 6.2 d'aquest capítol, per tal de seguir l'ordre seqüencial dels fitxers.

Si toca l'altre torn, retorna la informació del fitxer prèviament seleccionat com a intercalat (el que es vol reproduir la meitat de les vegades) al reproductor, per tal de visualitzar-lo.

6.4 Algorisme de reproducció aleatòria sense repetició (permutació)

Algorisme 4 : algorisme de reproducció aleatòria

```
Funció: reproducció_aleatoria (llista):
Entrada :  (Objecte de classe) llista, booleà torn
Sortida   : element de la classe llista
elements_no_reproduits = llista.elements_no_reproduits()
si elements_no_reproduits.mida == 0:
        llista.marca_tots_com_no_reproduits()
        retorna servidor.fitxerPerDefecte
sinó:
        elementAuxiliar = agafa_un_element_aleatori (elements_no_reproduits)
        auxiliarId = elementAuxiliar.id
        llista.busca_per_id(auxiliarId).canvia_id(9999)
        primerId = llista.primerElement.id
        llista.busca_per_id(9999).canvia_id(primerId)
        llista.busca_per_id(9999).canvia_id(auxiliarId)
        llista.marca_com_a_reproduit(llista.primerElement)
        retorna llista.primerElement
```

Figura 6.5: Algorisme de reproducció aleatòria representat amb pseudocodi (Font: Elaboració pròpia)

Aquest algoritme (visible a la figura 6.5) funciona de manera molt diferent als anteriors. Primer de tot carrega de la base de dades totes les entrades a llista que compleixen la condició de no haver estat reproduïts abans. Si tots han estat reproduïts, el pas anterior haurà retornat una estructura de dades buida. En aquest cas, es canvien tots els valors de la columna *Played* pel de "no reproduïts" de cara a la pròxima petició del següent element del reproductor.
Per tal d'indicar que s'han reproduït tots i que comença una nova "iteració" de la llista, es retorna la informació del fitxer per defecte. Tot i així, aquest pas no sempre ha de ser necessari, d'aquesta manera es pot saber cada cop que comença i acaba l'algorisme.
Si encara queden fitxers per reproduir, agafa una entrada aleatòria d'aquest conjunt, es guarda el seu identificador d'entrada en una variable i es modifica l'identificador de l'entrada que té a la taula per un molt improbable de ser-hi. Podria haver-hi diferents alternatives a aquest pas, com ara buscar l'*id* últim i sumar-li un; però és més eficient la manera utilitzada, ja que no ha de fer cap cerca més a la taula.
S'agafa la primera entrada en llista, es guarda en una variable el seu *id* i acte seguit s'intercanvien els identificadors (el d'aquesta i el de la entrada anterior). L'element aleatori ara està al principi i el que era el primer està al final. Es recupera doncs l'*id* antic del que ara és el primer i s'imposa com a nou *id* per a l'últim element actual. Acte seguit es marca com a reproduït (perquè no surti a la pròxima iteració) el primer element i es retornen al reproductor, les seves metadades.
A la figura 6.6, es pot veure un petit exemple:

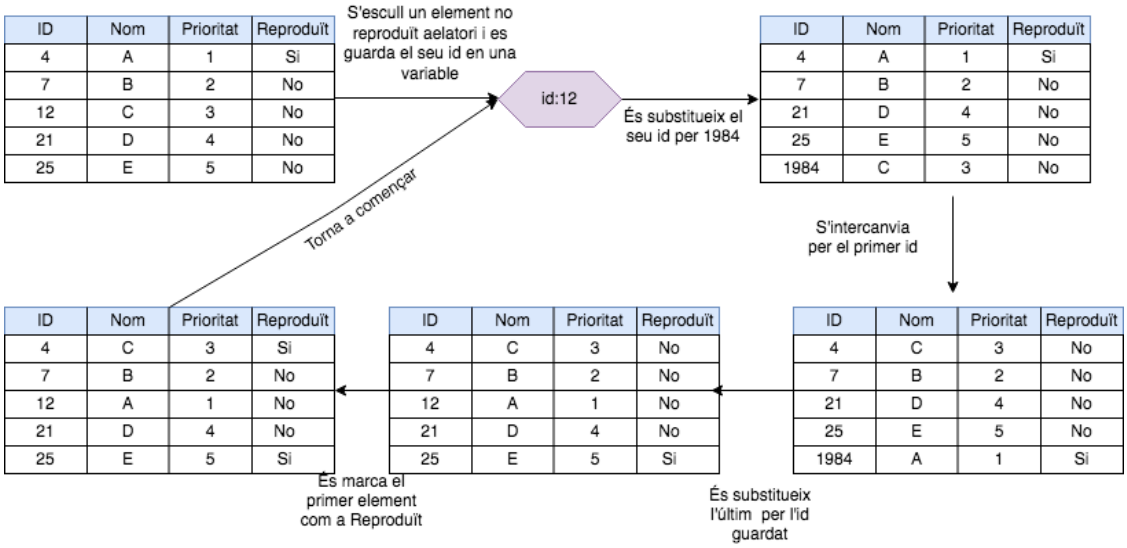


Figura 6.6: Exemple de l'algorisme de reproducció aleatòria (Font: Elaboració pròpia)

Aquest algorisme té una sèrie d'avantatges:

- No utilitza una altra taula per tal de conservar la primera; l'espai que necessita és una variable d'un enter només per guardar l'*id*.
- No altera l'ordre de la llista; en mode seqüencial veuríem els fitxers en el mateix ordre ja que funciona per la prioritat que, amb aquest algorisme, no s'altera
- A la interfície de control, l'usuari veu que el que estava primer a la llista, ara està al final i el que s'està reproduint a la cartellera, es troba al capdavant de la taula que veu.

6.5 Algorisme de reproducció intercalada aleatòria sense repetició (permutació)

Algorisme 5 : algorisme de reproducció intercalada aleatòria

```
Funció: reproducció_intercalada_aleatoria (llista):
Entrada :  (Objecte de classe) llista, booleà torn
Sortida   : element de la classe llista
si torn == aleatori o llista.primerElement != servidor.fitxerIntercalat:
        torn = intercalat
        reproducció_intercalada_aleatoria (llista)
        retorna  reproducció_seqüencial(llista)

sinó:
        torn = aleatori
        retorna servidor.fitxerIntercalat
```

Figura 6.7: Algorisme de reproducció aleatòria intercalada representat amb pseudocodi
(Font: Elaboració pròpia)

A nivell d'algorismes, l'algorisme representat a la figura 6.7 es tracta d'una combinació dels anteriors: l'aleatori i l'intercalat seqüencial, però amb una petita variació. Per tal de no repetir tants cops el fitxer que consta com a intercalat quan no és el seu torn, es comprova que no s'hagi reproduït just abans, en el torn de reproducció aleatòria. Això és fàcil, ja que només ha de comprovar si la primera entrada a la llista coincideix amb l'entrada guardada en la variable com a intercalada. Aquesta variació era necessària per tal d'evitar el següent cas representat a la Figura 6.8:

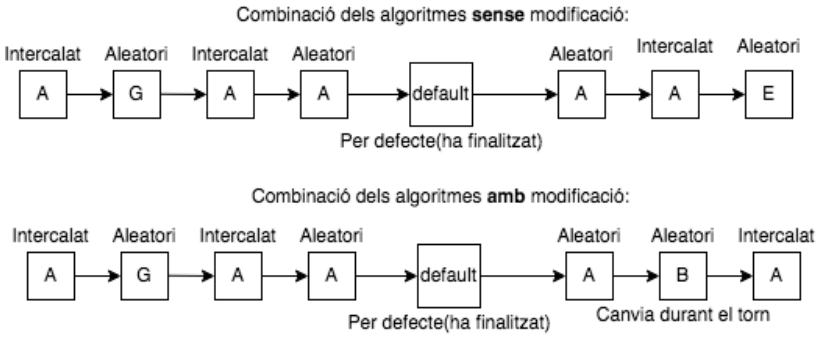


Figura 6.8: Exemple de l'algorisme de reproducció aleatòria intercalada (Font: Elaboració pròpia)

En aquest cas extrem, l'usuari davant de la cartellera i sense veure la llista de reproducció, veuria l'element A massa sovint. De fet, l'únic que el separa de veure'l 4 cops seguit és el fitxer per defecte. Amb la modificació, en canvi, en un cas com aquest l'usuari només el veuria com a màxim dos cops seguits, en la mateixa o en diferent "passada" de la llista.
A banda d'aquesta petita variació, els dos algoritmes funcionen de la mateixa manera internament (quan és el seu torn) que abans.

7. Resultats

7.1 Proves realitzades

A les reunions de cada *sprint*, s'observava el progrés realitzat al projecte a partir d'una demostració funcional de les aplicacions.
D'aquesta manera, les propostes de millora o proves concretes sobre funcionalitats, es podien fer poc després d'haver-se implementat i no en la prova total del sistema, a la fase de testeig (establerta dues setmanes abans de finalitzar-ho tot), ja que hi hauria menys temps i molts més problemes acumulats. En aquest sentit, les proves van anar molt bé. Però tot i així, el problema és que aquestes demostracions eren d'una duració suficient per poder mostrar la feina feta però no tan llargues ni exhaustives per poder trobar alguns *bugs* de difícil reproducció.
També, al principi, les proves es centraven sobre *back end* i s'utilitzava una interfície de control de prova per realitzar-les. Això facilitava molt els tests de funcionalitats. El problema però, va ser en implementar el disseny de la interfície de control final, ja que van sortir molts *bugs* (a la part de *front end*). Això és degut a que aquesta versió era més complexa que la de prova i contenia molts elements nous, com ara una taula paginada amb cercador.

