Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Disseny i implementació d'un servei d'streaming de vídeo a la carta i vídeo en directe

Document: Memòria

Alumne: Roger Bassons Renart

Tutor: Antonio Bueno Delgado / Lluís Fàbrega Soler
Departament: Arquitectura i Tecnologia de Computadors
Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any) Juny/2017

Índex

1 Introducció, motivacions, propòsit i objectius del projecte

1.1 Introducció

Actualment els serveis d'streaming de vídeo a la carta ("on demand vídeo streaming") i de vídeo en directe ("live video streaming") són molt populars. Aquest tipus de serveis permeten als usuaris compartir vídeos a través de la xarxa d'una manera intuïtiva i simple.

L'streaming de vídeo requereix una gran infraestructura de xarxa al darrera per a poder servir contingut arreu del món amb una qualitat de servei homogènia. Alguns serveis comercials com Youtube, Netflix i Twitch varen sorgir a voltants de 2005-2011 i des de llavors la base d'usuaris no ha parat d'incrementar-se. Actualment estem en un punt on l'streaming de vídeo està substituint la televisió convencional com a principal mètode de consum de vídeo.

1.2 Motivacions, propòsit i objectius

Serveis comercials com Youtube, Netflix i Twitch ofereixen una gran varietat de funcionalitats que els diferencia de la resta. Aquests serveis porten anys d'evolució al darrera. Raó per la qual el servei ofert és molt robust i capaç de servir a una gran quantitat d'usuaris simultàniament arreu del món. Hi ha ocasions, però, on els usuaris requereixen tenir un control més precís sobre el contingut per assegurar-ne la privacitat i la seguretat. Per aconseguir-ho necessiten disposar de la seva pròpia infraestructura. A més, d'aquesta manera, no s'està subjecte als Termes i Condicions del Servei aliens i es pot oferir una interfície que s'ajusti a les necessitats dels usuaris. Un exemple d'aquesta necessitat podria ser una empresa que difon vídeos, a la carta o en directe, als treballadors o clients per motius de formació. Aquests vídeos podrien contenir informació sensible o confidencial amb certs requeriments legals. Per poder assegurar la seguretat dels vídeos l'empresa pot implementar un servei d'streaming adaptat a les seves necessitats.

És per això, que aquest projecte no té intenció d'oferir un servei millor o competir amb els serveis comercials, ja que és un mercat amb requeriments molt alts i difícil d'entrar des d'un punt de vista tècnic i econòmic. Tanmateix, aquest projecte complementa aquests serveis en sectors on és important assegurar el control del contingut.

L'objectiu és dissenyar i implementar un servei d'streaming que ofereixi vídeos a la carta i vídeos en directe. Per tal d'assolir aquest objectiu és necessari crear un servidor i un client web que implementi el servei i compleixi les següents característiques:

- Llistat i cerca de vídeos
- Reproducció de vídeos a la carta i vídeos en directe
- Interfície simple i intuïtiva
- Permetre pujar vídeos als usuaris
- Permetre comentar vídeos als usuaris

1.3 Estructura de la documentació

A continuació s'exposa l'estructura d'aquest document

1. Introducció, motivacions, propòsit i objectius del projecte: Situa el marc del projecte, defineix les raons per la qual s'ha escollit aquest projecte i defineix els objectius.
2. Estudi de viabilitat: Justifica els paràmetres necessaris per a dur a terme el projecte
3. Metodologia: S'exposa la metodologia que s'ha seguit alhora d'implementar el sistema informàtic.
4. Planificació: Es defineix l'estratègia seguida per arribar als objectius.

5. Marc de treball i conceptes previs: Aquí s'exposaran una sèrie de conceptes necessaris per fer un bon seguiment del projecte

6. Requisits del sistema: Els requisits que ha de complir el sistema, tant funcionals com no funcionals.

7. Estudis i decisions: En aquest apartat es descriu el maquinari, les llibreries o programari utilitzats durant el desenvolupament del projecte amb la justificació de cada elecció.

8. Anàlisi i disseny del sistema: S'exposa un anàlisi és de les necessitats del sistema i el disseny que les soluciona.

9. Implementació i proves: Es detalla el procés d'implementació juntament amb els problemes i solucions que han aparegut.

10. Implantació i resultats: Detalla el procés requerit per implantar el sistema desenvolupat i els resultats obtinguts.

11. Conclusions: Conclusió del projecte i dels resultats.

12. Treball futur: Possibles ampliacions, millores o treballs futurs que es poden realitzar.

13. Bibliografia: Referències utilitzades per desenvolupar el projecte.

14. Annexos: Conté explicacions que no són indispensables per la comprensió global del projecte.

15. Manual d'instal·lació: Especifica les passes a seguir per instal·lar el projecte en un ordinador.

2 Estudi de viabilitat

2.1 Viabilitat tècnica

Perquè aquest projecte es pugui realitzar, fan falta llibreries i eines que serveixin per desenvolupar serveis web. Abans de començar s'ha analitzat que existeixen totes les llibreries i eines d'ús gratuït necessàries per tal de dur a terme el projecte.

2.2 Viabilitat econòmica

El projecte l'implementaré jo mateix utilitzant software amb llicències de codi lliure i el desplegament de l'aplicació no es durà a terme en un servidor amb cost. A més, el projecte es farà disponible com a codi lliure amb una llicència compatible amb les llicències del software utilitzat.

Per tant, econòmicament aquest projecte no suposa cap inversió o despesa.