El testeig per part del client es va realitzar just en acabar el cinquè *sprint* (el mateix on es va implementar tota la interfície d'usuari nova). El client va disposar del sistema i va realitzar proves durant un llarg període de temps. Consistien en un ús a fons de les funcionalitats de l'aplicació de manera repetitiva. Per això van sortir molts *bugs* que no havien sigut detectats. Per exemple, si hi havia massa elements a la taula, aquesta els distribuïa per defecte en pàgines de només cinc elements; vaig haver de treure aquest límit per no tenir tantes pàgines que dificultaven la visualització del llistat de contingut. A banda de solucionar-los, es van voler implementar noves funcionalitats, que alhora, van provocar nous *bugs* (tot i que cap greu). Es va decidir doncs allargar el termini per finalitzar el sistema 15 dies més. Els *bugs* més importants es troben a la següent figura 7.1, i les funcionalitats demanades (i implementades) duran aquest període, a la figura 7.2:

Problema	Explicació	Solució
Quan s'utilitza el botó de seleccionar tots els elements però sense haver-ne seleccionat un prèviament no apareixen els botons.	La variable que feia mostrar al *front end* només canviava dins del mètode que es crida a seleccionar un element.	S'ha afegit la modificació de la varibale en el metode que controla la selecció múltiple a la taula.
Cada certs minuts totes les crides **a *end points* que estan protegits retornen errors 400** No es pot afegir cap usuari, afegir contingut, etc. En canvi si tanques la sessió i tornes a iniciar-la, funciona fins que passa un altre cop.	El mètode que genera els *tokens* a Flask tenen una caducitat que es pot canviar, però si no s'especifica el contrari, s'aplica una de 15 minuts.	S'ha modificat el temps de caducitat i s'avisa a l'usuari que ha de tornar a iniciar sessió per tal de seguir utilitzant la interfície de control.
La *playlist* sempre conté l'element reproduint-se al final de la taula i el següent al principi. Hauria d'aparèixer l'actual en primera posició i el següent a reproduir-se a la segona	L'algorisme de reproducció estava dissenyat per fer un treure i retornar l'element primer de la cua.	Modificar l'algorisme per tal d'eliminar l'anterior i retornar el nou primer.
Un usuari administrador que ha iniciat sessió, pot esborrar el seu propi compte.	El sistema no tenia en compte qui demanava la petició i podia deixar sense administradors si s'esborrava l'únic que hi havia	Cap administrador pot esborrar el seu propi compte (des del que ha iniciat sessió) de manera que com a mínim, si el vol esborrar, necessitarà tenir-ne un altre.

Figura 7.1: llistat dels *bugs* més importants trobats durant el testeig (Font: Elaboració pròpia)

Petició	Explicació	Solució
Surten múltiples missatges confirmant que s'han afegit els elements a la llista.	La confirmació del servidor es feia individualment.	S'ha eliminat les alertes de confirmació d'elements afegits a la llista correctament i només es mostren les informacions d'errors.
Modificar el comportament de la llista per tal de poder tenir prioritat a algun fitxer.		Afegit un mode nou que té en compte si el seu torn és de reproducció seqüencial o si ha de reproduir el fitxer seleccionat com a intercalat. *Veure l'explicació de l'algorisme dins de l'apartat.*
Modificar el comportament de la llista per tal de poder reproduir els fitxers de manera aleatòria i amb certa prioritat		Afegits dos modes nous; un de reproducció amb permutació i un altre que és la combinació d'aquest amb l'intercalat
Poder seleccionar els modes des de la interfície d'usuari		Creació d'un llistat d'opcions a la pestanya de "llista de reproducció" per poder seleccionar entre els diferents modes

Figura 7.2: llistat de les noves funcionalitats demanades per a l'últim *sprint* (Font: Elaboració pròpia)

Finalment, es va poder donar per finalitzat el procés de desenvolupament i es va procedir a la instal·lació física del sistema que es troba explicada al següent apartat 7.2.1.

7.2 Resultats

7.2.1 Interfície d'usuari resultant

Basant-me en el disseny elaborat i afegint les funcionalitats requerides a l'*sprint* final,
La interfície de control ha resultat així:

Figura 7.3: Captura de pantalla de la interfície d'usuari final. Finestra d'inici de sessió (Font: Elaboració pròpia)

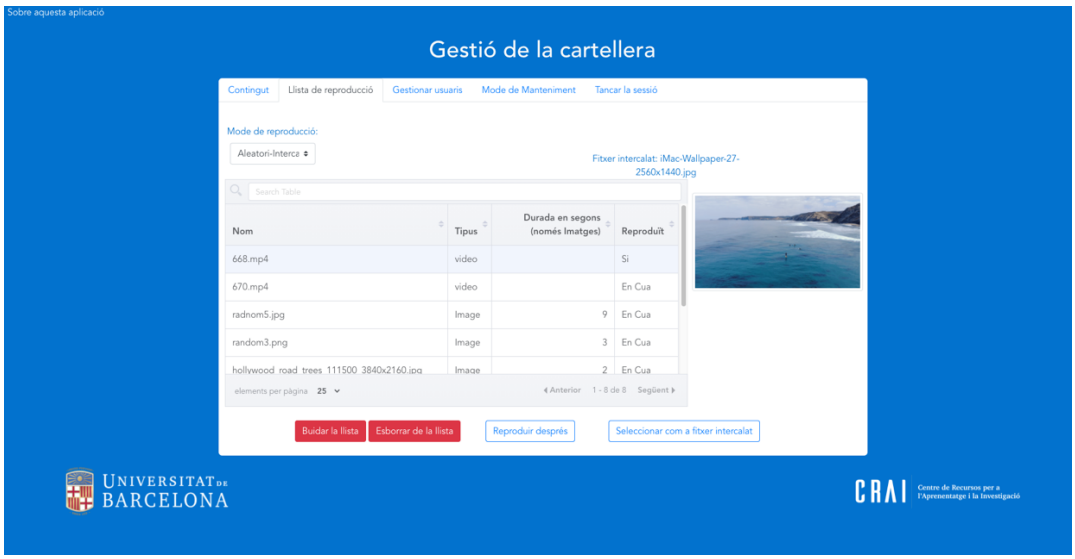


Figura 7.4: Captura de pantalla de la interfície d'usuari final. Pestanya "llista de reproducció"(Font: Elaboració pròpia)

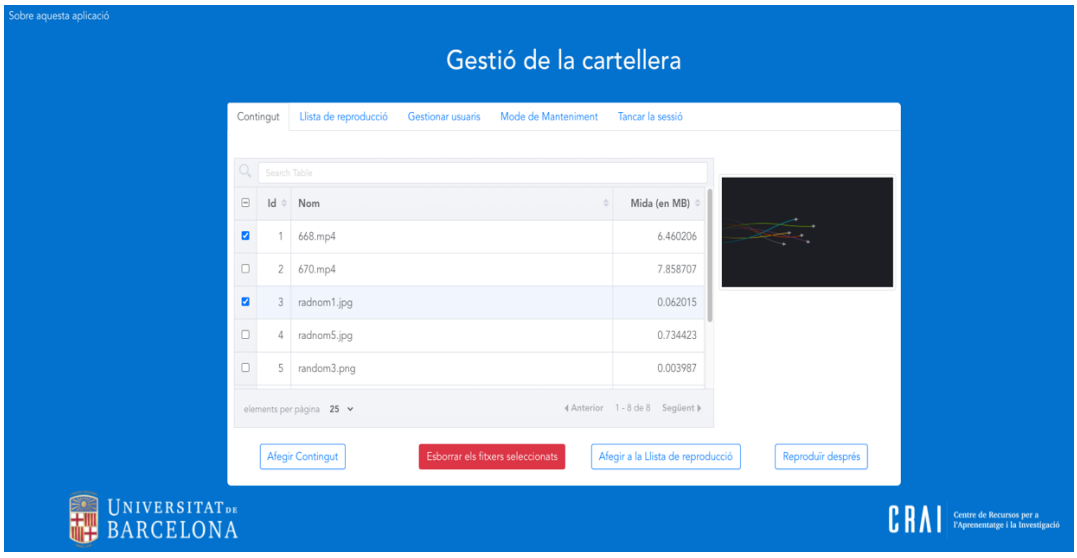


Figura 7.5: Captura de pantalla de la interfície d'usuari final. Pestanya "contingut"(Font: Elaboració pròpia)

Figura 7.6: Captura de pantalla de la interfície d'usuari final. Secció "afegir contingut" pertanyent a la pestanya "contingut" (Font: Elaboració pròpia)

Figura 7.7: Captura de pantalla de la interfície d'usuari final. Pestanya "Gestió d'usuaris" (Font: Elaboració pròpia)

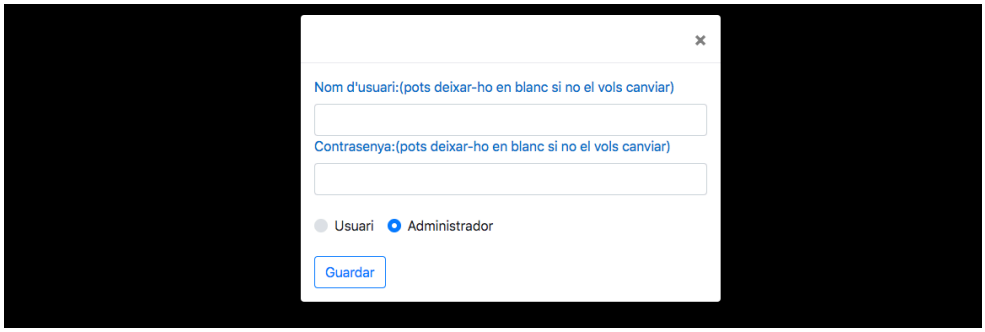


Figura 7.8: Captura de pantalla de la interfície d'usuari final. Finestra emergent de modificació d'usuaris pertanyent a la pestanya "Gestió d'usuaris" (Font: Elaboració pròpia)

Figura 7.9: Captura de pantalla de la interfície d'usuari final. Pestanya "Mode manteniment" (Font: Elaboració pròpia)

Figura 7.10: Captura de pantalla de la interfície d'usuari final. Pestanya "Tancar la sessió" (Font: Elaboració pròpia)

7.2.2 Instal·lació física resultant

A causa de les limitacions de la Raspberry, finalment per poder programar l'engegada, es va recórrer a una solució de *hardware*: un endoll programable que es pot veure a la figura 7.11. Té programat l'horari del CRAI i pel propi funcionament de la Raspberry, en rebre corrent, inicia el sistema automàticament.
En canvi per apagar, es fa servir una solució basada en *software* (un *script* en Bash) que atura tot el funcionament del sistema operatiu uns minuts abans que ho faci l'endoll per tal d'apagar correctament la Raspberry Pi.