2.2.1 Pressupost

Tot i què el projecte no suposarà cap despesa, a continuació es fa una estimació del pressupost necessari en el cas de que s'hagués de contractar a programadors. Un projecte com aquest es pot implementar en un mes utilitzant 1 programador a temps complet i, si es desitja que la web tingui un bon disseny gràfic, faria falta contractar un dissenyador per que faci un disseny personalitzat i a mida per la web. A la taula 1 es pot observar la estimació del cost econòmic:

Taula 1: Pressupost

	€/h	Hores totals	Cost (€)
Programador	13	160	2080
Dissenyador	11	40	440
Total			2520

3 Metodologia

La metodologia és un procés que defineix com s'organitza i es separa la feina alhora de desenvolupar una aplicació.

La metodologia que he utilitzat és la iterativa i incremental que consisteix en desenvolupar la aplicació en una sèrie d'iteracions. A cada iteració s'afegeix noves funcionalitats i el producte resultant de la iteració és un prototip. El prototip és l'aplicació amb amb un subconjunt de les totes les funcionalitats a implementar. El procés que he seguit és el següent:

1. Recollir i estudiar els requisits.
2. Estudi dels diferents frameworks i llibreries.
3. Escollir frameworks i llibreries a utilitzar.
4. Disseny general del sistema d'acord amb els requisits recollits.
5. Iteració: Desenvolupament d'un conjunt de funcionalitats.
6. Si durant el desenvolupament fa falta canviar el disseny tornem al pas 4.
7. Prova del que s'ha desenvolupat. Si no es supera la prova o no es compleixen els requisits és torna enrere per solucionar l'última iteració de desenvolupament altrament es passa a la següent iteració.

El procés iteratiu continua mentre no es compleixen tots els requisits. La documentació del projecte s'ha anat fent a mesura que s'ha desenvolupat el projecte i al final de tot s'ha comprovat que la documentació era vàlida i actualitzada.

4 Planificació

La proposta del projecte és va fer el Febrer del 2017 però la idea general del projecte ja la tenia pensada des del Desembre del 2016. La data proposada d'entrega és per el Juny del mateix any i, per tant, el projecte té una duració aproximada d'uns 5 mesos.

La planificació del projecte és defineix mitjançant 4 blocs:

- Anàlisi de requeriments
- Disseny
- Implementació
- Documentació

Aquests blocs representen la feina a fer en aquest mateix ordre. La documentació és una excepció de l'ordre ja que s'ha de fer mentre es va desenvolupant el projecte. Cada bloc, a més, està format per altres petites tasques i la previsió temporal de cada tasca del projecte està representada al diagrama de Gantt de la figura 1.

Figura 1: Diagrama de Gantt

5 Marc de treball i conceptes previs

5.1 Model TCP/IP

Per poder comprendre els protocols de streaming es necessari conèixer com està estructurada la xarxa. La xarxa està dividia en capes on cada capa és una abstracció de funcionalitats de xarxa. Al llarg de la història de la Internet s'han definit diversos models que defineixen el nombre de capes i les funcionalitats de cada una. Però, la especificació oficial suportada per la IETF (Internet Engineering Task Force) que és coneix amb el nom TCP/IP és l'estàndard.

Aquest model consta de 4 capes:

- Capa 4: Aplicació, Serveis de intercanvi d'informació
- Capa 3: Transport, Comunicació extrem a extrem
- Capa 2: Interxarxa, Transport de paquets a través de múltiples xarxes
- Capa 1: Xarxa, Transport de paquets en una xarxa local

Alhora de servir contingut multimèdia les capes que influeixen més sobre el rendiment i, per tant, l'experiència de l'usuari són la capa 3 i 4. L'elecció de la combinació de protocols de les dues capes defineix com s'envia el contingut i estableix el retard i el temps de resposta del servei multimèdia.

5.2 Model client-servidor

La manera més utilitzada per organitzar aplicacions distribuïdes és utilitzant el model Client-Servidor. Aquest model consisteix en repartir les tasques de l'aplicació entre el Servidor i el Client. El servidor proveeix algun servei i el client envia peticions al servidor per utilitzar el servei esmentat. L'ús més habitual d'aquest model consisteix en un servidor i múltiples clients. El servidor pot ser local i executar-se al mateix sistema que els clients però

normalment és remot i s'accedeix a través de la xarxa. El servidor sol ser concurrent ja que pot executar un o més programes servidor. Els clients inicien peticions dirigides al servidor per tal d'utilitzar un servei en particular i la petició té associada informació per conèixer la operació i objecte que està demanant al servidor.

La comunicació entre el servidor i els clients és realitza amb Web Services.

5.3 Web services

El terme Web Services és molt utilitzat però sovint amb significats diferents. Existeixen definicions que varien des de molt genèriques a molt especifiques. La definició del W3C és: "a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols".

Per tant, es pot dir que els Web Services han d'estar definits, descrits i disponibles. Aquests requisits impliquen oferir una interfície accessible i la qual és utilitzada per interactuar amb la aplicació d'una manera ben definida.

5.4 API REST

La API REST és conjunt de criteris per dissenyar la arquitectura que estableix la comunicació entre els clients i el servidor. Aquesta consisteix en construir un RESTful Web Service. REST no depèn de cap protocol però en la gran majoria de casos s'utilitza el protocol HTTP. El servei RESTful es construeix al voltant dels recursos. Un recurs és qualsevol cosa o objecte que tingui sentit per si sol i és pugui referenciar com per exemple una imatge, document, resultat d'un algorisme, etc. Per identificar els recursos s'utilitzen les URI i corresponen al nom únic del recurs.