Figura 7.11: Endoll programable instal·lat al CRAI Biblioteca de Matemàtiques i Informàtica (Fotografia: Roger Angela)

Figura 7.12: ubicació de la instal·lació de la raspberry al CRAI Biblioteca de Matemàtiques i Informàtica (Fotografia: Roger Angela)

La TV es troba col·locada amb un suport de paret, de manera que la placa s'havia de col·locar enganxada darrere, tal com es pot veure a la figura 7.12 i gràcies a les dimensions reduïdes que té, no ha sigut cap problema.

8. Conclusions

8.1 Treball Futur

A mesura que s'ha anat finalitzat el projecte, han sorgit noves idees que podrien aplicar-se conjuntament amb algunes idees inicials que no s'han arribat a desenvolupar. Aquestes darreres són:

- Programar en el temps un contingut. Podria ser interessant que es pogués programar un contingut amb total antelació (des de mesos fins minuts abans) en una data. Per exemple, que el dia de la Matefest-Infofest mostri per pantalla a les 12:00 que hi ha un concurs. Un altra idea podria ser que en comptes de reproduir-se en un cop, es reproduís conjuntament amb la resta d'elements només en un període donat, com ara un contingut sobre Nadal només els dies abans de marxar al desembre.

- Tenir una tercera aplicació web amb què es pogués controlar l'apagada, reinici de la placa o reinici del Docker. Ara per ara, per poder fer aquestes operacions, cal connectar-se per SSH a la Raspberry. Degut a la naturalesa de la virtualització a Docker, no es poden controlar els aspectes del sistema operatiu des del codi que s'està executant dins del contenidor (tot i que des d'una *app* en Python sí que es pot).

- Poder "reproduir" documents. Això ha quedat implementat en part, ja que per la part de *back end* no té cap problema i només hauria de temporitzar el temps que es veu per pantalla. Però per a presentacions Power Point o per documents de més d'una pàgina també caldria implementar tant a *back end* com a *front end* l'opció de temporitzar cada pàgina, ja que com la cartellera no és tàctil ni té idea de ser interactiva amb els usuaris que la veuen (excepte pel personal autoritzat i fent ús de l'aplicació de control) no es podria avançar la navegació.

També altres que van sorgir durant el desenvolupament:

- Crear llistes personalitzades. De manera que el sistema tingués diferents llistes com ara una per a utilitzar cada mes, cada semestre... També podrien ser privades per a cada usuari i que l'usuari escollís quina es reprodueix. Tot i així, el sistema d'usuaris no està pensat d'aquesta manera, llavors podria ser que hi hagués llistes quasi idèntiques ocupant espai a memòria i que fossin molt similars (quant als elements). També sorgiria un nou problema d'aquesta idea: podria ser que dos usuaris volguessin mostrar per pantalla continguts diferents i s'haurien de posar d'acord sobre quina llista hauria de mostrar-se.

- Fer que els fitxers multimèdia tinguessin una prioritat i els algoritmes les tinguessin en compte com a ordre per reproduir. Uns elements marcats com a favorits, que es reproduirien amb més freqüència.

- Poder fer selecció múltiple de contingut a la taula de la llista de reproducció en comptes de seleccionar-ne un o tots (com funciona ara). Això va ser treballat, però el component de la taula (vue-good-table) no permet mantenir la selecció a cada actualització de les dades, que per tal de funcionar a temps real i poder saber en tot moment què s'està reproduint sense mirar la cartellera, es fa una petició de les dades *a l'end point playlist* cada segon. La solució doncs seria utilitzar un altre component de *vue* que ho permetés.

- Possibilitat de renombrar fitxers, ja que ara per ara, és una tasca que s'ha de realitzar prèviament a l'enviada d'arxius a la Raspberry.

- 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.
- Poder combinar la reproducció dels fitxers multimèdia amb una pantalla com la de la següent figura:

Figura 8.1: exemple de disseny que combina elements multimèdia amb interactius (Font: Elaboració per part de Roger Angela)

8.2 Conclusions

Aquest projecte ha suposat la consolidació de coneixements i habilitats adquirides a les diferents assignatures cursades al llarg del Grau. També l'aprenentatge de nous, sobretot dels orientats a la programació d'aplicacions web. Personalment, ha resultat una experiència diferent, ja que es tractava d'implementar una solució per poder ser utilitzada al món real, a diferència de molts treballs previs realitzats a la Facultat. Això ha comportat haver d'estudiar profundament tots els detalls i tenir en tot moment en compte totes les possibles accions per part dels usuaris, per evitar que puguin dur el sistema a un estat d'error, com mai ho havia fet abans.

Les prestacions principals del sistema aconseguides amb tot el procés explicat en aquest document són:

Senyalització digital capaç de visualitzar imatges i vídeos.

Gestió de la reproducció, de la configuració i dels elements multimèdia a través de la xarxa local mitjançant la interfície d'usuari elaborada.

Accés limitat (a la modificació de la configuració) al personal autoritzat, mitjançant credencials d'usuari.

Possibilitat d'escollir entre diferents modes de reproducció per tal d'alterar l'ordre de visualització del contingut multimèdia al sistema.

Compatibilitat multi plataforma del *software* desenvolupat

Facilitat d'instal·lació, facilitant l'exportació (si es dóna el cas) a altres centres.

Per tant, considero que el sistema de senyalització digital dissenyat i implementat compleix els requeriments inicials demanats pel client i pot donar-se el projecte per finalitzat. Tot i així, sense descartar que el seu procés de desenvolupament es podria continuar per tal de realitzar noves funcionalitats com les que es troben al subapartat anterior.

Annexos

Glossari

Back end: Capa del software no visible per al 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.

NUC computer: ordinadors de format compacte del fabricant Intel. Molt útils per realitzar solucions com la tractada en aquesta memòria.

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 basat en la transferència d'estats representacionals (REST).

Figures

Figura A1: *Product Backlog*

US:	Com a ...	Vull	Per tal de	Es èpica?	Criteri d'acceptació
US01	TOTS	Veure el contingut per pantalla		si	
US02	Usuari	Enviar contingut al dispositiu de la cartellera des de l'app de gestió	Poder reproduir-ho	No	L'arxiu que l'usuari vol reproduir es troba al disc dur de la Raspbery
US03	Usuari	Veure el contingut que hi ha al dispositiu des del meu ordinador	Poder gestionar-ho	Si/No	Es veu que correspon els fitxers accedint físicament al dispositiu amb el llistat que tenim a l'aplicació de gestió
US04	Usuari	Eliminar contingut del dispositiu de la cartellera	No disposar més d'ell	No	Que es pugui comprovar que no hi és físicament l'arxiu a l'emmagatzematge de la Raspberry
US05	Usuari	Veure la llista de reproducció	Saber l'ordre de reproducció	Si	Observar que l'ordre dels fitxers que s'està mostrant per pantalla coincideix amb el mostrat
US06	Usuari	Seleccionar per a reproduir un contingut a continuació del que s'està reproduint	imposar un fitxer immediatament després	No	És reprodueix seguit del fitxer actual un marcat prèviament
US07	Usuari	Treure elements de la Cua de reproducció	no reproduir-los més	No	Al eliminar el fitxer de la llista, després de reproduir-la sencera , no s'ha visualitzat l'element cap cop
US08	Usuari	Programar la reproducció d'un element donada una data	reproduir un fitxer en un dia i hora concrets	No	Que es pugui visualitzar el contingut desitjat a la data prèviament especificada.
US09	Usuari	Apagar el dispositiu de la cartellera localment	No haver d'apagar el dispositiu amb perifèrics connectats	no	Que el dispositiu s'apagui des de l'ordinador de control
US10	Usuari	Reiniciar el dispositiu de la cartellera localment	tornar a tenir el sistema en funcionament en cas d'aturada	No	Que el dispositiu s'apagui des de l'ordinador de control i torni a iniciar l'app de cartellera automàticament
US11	Usuari	Programar l'apagada del dispositiu de la cartellera	No haver d'apagar el dispositiu manualment	No	Que a l'hora indicada el dispositiu deixi de funcionar
US12	Usuari	Programar l'inici del dispositiu de la cartellera	No haver d'iniciar el dispositiu manualment	No	Que a l'hora indicada el dispositiu s'engegui i automàticament obri l'app de la cartellera començant la reproducció
US13	Usuari	Visualitzar documents temporalment	veure'ls a la cartellera	No	És mostra el document cada cert temps i després canvia al següent element de la llista
US14	Usuari	Visualitzar temporalment imatges	veure-les a la cartellera	si	És mostra la imatge cada cert temps i després canvia al següent element de la llista
US15	Usuari	Iniciar sessió	Accedir a la aplicació	Si	L'usuari inici sessió amb unes credencials i només així pugui veure la interfície de control
US16	Usuari	Tancar la sessió	Sortir de la aplicació	Si	L'usuari tanqui la sessió i deixi de veure la interfície de control
US17	Usuari	tenir una interfície de control de la cartellera al meu navegador	no haver d'accedir físicament al dispositiu per tal de modificar el comportament de reproducció	Si	
US18	Usuari	Que no tots els elements enviats al dispositiu s'afegeixin a la cua de reproducció	tenir més control sobre la reproducció	Si	Al enviar un fitxer multimèdia mitjançant l'app des de el navegador no s'afegeix directament a la llista de reproducció

Figura A2: Sprint Backlog

US:	Com a ...	Vull	Sprint	Estimació (en hores)
US01	TOTS	Veure el contingut per pantalla	1	30
US02	Usuari	Enviar contingut al dispositiu de la cartellera des de l'app de gestió	2	13
US03	Usuari	Veure el contingut que hi ha al dispositiu des de el meu ordinador	2	17
US04	Usuari	Eliminar contingut del dispositiu de la cartellera	3	8
US05	Usuari	Veure la llista de reproducció	3	4
US06	Usuari	Seleccionar per a reproduir un contingut a continuació del que s'està reproduint	3	10
US07	Usuari	Treure elements de la Cua de reproducció	3	6
US08	Usuari	Programar la reproducció d'un element donada una data	4	21
US09	Usuari	Apagar el dispositiu de la cartellera localment	4	9
US10	Usuari	Reiniciar el dispositiu de la cartellera localment	5	2
US11	Usuari	Programar l'apagada del dispositiu de la cartellera	5	6
US12	Usuari	Programar l'inici del dispositiu de la cartellera	6	10
US13	Usuari	Visualitzar documents temporalment	6	2
US14	Usuari	Visualitzar temporalment imatges	6	6
US15	Usuari	Iniciar sessió	6	2
US16	Usuari	Tancar la sessió	6	1
US17	Usuari	tenir una interfície de control de la cartellera al meu navegador	6	1
US18	Usuari	Que no tots els elements enviats al dispositiu s'afegeixin a la cua de reproducció	6	1

Figura A3: *Sprint Backlog* de la replanificació

Com a ...	Vull	Per tal de	Sprint	Fet ?
TOTS	Veure el contingut per pantalla		1	Sí
Usuari	Enviar contingut al dispositiu de la cartellera	Poder reproduir-ho	2	Sí
Usuari	Veure el contingut que hi ha al dispositiu des del meu ordinador	Poder gestionar-lo	2	Sí
Usuari	Eliminar contingut del dispositiu de la cartellera	No disposar més d'ell	3	Sí
Usuari	Veure la llista de reproducció	Saber l'ordre de reproducció	5	Sí
Usuari	Reproduir un contingut a continuació del que s'està reproduint	Imposar un fitxer immediatament després	4	Sí
Usuari	Treure elements de la cua de reproducció	No reproduir-los més	5	Sí
Usuari	Reproduir la llista en un ordre aleatori sense repetició (*Shuffle*)	No veure la mateixa seqüencia de reproducció tota l'estona	7	Sí
Usuari	Reproduir la llista en ordre seqüencial amb elements intercalats	Donar més prioritat a un fitxer	7	Sí
Usuari	Reproduir la llista en ordre aleatori amb elements intercalats	Donar més prioritat a un fitxer però sense que repeteixi la seqüencia de reproducció	7	Sí
Usuari	Escollir l'element intercalat	Donar-li més prioritat de reproducció que a la resta	7	Sí
Usuari	Escollir el mode de reproducció	Canviar l'ordre de reproducció	7	Sí
Usuari	Un mode de manteniment	Gestionar el contingut correctament	3	Sí
Usuari	Iniciar sessió	Accedir a l'aplicació	6	Sí
Usuari	Tancar la sessió	Sortir de l'aplicació	6	Sí
Administrador	Crear comptes d'usuari	Que ells puguin accedir a l'aplicació de control	6	Sí
Administrador	Esborrar comptes d'usuari	Denegar l'accés a l'aplicació de control	6	Sí
Administrador	Modificar els noms dels comptes d'usuari	Poder canviar el nom d'accés	6	Sí
Administrador	Modificar les contrasenyes dels comptes d'usuari	Poder canviar la contrasenya d'accés	6	Sí
Administrador	Modificar el rol dels usuaris	Poder donar/treure permisos als usuaris	6	Sí
Administrador	Impedir esborrar el meu propi compte des de el meu compte mateix	No eliminar accidentalment el propi compte d'administrador	6	Sí
Usuari	Programar la reproducció d'un element donada una data	Reproduir un fitxer en un dia i hora concrets		No
Usuari	Apagar el dispositiu de la cartellera remotament	No haver d'apagar el dispositiu amb perifèrics connectats	3	Sí
Usuari	Reiniciar el dispositiu de la cartellera remotament	Tornar a tenir el sistema en funcionament en cas d'aturada	7	Sí
Usuari	Programar l'apagada del dispositiu de la cartellera	No haver d'apagar el dispositiu manualment	7	Sí
Usuari	Programar l'inici del dispositiu de la cartellera	No haver d'iniciar el dispositiu manualment		No
Usuari	Visualitzar documents temporalment	Veure'ls a la cartellera		No
Usuari	Visualitzar temporalment imatges	Veure-les a la cartellera	3	Sí
Usuari	Definir la duració de visualització de cada imatge	Decidir quant temps es mostra	5	Sí
Usuari	Poder veure el progrés d'enviament del contingut	Saber l'estat de la tramesa	3	Sí
Usuari	Poder veure miniatures al llistat de contingut	Saber com és el fitxer	3	Sí
Usuari	Poder veure miniatures a la llista de reproducció	Saber com és el fitxer a reproduir	3	Si
Usuari	Poder seleccionar com a favorit un element de la llista de reproducció	Que es reprodueixi amb més freqüència		No
Usuari	Poder utilitzar un cercador en la llista de reproducció	Trobar un element	5	Sí
Usuari	Poder utilitzar un cercador al llistat de contingut	Trobar un element	5	Sí
Usuari	Tenir paginada la llista de reproducció	No haver de fer un *scroll* molt llarg per arribar al final	5	Sí
Usuari	Tenir paginat el llistat de contingut	No haver de fer un *scroll* molt llarg per arribar al final	5	Sí
Usuari	Poder buidar tot el llistat de contingut amb un sol clic	No haver de clicar/seleccionar a cada element	7	Sí
Usuari	Eliminar tota la llista de reproducció amb un sol clic	No haver de fer clicar/seleccionar a cada element	7	Sí
Usuari	Selecció múltiple dels continguts	No haver d'afegir els continguts d'un en un	7	Sí