Les característiques generals del servei REST són:

- Representació: els objectes es representen com recursos.
- URIs: identificador d'un recurs en concret o un conjunt de recursos.
- Missatges: comunicació per obtenir o utilitzar un recurs. Normalment s'utilitza el protocol HTTP amb els verbs GET, PUT, UPDATE, DELETE.
- Stateless: Tots els missatges de petició-resposta succeeixen amb isolació. No fa falta cap context per realitzar un missatge de petició.
- Caching: Permet emmagatzemar de forma temporal la resposta d'un missatge si el recurs no ha canviat. D'aquesta manera s'eviten comunicacions innecessàries entre el servidor i el client i s'estalvia amplada de banda.

5.5 Pàgines web estàtiques & dinàmiques

Les pàgines web poden ser estàtiques o dinàmiques. En el primer cas el contingut que és presenta a tots els usuaris és exactament el mateix independentment del context. Les pàgines web estàtiques són útils quan el contingut de la pàgina no canvia gaire sovint. La experiència d'usuari pot ser interactiva si es combinen diverses pàgines estàtiques. Les pàgines dinàmiques, en canvi, permeten canviar dinàmicament el contingut de la mateixa pàgina. L'avantatge principal de les pàgines dinàmiques és que permet desenvolupar pàgines web de manera més fàcil i escalable. En certs casos la diferència entre estàtic i dinàmic és difícil de distingir com per exemple generadors de pàgines web estàtiques com JekyII.

5.6 Server-rendering & Client-rendering

A les pàgines web dinàmiques la generació del contingut dinàmic és pot fer al servidor o al client.

5.6.1 Server Rendering

És la manera tradicional i consisteix en que el servidor rep la petició del client i s'encarrega de construir la pàgina web d'acord amb la petició. A continuació, el servidor respon amb un document HTML complet i preparat per a que el navegador del client ja el pugui mostrar directament sense haver de fer càlculs o processament extra.

5.6.2 Client Rendering

La manera "moderna" de desenvolupar webs és utilitzant client rendering. Utilitzant client scripting amb llenguatges com el Javascript la pàgina web es converteix en una aplicació que per si sola pot generar contingut. El client utilitza l'api rest que el servidor proporcionar per obtenir la informació necessària per construir la web i un cop rebuda la mostra a l'usuari.

5.7 Single Page Application

Una SPA és una aplicació web que carrega un únic document HTML i la pàgina s'actualitza a mesura que l'usuari interacciona sense utilitzar altres documents HTML. El contingut es pot obtenir utilitzant crides rest a un servidor. En cap moment es recarrega la pàgina o es carrega una altra pàgina i l'experiència d'usuari pot ser semblant a les aplicacions tradicionals amb múltiples pàgines únicament que s'aprofiten els avantatges de SPA.

5.8 Multiple Page Application

Una MPA és una aplicació web en la qual, cada vegada que s'ha de produir un canvi, el client fa una petició al servidor i el servidor és l'encarregat de renderitzar la pàgina web i retronar-la al client.

5.9 Search Engine Optimitzation

El SEO consisteix en afectar la visibilitat d'un lloc web a un motor de cerca com Google, Yahoo, Bing, etc. L'objectiu és aparèixer el més amunt possible en cerques relacionades amb el contingut del web.

5.10 Vídeo

5.10.1 Compressió

Considerant que s'ha enregistrat vídeo amb les característiques següents:

- Megapixels: 2.1 MP (2,073,600 pixels)
- Aspect ratio: 1.78
- RAW size of 1 frame (Bayer masked): 3.11 MB (2.1 MP x 12 bits)
- Uncompressed RGB 8-bit frame: 6.22 MB (2.1 MP x 24 bits)
- Uncompressed RGB 16-bit frame: 12.4 MB (2.1 MP x 48 bits)

El vídeo no comprimit (raw) és una font de dades molt gran. Per exemple un vídeo de 1920x1080 a 25fps i amb profunditat de color de 8 bits amb una duració de 120 min representa 1.12TB de mida i per poder reproduir-lo fa falta assolir 1.24Gbps. És obvi que no sempre es disposa d'aquests recursos. Per tal de reduir la mida del vídeo, aquest es comprimeix. La compressió del vídeo s'aconsegueix aprofitant la redundància del vídeo. Per exemple donats diversos frames consecutius d'un vídeo és possible que tinguin moltes similituds ja que pot ser surten els mateixos objectes. Un altre fenomen que s'aprofita per comprimir el vídeo és la irrellevància del senyal de vídeo. Consisteix en informació del vídeo que no és rellevant per l'ull humà. L'espai que ocupa aquesta informació es pot aprofitar per emmagatzemar informació que pugui ser percebuda per l'ull humà (informació útil).

Per tal de facilitar la cooperació entre aplicacions i entre els desenvolupadors s'han establert una sèrie d'estàndards al llarg dels anys: MPEG-1, MPEG-2, MPEG-4, H.264, VP8, HEVC (H.265), VP9. Cada un dels estàndards defineix com ha de ser la sortida de la codificació d'un vídeo. El més utilitzat actualment és l'H.264 i el successor és l'HEVC més conegut com H.265.