Figura A4: Diagrama de Classes Aplicació Cartellera

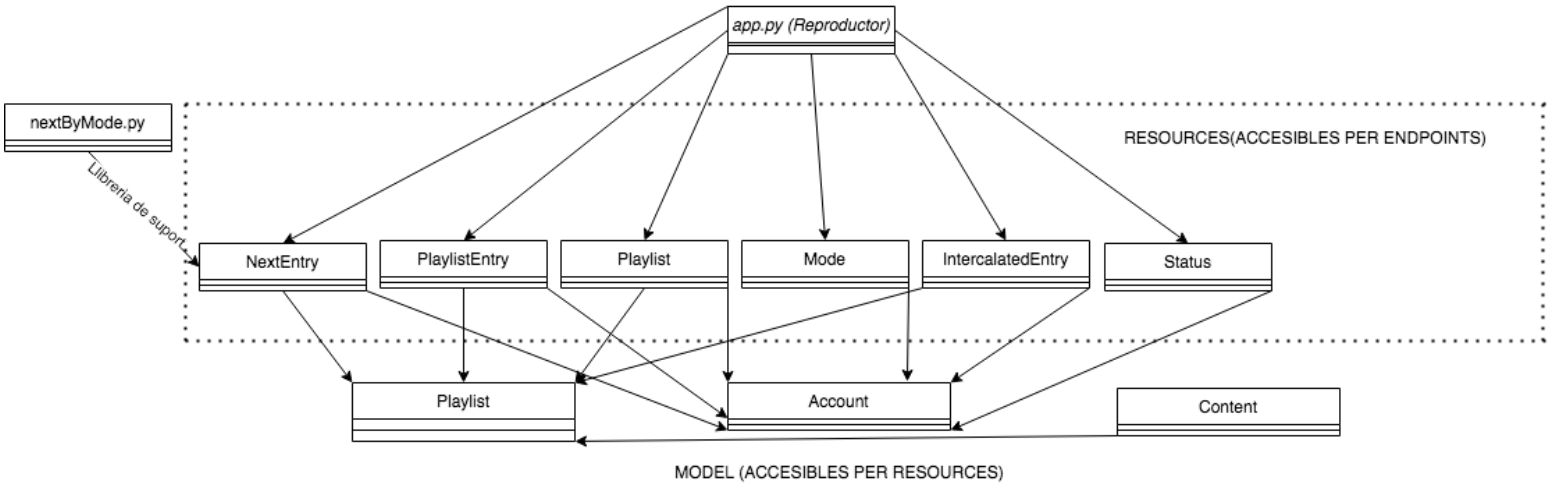
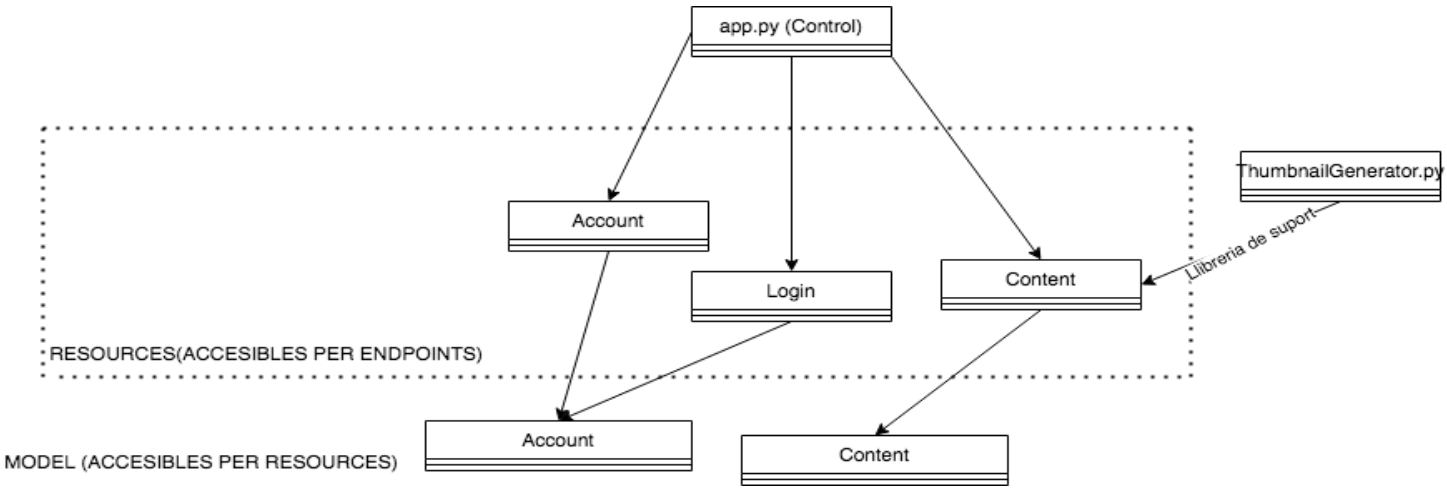


Figura A5: Diagrama de Classes Aplicació Control



Llistat complet *end points*

1.Aplicació Cartellera

/mode *Accès limitat amb credencials:* Si
GET: /mode

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"mode": “inter”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

PUT: /mode

EXEMPLE COS DE PETICIÓ:
{
"mode": “inter”
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

/status *Accès limitat amb credencials:* No
GET: /status

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"message": “online”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

PUT: /status

EXEMPLE COS DE PETICIÓ:
{
"message": “stop”
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

/playlistEntry *Accès limitat amb credencials:* Si
DELETE: /playlistEntry/<string:name>

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

POST: /playlistEntry

EXEMPLE COS DE PETICIÓ:
{
"data": {
“name”: “1.jpg”,
“type”: “Image”,
“duration”: 8,
}
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

/intercaledEntry *Accès limitat amb credencials:* Si
DELETE: /intercaledEntry

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

POST: /intercaledEntry

EXEMPLE COS DE PETICIÓ:
{
"data": {
“name”: “1.jpg”,

“type”: “Image”,
“duration”: 8,
}
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

GET: /intercaledEntry

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"data": {
“name”: “1.jpg”,
“type”: “Image”,
“duration”: 8,
}
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}
/nextEntry *Accès limitat amb credencials:* Parcialment
POST: /nextEntry *Accès limitat amb credencials:* Si

EXEMPLE COS DE PETICIÓ:
{
"data": {
“name”: “1.jpg”,
“type”: “Image”,
“duration”: 8,
}
}
200 COS DE RESPOSTA:
{
"message": “Petició processada correctament”
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

GET: /nextEntry *Accès limitat amb credencials:* NO

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"data": {
“name”: “1.jpg”,
“type”: “Image”,
“duration”: 8,
}
}

400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

/playlist *Accès limitat amb credencials:* Si
GET: /playlist

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
"data": {
{
“name”: “1.jpg”,
“type”: “Image”,
“duration”: 8,
“path”: “./static/1.jpg”,
“priority”: 8,
“isPlayed”: 0,
},
}
}
400 COS DE RESPOSTA:
{
"message": “Hi ha hagut un problema amb la teva petició”
}

2.Aplicació Control

/account *Accès 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": “Usari amb ['username': user1] esborrat correctament"
}
400 COS DE RESPOSTA:
{
"message": “Usari amb ['username': user1] no trobat"
}

/accounts *Accès 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"
}

/content *Accès limitat amb credencials:* Si
GET:
EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{
content: [{id: 1, name: "radnom1.jpg", path: "/app/media/radnom1.jpg", size: "62015"},…]
}
400 COS DE RESPOSTA:
{
"message": “Usuari no trobat ”
}

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> *Accès limitat amb credencials:* Si
GET:

EXEMPLE COS DE PETICIÓ:
BUIT
200 COS DE RESPOSTA:
{<thumbnail file>}
400 COS DE RESPOSTA:
{
"message": “Thumbnail no trobat ”
}

Manual d'instal·lació

Per tal de poder fer servir les aplicacions els únics requisits previs són:
Tenir instal·lat Docker desktop (Windows i Mac)
Tenir instal·lat Docker i Docker compose (Linux)
Tenir instal·lat un navegador d'internet (Tots)

Preparació de les aplicacions

Degut a la naturalesa del llenguatge Python, el codi no és compilat, serà interpretat en execució dins de les imatges de Docker que generarem a continuació.
Amb la terminal, hem d'anar fins a la carpeta on estigui el codi font.
En cas de Bash (Mac i Linux)

```
cd /"Ruta_previa_al_codi_font"/Codi_font
```

Dins, haurem d'introduir:

```
docker-compose build
```

(depenent dels recursos de l'ordinador pot trigar fins a 40 minuts)

I ja el podríem executar de la manera que veurem a continuació.

Per defecte, al codi està inclòs el *front end* de Vue.Js compilat, no sent necessaris els passos detallats a continuació. Si es vol compilar de nou, es requereix tenir instal·lat el gestor de paquets npm.
Amb el terminal, anem a la carpeta de front end: (partint des de la carpeta de codi font)

```
cd Controller WebApp/frontend/
```

Allà instal·lem npm i bootstrap a la carpeta amb la següent comanda:

```
npm install
npm install --save bootstrap-vue
```

Quan acabi, hem de fer:

```
npm run build; npm run dev
```

Quan surti aquesta pantalla:

```
Your application is running here: http://localhost:8080
```

Fem Control + C per aturar .

Seguidament tornem a la carpeta anterior (l'arrel del codi font):

```
cd -
```

Per tal de generar les imatges de Docker , allà introduïm:

```
docker-compose build
```

(depenent dels recursos de l'ordinador aquest pas pot trigar fins a 40 minuts)

Els scripts inclosos (d'inici de les aplicacions amb visualització de la cartellera automàticament amb l'engegada de la Raspberry, d'aturada i de reinici) només funcionen a Linux.

Configuració d'inici automàtic de les aplicacions i el navegador.

Amb la terminal introduïm (en comptes de pi, el propi username):

```
sudo nano /home/pi/.bashrc
```

I seguidament:

```
./home/pi/start.sh
```

Per executar les aplicacions

Per terminal, accedim a la carpeta codi font com hem fet prèviament al pas d'instal·lació.
Un cop allà, introduïm:

```
docker-compose up
```

Per accedir a les aplicacions, hem d'obrir el 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)
No és necessari tenir obertes les dues *apps* alhora per comprovar el funcionament només d'una.

Manual d'instruccions

La característica principal de la interfície d'usuari és que permet realitzar fàcilment les operacions. En cas de ser necessari, es veu com realitzar-les amb aquest recorregut guiat sobre el panell de control:

Només accedir a la url (http://127.0.0.1:80/controlPanel si és una demo del projecte, si es troba instal·lat, amb la URL del subdomini proporcionada), des de el navegador es pot veure aquesta pantalla de la figura 9.1:

Figura 9.1: Captura de pantalla de la interfície d'usuari final. Panell d'inici de sessió
(Font: Elaboració pròpia)

Les credencials per accedir a la demo adjuntada són [usuari: admin, contrasenya:admin1998]

Gestió de la reproducció

Un cop comprova les credencials, entrem al panell de gestió de la cartellera i veiem directament la pestanya de la llista de reproducció de la figura 9.2.

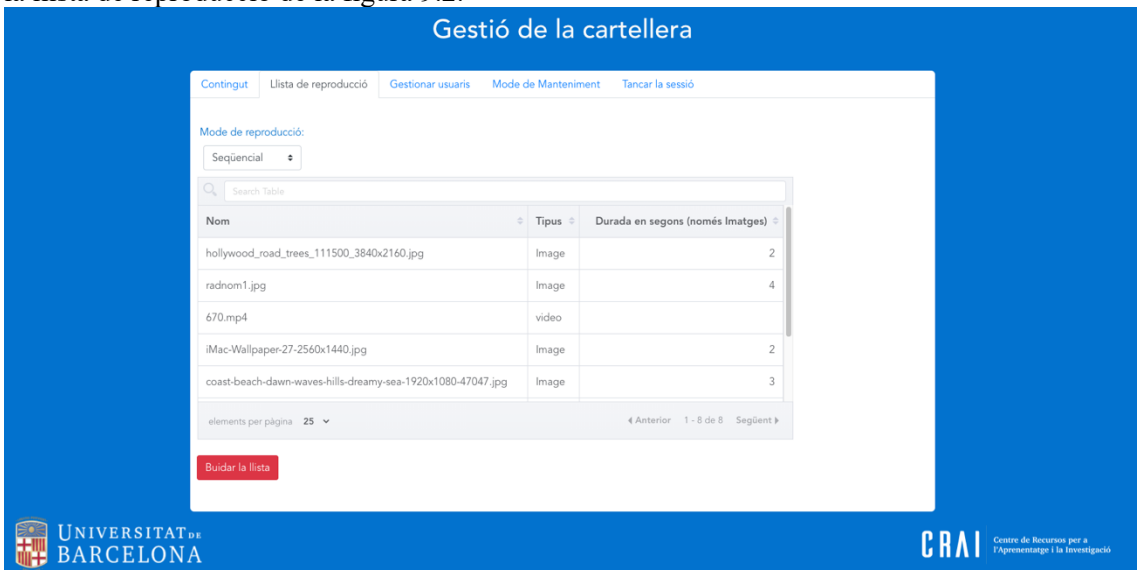


Figura 9.2: Captura de pantalla de la interfície d'usuari final. Pestanya "Llista de reproducció" sense cap element seleccionat (Font: Elaboració pròpia)

Per defecte surt només un botó , una llista d'opcions per canviar el mode i la llista de reproducció tabulada. El primer començant des de l'esquerra, ens permet buidar la llista de reproducció.

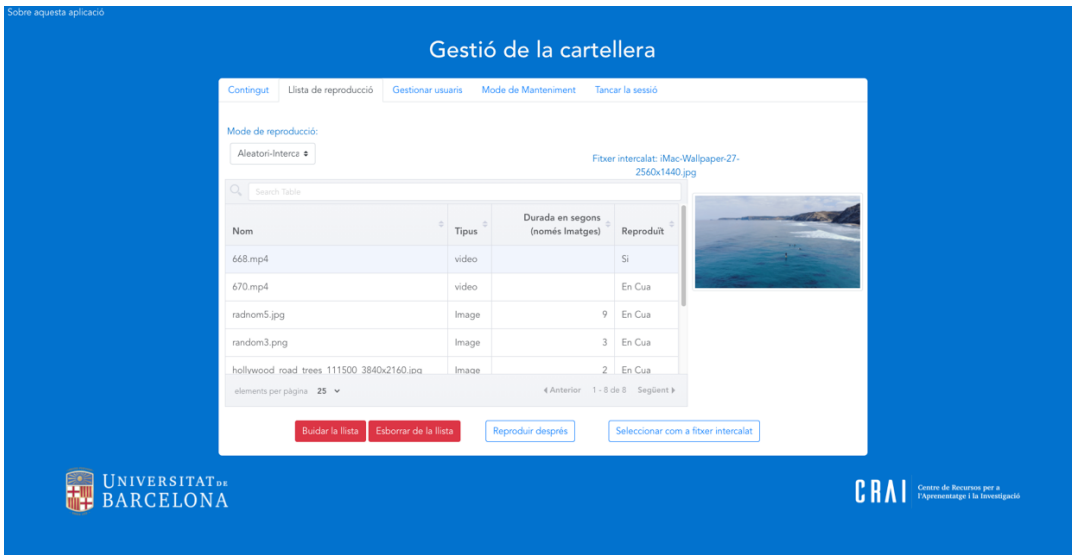


Figura 9.3: Captura de pantalla de la interfície d'usuari final. Pestanya "Llista de reproducció" amb un element seleccionat (Font: Elaboració pròpia)

Al seleccionar un element, com podrem veure a la figura 9.3 que apareixen aquests botons juntament amb una miniatura del fitxer:
Esborrar de la llista (el fitxer seleccionat)

Reproduir després (no alterarà la llista però sigui el mode que sigui, reproduirà abans de continuar, aquest fitxer)

Seleccionar fitxer com a intercalat (només visible als modes intercalats i permet assignar una entrada a la llista com a tal).

En els dos casos anteriors, si es tracta d'un fitxer estàtic, ens mostrarà la finestra emergent de la figura 9.4 per tal de demanar el temps de visualització:

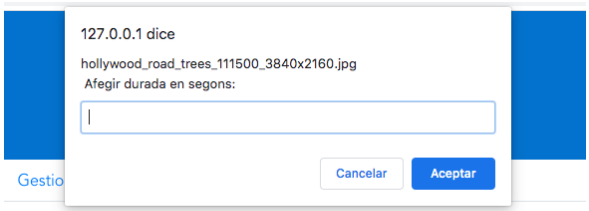


Figura 9.4: Captura de pantalla de la interfície d'usuari final. Finestra emergent duració de la visualització
(Font: Elaboració pròpia)

En cas de deixar-ho buit, el sistema afegeix per defecte 6 segons

Per dur a terme aquestes funcions, sobre un element desitjat, només cal, seleccionar-ho, i un cop es veu la seva miniatura i es marca de color blau a la taula, prémer el botó de l'acció desitjada.

Modes de reproducció

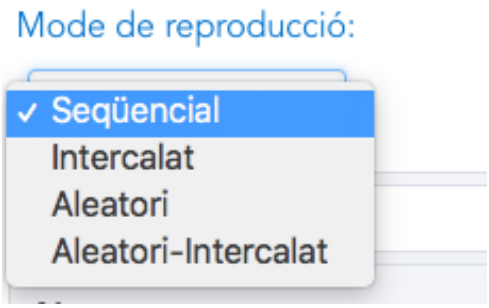


Figura 9.4: Captura de pantalla de la interfície d'usuari final. Pestanya "Llista de reproducció" selector de modes (Font: Elaboració pròpia)

L'ordre de reproducció dels continguts és pot dur a terme en diferents modes:

Mode Seqüencial:
L'ordre de la reproducció de fitxers en aquest mode és la esperada en un cua FIFO (*First-in*, *first-out*). Un cop acabada reproducció de la llista, torna a començar de nou.

Mode Intercalat:
Funciona de manera similar a l'anterior: també es tracta d'una cua FIFO però entre element i element es reprodueix un fitxer seleccionat prèviament per l'usuari com a intercalat. D'aquesta manera, es pot fer que un contingut important es visualitzi amb molta més freqüència que els altres.
Si l'usuari no selecciona cap fitxer, per defecte serà default.mp4

Mode Aleatori:
Per tal d'evitar que és reprodueixi un fitxer més cops que altres, aquest mode es tracta d'una permutació de l'ordre de la llista de manera aleatòria sense repetició. Funciona de la mateixa manera que un sorteig: Cada cop que hi ha un element seleccionat com a candidat, es treu fora i queden per les pròximes iteracions de l'algorisme els altres no escollits. Quan acaba de reproduir l'últim fitxer: reprodueix default.mp4 i torna a marcar com a "candidats" tots els elements de la llista. Fent que l'ordre cada cop sigui diferent.

Mode Aleatori-Intercalat:
Es tracta d'una combinació dels dos modes esmentats anteriorment. La llista de reproducció farà la selecció d'elements de manera aleatòria (tipus *shuffle*) però entre cadascú intercalarà un fitxer seleccionat prèviament.

Enviar un fitxer nou a la cartellera

Per tal d'afegir contingut nou al sistema per tal de poder reproduir-lo:
Primer canviem de pestanya i anem a la de contingut.

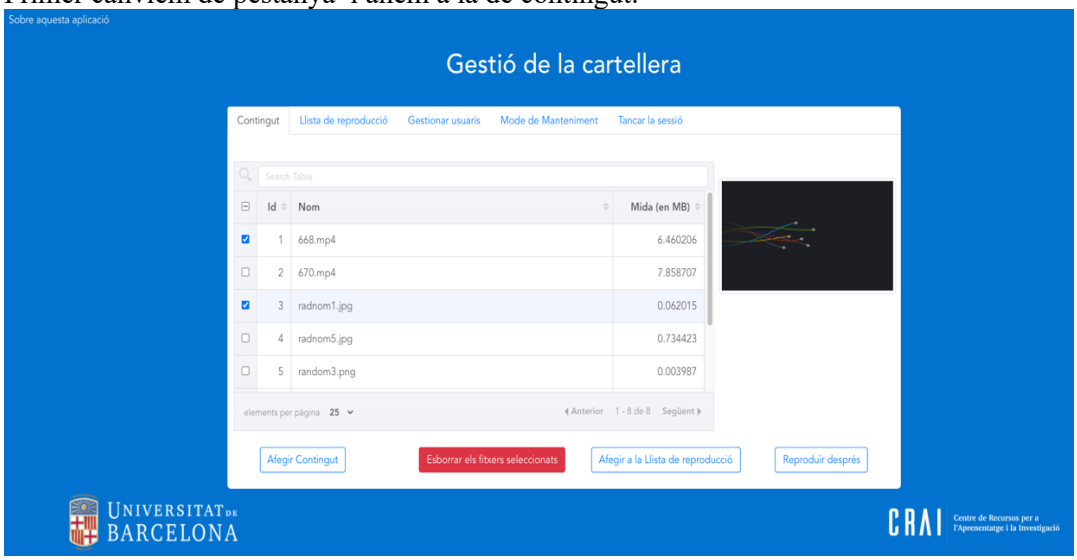


Figura 9.5: Captura de pantalla de la interfície d'usuari final. Pestanya "Contingut"
(Font: Elaboració pròpia)

Acte seguit, premem el botó "afegir contingut" . El contingut de la pestanya canviarà totalment per un com aquest:

Figura 9.6: Captura de pantalla de la interfície d'usuari final. Pestanya "Contingut" apartat afegir contingut (Font: Elaboració pròpia)

Un cop aquí, podem arrossegar els fitxers a la finestra o fer clic al rectangle i s'obrirà una finestra emergent que ens deixarà seleccionar múltiples fitxers del nostre ordinador.