5.10.2 Contenidors

El vídeo codificat (comprimit) s'emmagatzema dins d'un contenidor. El contenidor té el vídeo codificat juntament amb l'àudio codificat. El contenidor a més pot contenir informació de sincronització, subtítols i metadades. Alguns contenidors populars són mp4, webm, avi i mkv.

5.11 Streaming de vídeo

Existeixen dues maneres de transmetre vídeo per Internet: el mode descàrrega i el mode streaming.

En el mode descàrrega, l'usuari descàrrega el vídeo i posteriorment el reprodueix ja que és necessari el fitxer complet per poder descodificar el vídeo. En aquest mode també es pot fer el que és coneix amb el nom de descàrrega progressiva que consisteix en codificar el vídeo de manera que la informació necessària per reproduir-lo estigui al principi i, per tant, es pugui anar reproduint a mesura que es va descarregant. A dia d'avui aquest mode no s'utilitza per reproduir vídeo per Internet ja que descarregar el vídeo complet suposaria un temps d'espera inacceptable per l'usuari i tot i que el mode progressiu ho soluciona aquest no ofereix una manera d'adaptar-se a l'amplada de banda de l'usuari.

En el mode streaming no és necessari descarregar el vídeo complet ja que consisteix en reproduir el vídeo a mesura que es reben i descodifiquen petites parts del vídeo.

Els avantatges del vídeo streaming sobre el mode descàrrega són evidents i és normal que el mode streaming sigui el més utilitzat.

El ràpid creixement de l'streaming va ser impulsat per:

- Avanços en algoritmes de compressió d'àudio i vídeo
- Desenvolupament de servidors de streaming
- Millora de la banda ampla de la xarxa

L'streaming de vídeo en directe és un cas particular de l'streaming de vídeo en que el vídeo s'enregistra, codifica i envia en temps real a mesura que la font captura el vídeo.

5.12 Arquitectura Streaming

A continuació hi ha un breu resum de l'arquitectura streaming, es a dir, que fa falta per que funcioni l'streaming.

5.12.1 Compressió de vídeo

El vídeo raw és comprimeix per facilitar una transmissió més eficient. La compressió es realitza utilitzant la tècnica de codificació de vídeo escalable. Aquesta tècnica és capaç de suportar les fluctuacions de l'amplada de banda de la xarxa.

5.12.2 Control de qualitat (Capa aplicació)

Per compensar l'ampli rang de condicions de xarxa i qualitat de presentació dels usuaris es poden utilitzar varies tècniques de control a nivell d'aplicació. Entre aquestes tècniques s'inclou control de congestió i control d'error.

5.12.3 Serveis de distribució de continguts multimèdia continus

Per poder proveir una presentació del contingut d'alta qualitat es requereix un suport de xarxa. Aquest ha de reduir el retard i el percentatge de paquets perduts. Implementats a sobre de IP, alguns d'aquests serveis són el network filtering, l'application-level multicast i el content replication.

5.12.4 Servidors d'streaming

Els servidors d'streaming són una peça crucial alhora de proveir serveis d'streaming. El servidors processen les dades multimèdia amb limitacions temporals i suporten operacions de control interactives. A més, els clients han d'obtenir els mitjans de manera síncrona (àudio i vídeo).

5.12.5 Mecanismes de sincronització multimèdia

La sincronització multimèdia és una funcionalitat que diferència les aplicacions multimèdia d'altres aplicacions de dades. Aquests mecanismes permeten que el receptor visualitzi els mitjans multimèdia tal com es van capturar originalment. Per exemple, que l'àudio i el vídeo estiguin sincronitzats i, per tant, els llavis d'una persona que parla estiguin igualats amb l'àudio.

5.12.6 Protocols de streaming

Els protocols estan dissenyats per oferir la comunicació entre els servidors d'streaming i els clients. Els protocols aporten serveis com network adressing, transport i control de sessió.

5.13 Problemes de l'streaming de vídeo per Internet

L'streaming de vídeo pot arribar a ser difícil ja que la Internet actual només ofereix un servei "best-effort". Conseqüentment no hi ha cap garantia de

rendiment i fiabilitat. L'objectiu de l'streaming de vídeo és dissenyar un sistema per poder entregar vídeo d'alta qualitat tractant amb les següents variables de magnituds desconegudes i dinàmiques:

- Amplada de banda
- Variació del retard
- Perdues
- Retard mig

L'amplada de banda disponible entre dos punts a la Internet sol ser desconeguda i variable en el temps. En el cas que es transmeti a més velocitat de la banda ampla disponible llavors es congestiona la xarxa, es perden paquets i, per tant, disminueix la qualitat del vídeo transmès. L'objectiu, doncs, és estimar la amplada de banda disponible per no transmetre el vídeo a una velocitat superior.

El retard que experimenta cada paquet és diferent. Aquesta variació entre retards s'anomena jitter. Aquest fenomen és un problema ja que el receptor ha de rebre, descodificar i reproduir el vídeo a una velocitat de fotogrames constant. Qualsevol retard alhora de rebre un fotograma que sigui originat pel jitter pot produir problemes per reconstruir el vídeo. Una solució per minimitzar els efectes del jitter és utilitzar un buffer al receptor. Aquest pot compensar el jitter però afegeix un retard considerable.

Les pèrdues són comunes a la xarxa i tenen un efecte negatiu a la qualitat del vídeo reconstruït. Per combatre les pèrdues, l'streaming de vídeo està dissenyat amb un control d'error. Existeixen varies maneres de realitzar el control d'error:

- FEC
- Retransmetre
- Codificació de vídeo error-resilient

5.14 Protocols d'streaming

5.14.1 RTP

Real-Time Protocol és un protocol a nivell d'aplicació que ofereix transport d'àudio i vídeo end-to-end i en temps real. Aquest s'utilitza conjuntament amb el protocol RTCP, el qual s'encarrega d'especificar el QoS i la sincronització entre els fluxos multimèdia. El protocol RTP és el que transporta les dades. Aquest protocol és utilitzat sobre UDP, tot i que també funciona per sobre de TCP. Normalment no s'utilitza sobre TCP perquè afegeix retard al oferir un servei fiable i ordenat.

5.14.2 HTTP

El protocol HTTP és la manera més simple de fer streaming de vídeo ja que s'aprofita la estructura de la Internet ja muntada i en funcionament. Aquest protocol només requereix un servidor web i, a més, al utilitzar el port 80 és menys probable que sigui bloquejat pels firewalls. Per sobre de HTTP existeixen molts protocols d'streaming però els més coneguts són MPEG-DASH, HLS, HDS i Smooth.

5.14.3 MPEG-DASH i HLS

L'MPEG-DASH i HLS són protocols que utilitzen la tècnica d'Adaptative bitrate streaming. Per tal d'aconseguir-ho parteixen el vídeo en segments i cada segment està disponible en diferents bit rates.

Tots dos tenen un fitxer índex que conté tota la informació sobre els segments. Aquesta informació és necessària perquè els clients puguin reproduir el vídeo i adaptar el bitrate a l'amplada de banda. El DASH té un fitxer MPD (Media presentation description) que conté tota la informació sobre els segments i els segments poden ser codificats amb qualsevol codec. L'HLS conté aquesta informació al fitxer m3u8 però els segments han d'estar codificats en MPEG-2 TS.

5.14.4 RTMP

Real-Time Messaging Protocol (RTMP) era inicialment un protocol propietari utilitzat per transmetre dades entre un reproductor Flash i el servidor. Aquest protocol funciona sobre TCP i permet connexions persistents de baixa latència.

5.15 Adaptive bitrate streaming

L'Adaptive bitrate streaming és una tècnica de streaming que, mitjançant la amplada de banda disponible de l'usuari, ajusta la qualitat del vídeo per tal de proporcionar poc buffering i una bona experiència d'usuari. Per tant, s'ajusta la velocitat de transmissió del vídeo i la seva qualitat a la situació de la xarxa: si hi ha massa pèrdues o massa retard/jitter, es baixa la velocitat/qualitat, i si la cosa millora, es puja. L'objectiu és proporcionar la millor qualitat d'experiència a l'usuari. Aquesta tècnica, en els protocols com HLS i MPEG-DASH la realitza exclusivament el client ja que, mitjançant un fitxer índex, coneix com està segmentat el vídeo i en quines qualitats.

5.16 CDN

Una CDN (Content Distribution Network) és una xarxa especialitzada en lliurament de contingut. En una xarxa de lliurament de continguts, el proveïdor té fitxers o contingut multimèdia que desitja que estiguin disponibles per descarregar per usuaris geogràficament distribuïts. Una CDN, doncs, proveeix els fitxers a múltiples servidors a través de la xarxa perquè la majoria d'usuaris tinguin una proximitat al servidor de descàrrega o streaming el més propera possible. Aquest fet afavoreix una latència menor i una capacitat major que la del servidor original, on realment els fitxers estan emmagatzemats. El principal problema dels CDNs és com realitzar la distribució dels fitxers i decidir des de quin servidor s'ha de descarregar el contingut un client determinat. Existeixen varies estructures per solucionar aquest problema.

Una CDN és una col·lecció d'elements de xarxa disposats d'una manera de-

terminada per entregar continguts als usuaris eficaçment. La estructura pot ser centralitzada, jeràrquica o completament descentralitzada. Com s'interconnecten i interactuen els nodes d'aquesta estructura de la CDN també pot prendre diverses formes. Tot i això, en general les funcionalitats d'una CDN inclou:

- Redirecció de peticions i serveis de distribució: Redirigir/Delegar una petició al millor servidor més proper.
- Outsourcing de contingut i de serveis de distribució: Replicar/Cachejar contingut cap als servidors respecte servidors origen.
- Serveis de negociació de continguts: Complir necessitats d'usuaris o grups d'usuaris en concret.
- Serveis de monitoratge: gestionar els components de xarxa i monitorar l'ús de continguts i rendiment de servidors.

5.16.1 Com s'utilitza una CDN

Per utilitzar una CDN, un cop contractat el servei, s'ha de enviar peticions a la CDN amb els fitxers que s'han de distribuir. Un cop s'ha fet a petició la CDN és la encarregada de distribuir el contingut als servidors d'arreu del món que s'hagin contractat.

6 Requisits del sistema

Alhora de definir els requisits del sistema es diferencia en Requisits funcionals i Requisits no funcionals. Els requisits funcionals són els que defineixen o descriuen una funcionalitat del sistema. Els requisits no funcionals no defineixen cap funcionalitat però són clau per la operació del sistema. Dit d'una manera més senzilla els requisits funcionals defineixen que ha de fer el sistema (disseny) i els requisits no funcionals defineixen com a de ser el sistema (arquitectura).

6.1 Requisits funcionals

L'aplicació pot ser utilitzada per un usuari no identificat o un usuari identificat.