Figura 9.7: Captura de pantalla de la interfície d'usuari final. Pestanya "Contingut" apartat afegir contingut després d'afegir fitxers (Font: Elaboració pròpia)

Al afegir-ne, sortiran les miniatures (si es tracta d'imatges) i una barra per informar del progrés. Aquí tenim dues opcions, descartar l'enviada dels fitxers, o començar-la.
Si s'ha enviat correctament, sortirà una icona com aquesta de la següent figura:

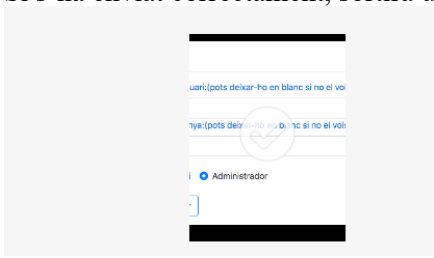


Figura 9.8: Captura de pantalla de la interfície d'usuari final. Fitxer enviat correctament
(Font: Elaboració pròpia)

I en cas contrari, com aquesta altre, descrivint el missatge:

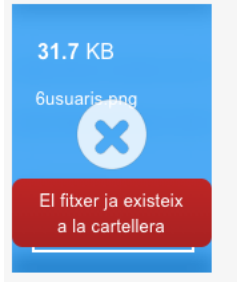


Figura 9.9: Captura de pantalla de la interfície d'usuari final. Fitxer no enviat correctament
(Font: Elaboració pròpia)

Nota: Els noms dels fitxers no han de contenir espais ni caràcters especials com "ñ" o " ç". Tampoc signes d'accentuació.

Per tornar a veure el llistat de fitxers multimèdia, premem el botó d'enrere.

Opcions dels fitxers (eliminar, afegir a llista i reproduir després)

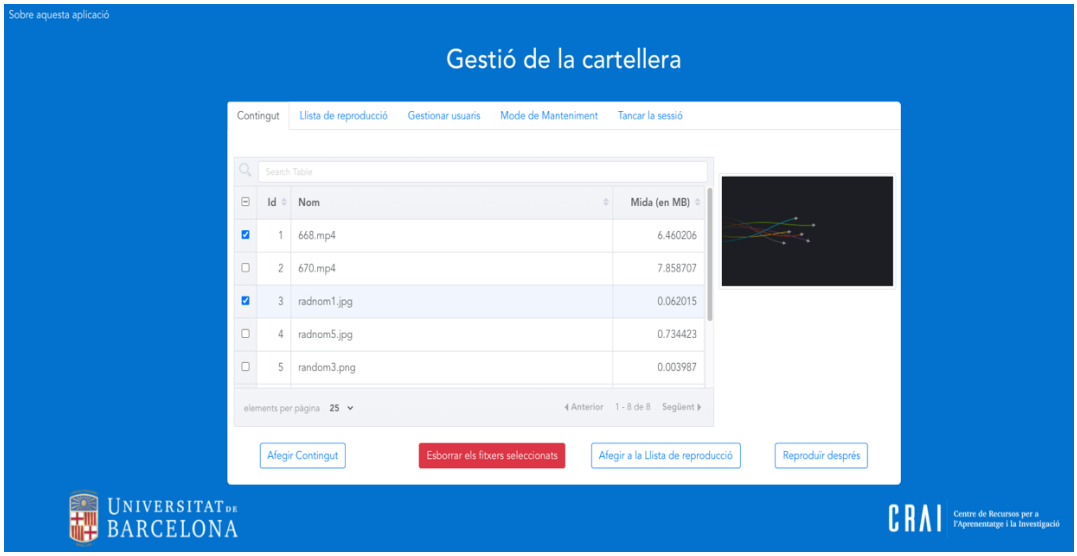


Figura 9.10: Captura de pantalla de la interfície d'usuari final. Pestanya "Contingut"
(Font: Elaboració pròpia)

A la figura 9.10, podem fer diferents operacions sobre els fitxers (mitjançant la seva selecció amb el *checkbox* si són més d'un o simplement seleccionat amb el cursor quedant aquest de color blau) tal com indiquen els botons (de la mateixa manera que a la pestanya de la llista, es veuen segons si s'ha seleccionat un fitxer o no):

Eliminar els elements seleccionats. Si es vol tornar a disposar d'aquests fitxers, s'ha de tornar a enviar de nou amb el procés explicat anteriorment.

Afegir a la llista de reproducció.

Reproduir després. No afegeix el fitxer a la llista i serveix per mostrar puntualment un fitxer. En cas de voler-lo afegir, cal prémer el botó "Afegir a la llista de reproducció"

En els dos casos anteriors, si es tracta d'un fitxer estàtic, ens mostrarà la finestra emergent de la figura 9.4 per tal de demanar el temps de visualització:

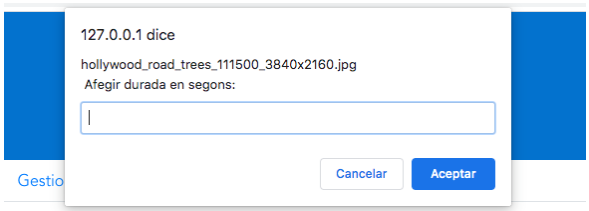


Figura 9.4: Captura de pantalla de la interfície d'usuari final. Finestra emergent duració de la visualització
(Font: Elaboració pròpia)

En cas de deixar-ho buit, el sistema afegeix per defecte 6 segons

Gestió d'usuaris

En cas de ser administradors, podem veure una pestanya més que els usuaris normals: Gestionar usuaris

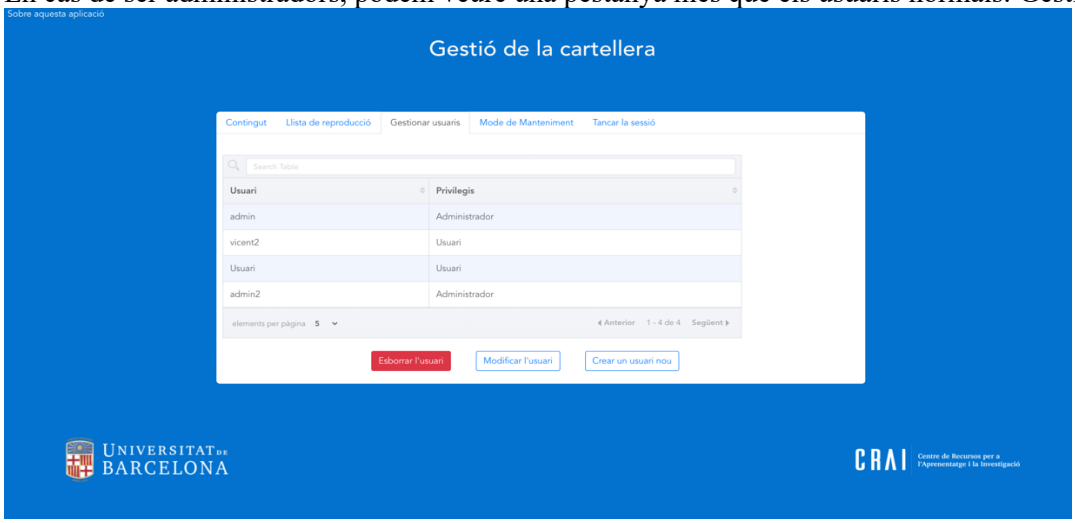


Figura 9.11: Captura de pantalla de la interfície d'usuari final. Pestanya "Gestió d'usuaris "
(Font: Elaboració pròpia)

En aquesta secció podem:
Afegir un nou usuari
Eliminar-ne (el seleccionat)
Modificar-ne (el seleccionat)

Tant en el primer com en l'últim cas, sortirà una finestra de diàleg com aquesta figura 9.12:

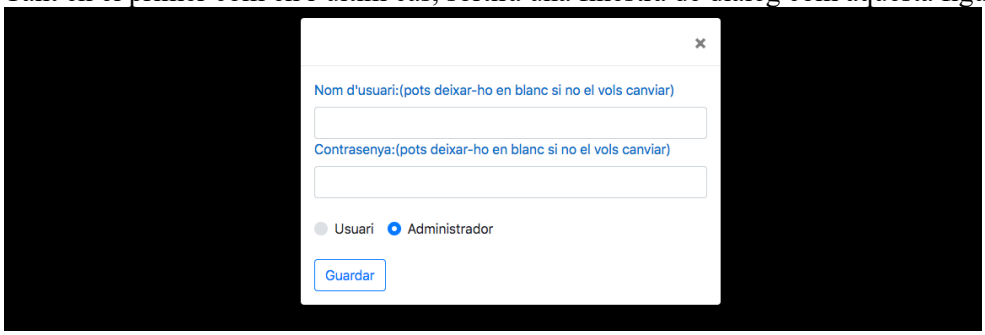


Figura 9.12: Captura de pantalla de la interfície d'usuari final. Finestra emergent modificar un usuari
(Font: Elaboració pròpia)

Amb la diferència que al crear no deixa escollir el rol. Això es perquè per defecte tots el usuaris seran sense privilegis. Si es vol canviar, s'ha de seleccionar l'usuari, prémer modificar, i canviar el rol.