6.1.1 Usuari no identificat (anònim)

- Identificar-se (login)
- Visualitzar un llistat dels últims vídeos penjats
- Visualitzar un llistat dels streams en directe que s'estan transmetent
- Veure un vídeo en concret
- Veure un stream en directe
- Buscar vídeos per paraules contingudes al títol
- Veure qui ha penjat el vídeo o stream
- Veure la data que es va penjar el vídeo o stream
- Veure la descripció del vídeo o stream
- Veure el número de persones que han vist un vídeo

- Veure els likes d'un vídeo
- Veure els unlikes d'un vídeo
- Veure els comentaris d'un vídeo

6.1.2 Usuari identificat (registrat)

L'usuari identificat té un nom d'usuari i contrasenya que l'identifiquen. Pot fer el mateix que l'usuari identificat a excepció de la funcionalitat de login. A més pot fer:

- Sortir (logout)
- Comentar un vídeo múltiples vegades
- Fer like a un vídeo
- Fer unlike a un vídeo
- Pujar un vídeo
- Emetre un stream de vídeo en directe

6.2 Requisits no funcionals

- Disposar de connexió internet suficientment ràpida
- Disposar d'un navegador web amb suport de Javascript ES5 i HTML5
- El navegador web ha de tenir el Javascript habilitat
- S'ha fer correctament el desplegament del servidor (servidor web i base de dades)
- És necessari un dispositiu capaç de reproduir vídeo HD

7 Estudis i decisions

7.1 Arquitectura de l'aplicació

Abans d'escollir quines llibreries utilitzar pel servidor i per el client s'ha de decidir com ha de ser l'arquitectura de tota la aplicació. Tal i com s'ha vist a l'apartat de marc de treball i conceptes previs existeixen diverses maneres de desenvolupar una aplicació web. Les decisions que s'han de prendre des d'un punt de vista d'arquitectura d'aplicació es detallen a continuació.

7.2 SPA (Single Page Application) vs MPA (Multiple Page Application)

7.2.1 Avantatges de SPA enfront a MPA

- Més ràpid i receptiu: La majoria de recursos com l'HTML, CSS i Javascript es carrega només un cop i la resta d'informació que es transmet durant la utilització d e l'aplicació són dades d'informació útil. Això implica que un cop carregada l'aplicació l'usuari experiència una interfície més fluida i ràpida.
- Desacoblament del frontend i el backend: En el desenvolupament del servidor no s'ha de escriure codi de la interfície d'usuari. El client i el servidor és poden desenvolupar per separats i la única cosa que els uneix és la API Rest. Si es fa correctament el manteniment de l'aplicació i el codi resultant pot ser més simple.
- Cache: El client pot guardar cache al localStorage del navegador per estalviar ample de banda i temps d'espera. D'aquesta manera el client pot arribar a treballar offline i en el moment que disposi de connexió a Internet els canvis que ha fet online es poden enviar al servidor.
- Reusabilitat: És pot reutilitzar el backend per a diversos clients (web, Android, iOS, ...).

7.2.2 Desavantatges de SPA enfront a MPA

- SEO: Difícil de fer optimització per a motors de cerca (Search Engine Optimitzation). Com que no hi han diverses pàgines on navegar, els robots de cerca no poden indexar tota la pàgina web. Per solucionar-ho es pot utilitzar tècniques que permeten navegar per la web utilitzant urls concretes com si es tractés d'una MPA.
- Carrega lenta: El temps d'espera inicial pot ser molt alt ja que s'ha de descarregar i carregar tota l'aplicació al client (Javascript).
- Javascript: Requereix que el navegador suporti el Javascript i que no estigui deshabilitat per l'usuari.
- Seguretat: És potencialment menys segur que una MPA ja que s'utilitza Javascript al navegador de l'usuari i això facilita atacs d'injecció de codi com per exemple Cross-Site Scripting (XSS).

L'aplicació és desenvoluparà amb el patró SPA pels avantatges esmentats. Concretament els avantatges que ens resulten més útil són:

- Desviar una part de la feina del servidor cap als clients (Client Rendering) per estalviar l'ús de xarxa i de cpu.
- Reusabilitat: Inicialment es desenvoluparà una aplicació web però utilitzant SPA ens força a implementar una API Rest que més endavant podria servir per desenvolupar clients natius a plataformes mòbil com per exemple Android i iOS.

Per tant, la aplicació web es desenvoluparà com una SPA utilitzant exclusivament Client Rendering.

7.3 Frameworks

Actualment existeixen molts frameworks web escrits en llenguatges molt diversos. És molt fàcil que ens trobem saturats d'informació alhora d'escollir-ne

un o trobar-nos forçats a utilitzar-ne un en particular perquè molta gent l'utilitza tot i que potser no és la millor elecció per nosaltres. Per decidir quin framework utilitzar és important tenir en compte la facilitat d'ús, corba d'aprenentatge, llenguatge utilitzat, funcionalitats que ofereixen, escalabilitat i el grau de flexibilitat i llibertat que proporcionen. A continuació hi ha una breu comparació dels frameworks més populars.