Activar/Desactivar el mode manteniment

Si volem posar el mode manteniment, primer canviem de pestanya a la de “Mode manteniment”. Podem saber si està activat o no pel missatge com es veu a la següent figura 9.13:

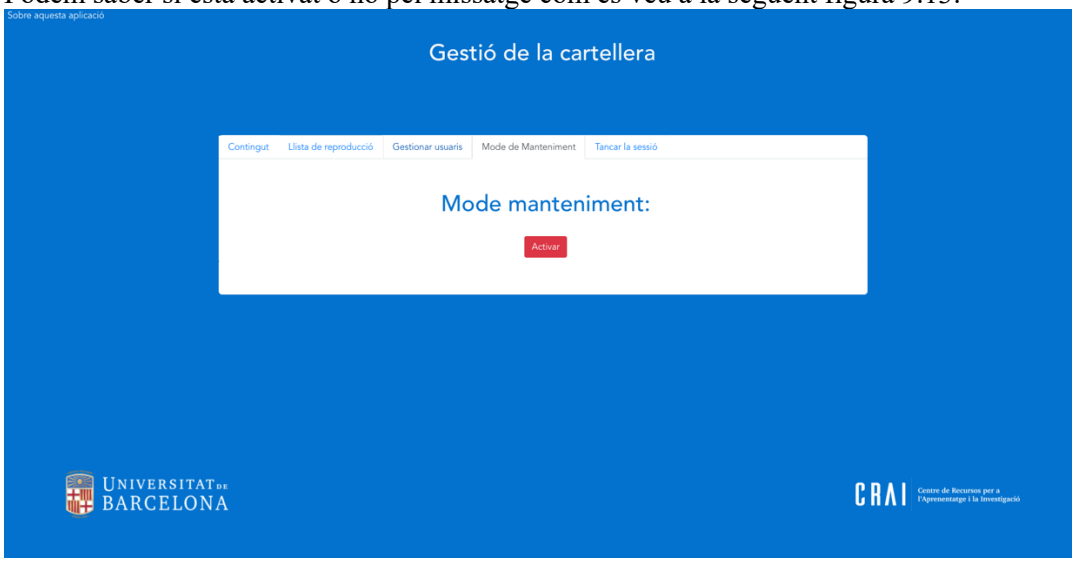


Figura 9.13: Captura de pantalla de la interfície d’usuari final. Pestanya “Mode manteniment” (Font: Elaboració pròpia)

Tancar sessió

Per tancar la sessió, només hem de canviar a la pestanya “Tancar la sessió” i prémer el botó:

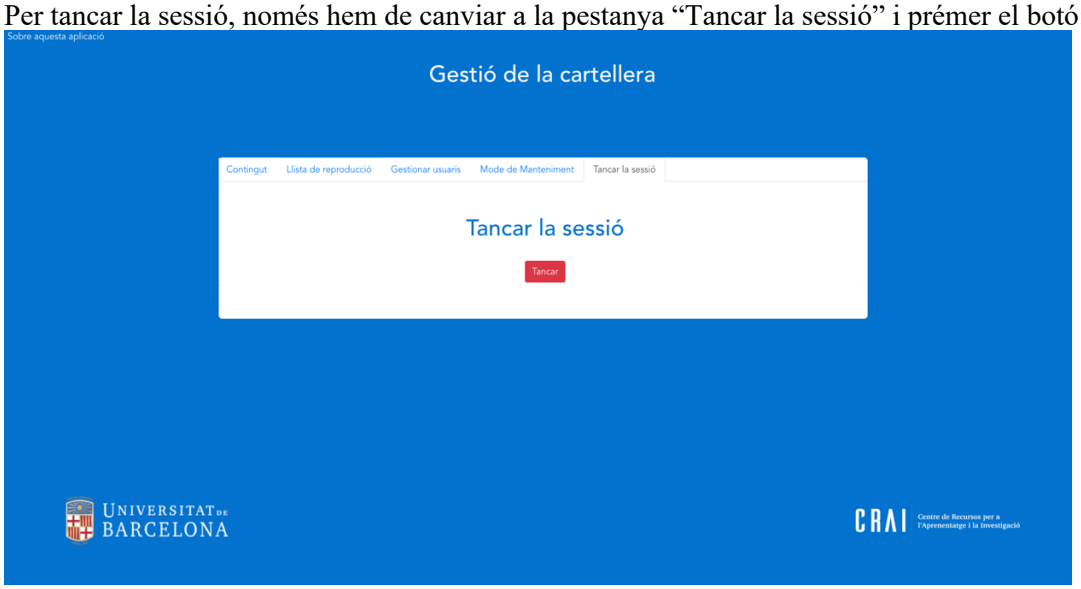


Figura 9.14: Captura de pantalla de la interfície d’usuari final. Pestanya “Tancar la sessió” (Font: Elaboració pròpia)

El sistema automàticament ens portarà a la finestra d’inici de sessió vista al principi d’aquest manual (Figura 9.1)

10.Bibliografia

Empey, Charlotte. *How to create a strong password*. Avast Blog. 15 August 2018. [Consultat el 2 de desembre de 2020]. Recuperat de https://blog.avast.com/strong-password-ideas

Fajar, Ridwab. *Build Python Web Application using Flask and Docker*. The Startup. Apr 18, 2020. [Consultat el 25 de setembre de 2020]. Recuperat de https://medium.com/swlh/build-python-web-application-using-flask-and-docker-1e38cfa12f22

Five Ways To Run a Program On Your Raspberry Pi At Startup. Dexter industries. [Consultat el 3 de gener de 2021]. Recuperat de https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/

Flask-HTTPAuth. [Consultat el 4 de desembre de 2020]. Recuperat de https://flask-httpauth.readthedocs.io/en/latest/

Flask web development, one drop at time. [Consultat el 23 de setembre de 2020]. Recuperat de https://flask.palletsprojects.com/en/1.1.x/quickstart/

Getting started. Bootstrap Vue. [Consultat el 23 d'octubre de 2020]. Recuperat de https://bootstrap-vue.org/docs

Griffith, Chris. *Uploading large files by chunking – featuring Python Flask and Dropzone.js.* Code Calamity. 31 de maig de 2018. [Consultat el 23 de desembre de 2020]. Recuperat de https://codecalamity.com/uploading-large-files-by-chunking-featuring-python-flask-and-dropzone-js/

Handling large file uploads with Flask. StackOverflow. 23 de juny de 2017. [Consultat el 13 d'octubre de 2020]. Recuperat de https://stackoverflow.com/questions/44727052/handling-large-file-uploads-with-flask

HTML Video. w3schools.com. [Consultat el 2 d'octubre de 2020]. Recuperat de https://www.w3schools.com/html/html5_video.asp

Jover Morales, Alex. *File Upload in Vue.js Using vue-dropzone*. Digitl Ocean Community. 5 d'abril de 2018. [Consultat el 6 d'octubre de 2020]. Recuperat de https://www.digitalocean.com/community/tutorials/vuejs-vue-dropzone

My mp4 won't display in flask when it did before. Stack Overflow. 6 de juliol de 2019. [Consultat el 2 d'octubre de 2020]. Recuperat de https://stackoverflow.com/questions/56915289/my-mp4-wont-display-in-flask-when-it-did-before

MoviePy. 2017. [Consultat el 2 de novembre de 2020]. Recuperat de https://zulko.github.io/moviepy/

Norman, Donald A. *The design of everyday things*. New York : Basic Books, 2013. ISBN 9780465050659.

Okoh, Michael. *How To Build and Deploy a Flask Application Using Docker on Ubuntu 18.04*- Digital Ocean Community. Originally Published onApril 2, 2019. [Consultat el 25 de setembre de 2020]. Recuperat de https://www.digitalocean.com/community/tutorials/how-to-build-and-deploy-a-flask-application-using-docker-on-ubuntu-18-04

Overview of Docker Compose. Docker docs. [Consultat el 25 de setembre de 2020]. Recuperat de https://docs.docker.com/compose/

Raghuwanshi, Monica. *Javascript Scheduling: setTimeout and setInterval*. Jan 17, 2018. [Consultat el 3 d'octubre de 2020].Recuperat de https://medium.com/@monica1109/scheduling-settimeout-and-setinterval-ca2ee50cd99f

SSH using Linux or Mac OS. [Consultat el 26 de setembre de 2020]. Recuperat de https://www.raspberrypi.org/documentation/remote-access/ssh/unix.md

Simic, Sofia. *How To Install Docker On Raspberry Pi*. December 12, 2019. Phoenixnap. [Consultat el 25 de setembre de 2020]. Recuperat de https://phoenixnap.com/kb/docker-on-raspberry-pi

Template Syntax. Vue.js. [Consultat el 23 de novembre de 2020]. Recuperat de https://vuejs.org/v2/guide/syntax.html

Use volumes. Docker docs. [Consultat el 23 de desembre de 2020]. Recuperat de https://docs.docker.com/storage/volumes/

Vue-good-table: A clean and simple VueJs (2.x) table plugin. Vue Forum. 1 de maig de 2017. [Consultat el 13 de novembre de 2020].Recuperat de https://forum.vuejs.org/t/vue-good-table-a-clean-and-simple-vuejs-2-x-table-plugin/11148

Vue good table. Xaksis. [Consultat el 13 de novembre de 2020]. Recuperat de https://xaksis.github.io/vue-good-table/