7.3.1 Client

7.3.1.1 AngularJS

- Fet i mantingut per Google
- Primera versió treta el 2010
- 55902 estrelles a Github
- Implementa el patró MVC
- Framework molt complet que vol facilitar la bona estructuració de codi per a applicacions web SPA complexes Segons l'equip que el manté l'aniran actualitzant i corregint errors mentre tingui popularitat

7.3.1.2 Angular

- Reescriptura completa des de zero de l'AngularJS
- Incompatible amb AngularJS
- Incorpora tot un seguit de millores que creuen que serà més fàcil pels programadors crear aplicacions complexes

7.3.1.3 Backbone.js

- Primera versió treta el 2010

- 26340 estrelles al Github
- molt lleugera i amb una única dependència obligatòria (Underscore.js)
- Basada en el patró MVP
- Molt flexible i poca estructura fixada

7.3.1.4 React

- Creat per Facebook
- Primera versió el 2013
- 67495 estrelles al Github
- Només és la V (View) del patró MVC i es pot utilitzar amb qualsevol altre framework web
- Proporciona una manera inovadora de construir interfícies sense utilitzar Templates
- Permet construir aplicacions web grans
- Es treballa sobre un DOM virtual
- Els objectius principals són ser ràpid, simple i escalable
- Per poder tenir "contenidors" de l'estat de l'aplicació es poden utilitzar llibreries com el Flux o Redux. Podria considerar-se com l'equivalent a la M del patró MVC.

Abans de decidir quin framework utilitzar, l'AngularJS, tot i que és un bon framework, queda descartat degut a que no és coneix fins quan es donarà suport ja que quan es comença un projecte és molt important el suport de la tecnologia utilitzada. En qüestió de poc temps podria quedar obsoleta i tot l'esforç per crear l'aplicació seria malgastat ja que l'Angular (reescriptura) no és compatible. Per tant les valoracions de la resta de frameworks es descriuen a continuació.

El Backbone és molt interessant ja que ocupa poc espai. No força un estructura d'aplicació i això vol dir que s'ha de dissenyar una bona base perquè sigui fàcil afegir canvis a mesura que l'aplicació vagi creixent. El fet de que proporcioni tanta llibertat al desenvolupador pot ser perillós ja que si no es construeix correctament des del bon principi cada vegada que s'afegeixi un canvi pot suposar canviar grans parts de l'aplicació.

L'Angular és una millora de l'AngularJS on s'introdueixen un munt de canvis. És una evolució de l'AngularJS el qual té molta documentació i, per tant, és un framework que ofereix estabilitat i escabilitat. El punt en contra és que, degut a la incompatibilitat amb l'anterior, molts han aprofitat per canviar el framework i encara no gaudeix de la mateixa popularitat que l'AngularJS.

El React és un projecte que representa un canvi molt gran alhora de desenvolupar webs ja que no es basa en el clàssic i ben conegut patró MVC. De fet només és la Vista d'aquest patró i es pot combinar amb qualsevol altre Framework inclosos els que s'han mencionat fins ara. És molt natural combinar-lo amb les arquitectures anomenades Flux, també creat per Facebook. Redux és la implementació més popular d'aquesta arquitectura. Segons Facebook el model MVC no és suficientment escalable per a aplicacions complexes. Defensen que es molt difícil preveure l'estat de l'aplicació. Per això aquesta arquitectura s'identifica perquè només existeix un flux de dades.

La decisió final, valorades les opcions anteriors, és la d'utilitzar React juntament amb Redux. A part de la tracció que està guanyant per la comunitat de programadors, s'ha escollit degut a que ofereix una documentació molt més simple que la resta i , per tant, és més fàcil començar un projecte i construir una base solida. A més, des del punt de vista de rendiment és el més ràpid ja que funciona modificant un DOM virtual i no real.

7.3.2 Servidor

A la part servidora com a opcions més populars per llenguatges de programació diferents són:

- Ruby on Rails (Ruby)

- Django (Python)
- ExpressJS (Javascript)

L'ExpressJS funciona sobre NodeJS, un runtime de Javascript. És molt simple i no ofereix cap mena d'estructura per si sol.

Ruby on Rails i Django són frameworks que ofereixen una estructura robusta amb les funcionalitats més comunes ja implementades (Base de dades...).

En aquest cas la opció més simple seria utilitzar ExpressJS ja que només s'ha desenvolupar una API Rest i no s'utilitzaran totes les funcionalitats d'un framework complet com Django i Ruby on Rails. Tot i això, aquests proporcionen la estructura base de l'aplicació ja feta i, per tant, s'aprofita codi estable que funciona i no s'ha de desenvolupar de zero. La decisió final per al servidor es basa en el coneixement del llenguatge de programació. En el meu cas, tinc més experiència amb Python i, per tant, s'ha decidit utilitzar Django.

7.3.3 Base de dades

La base de dades més coneguda és MySQL, actualment el fork de MySQL anomenat MariaDB és la "descendent" i comparteixen les mateixes funcionalitats i característiques. Tot i això, MariaDB té un manteniment millor ja que els desenvolupadors originals de MySQL són els que la mantenen. Una altre base de dades relacional és PostgreSQL que té com a objectiu principal complir l'estàndard SQL al 100%.

Una alternativa de base de dades no relacional bastant utilitzada és MongoDB. MongoDB no utilitza taules pre definides per emmagatzemar les dades, en comptes d'això utilitza documents JSON amb esquemes dinàmics. Els documents JSON es tradueixen sense problemes a objectes JSON.

MongoDB sembla que és més adequat per a projectes on és necessari molta flexibilitat i en aquest aspecte és el millor. Tot i això, MongoDB recomana utilitzar SQL quan es realitzen transaccions complexes.

S'utilitzarà MariaDB ja que, per sobre de tot és busca una base de dades robusta. A més, com que s'utilitzarà Django, aquest ja s'encarrega de fer la conversió de les dades a objecte de manera automàtica. Si no s'utilitzes Django podria ser que MongoDB fos la decisió més encertada ja que com a punt a destacar ens proporcionar les dades com objectes JSON la qual cosa és molt útil quan s'utilitzen frameworks com ExpressJS.

7.3.4 Protocols d'Streaming

7.3.5 Client

Alhora d'escollir un protocol d'streaming perquè els clients puguin visualitzar l'stream de vídeo existeixen dos protocols molt utilitzats: l'HLS i el DASH. Els dos funcionen seguint el mateix principi: segmentar el vídeo en diferents qualitats i el rendiment dels dos és molt semblant i difícil de diferenciar per a casos en general. Això és degut a que els dos es poden configurar per segmentar de diferent forma i amb qualitats de vídeo i àudio molt diverses. A la taula 2 es presenten les diferències més rellevants:

Funcionalitat	HLS	MPEG-DASH
Desplegament sobre servidor HTTP	✓	✓
Estàndard internacional		✓
Múltiples canals d'àudio		✓
Suport DRM	✓	✓
Independent dels codecs de vídeo i àudio		✓

Taula 2: Comparació HLS i MPEG-DASH

Els avantatges més importants que ofereix l'MPEG-DASH enfront l'HLS és que es un estàndard i que és pot utilitzar amb qualsevol codec de vídeo i àudio. Per tant, alhora d'implementar el client s'utilitzarà l'MPEG-DASH. Cal destacar que serveis com Youtube i Netflix utilitzen MPEG-DASH i que Twitch utilitza HLS.

7.3.6 Servidor

El servidor no necessita cap tipus de component extra perquè els clients pugin reproduir vídeos utilitzant els protocols anteriorment esmentats. Tot i això, el projecte també inclou permetre l'streaming de vídeo en directe. Per tant, s'ha d'escollir com es pot enviar un stream de vídeo des d'un client al servidor i que després múltiples clients puguin veure l'stream en directe (amb menys de 2 minuts de retard).

Si s'investiga que fan els serveis comercials podem observar com Youtube i Netflix utilitzen el protocol RTMP perquè el client que emet l'stream l'enviï al servidor. Posteriorment l'stream RTMP es converteix online a MPEG-DASH i HLS.

Una alternativa seria utilitzar el WebRTC, però aquest conjunt de protocols estan més orientats a fer connexions en temps real peer-to-peer. Potser amb una mica d'adaptació es podria utilitzar per realitzar la funcionalitat abans esmentada però l'RTMP és una manera molt simple de solucionar el problema i, per tant, s'ha escollit utilitzar l'RTMP.

El funcionament de l'streaming en directe, doncs, es farà amb el client emetent un stream RTMP i el servidor convertint online aquest stream en format MPEG-DASH.

7.3.7 Servidor web

Segons W3Techs el 49.4% de webs utilitza els servidor web Apache i el 34% Nginx. Aquestes estadístiques potser no són del tot encertades ja que una web pot utilitzar diversos servidors web al mateix temps. Tot i això, són un bon indicador que aquests dos dominen el mercat.

En termes generals Apache ha estat sempre el més utilitzat. Existeix molta documentació i té un sistema de mòduls molt flexible. Nginx és 8 anys més jove i s'ha popularitzat molt pel fet que utilitza menys recursos tot i anar més ràpid. Tanmateix, cap dels dos és millor que l'altre i de fet es complementen força be. A continuació hi ha un breu resum dels avantatges i desavantatges de Nginx enfront a Apache.

7.3.7.1 Avantatges Nginx:

- Millor rendiment per a contingut estàtic
- Configuració centralitzada

7.3.7.2 Desavantatges Nginx:

- Sistema de mòduls bastant dolent (fins fa poc la única opció era recompilar)
- No serveix contingut dinàmic per si sol
- Menys documentació

Una solució seria crear un híbrid per aprofitar el rendiment de l'Nginx amb els avantatges de l'Apache. Per exemple, utilitzar l'Nginx com a servidor principal que fa de proxy per l'Apache, el qual serveix el contingut dinàmic.

En el nostre cas els avantatges de Nginx pesen més que els desavantatges i, a més, sempre s'estaria a temps d'afegir l'Apache amb un canvi mínim a la configuració de Nginx si fos el cas que Nginx no s'ajustés a les nostres necessitats. Per tant, s'utilitzarà Nginx.

7.3.8 Entorn

Com a entorn de desenvolupament és important utilitzar les eines que s'ajustin més a les nostres necessitats. Personalment em va millor utilitzar un editor de text com l'Atom o el Vim juntament amb GNU/Linux en comptes d'utilitzar un IDE com el WebStorm ja que mitjançant plugins trobo que és més fàcil adaptar-lo al meu workflow. Tant el Django com el React incorporen servidors web de prova per fer el desenvolupament sense haver d'instal·lar i configurar un servidor web.

Per realitzar el disseny gràfic he utilitzat una eina que s'anomena Pencil i per fer els diagrames UML he utilitzat una eina que s'anomena UMLet.

7.3.9 Decisió final

Com a resum, el projecte és desenvoluparà amb els següents frameworks i tecnologies:

Client: React i Redux.

Servidor: Nginx, Django i MariaDB.

8 Anàlisi i disseny del sistema

En aquest apartat es presenta el disseny i l'anàlisi de l'aplicació.

8.1 Diagrama de classes

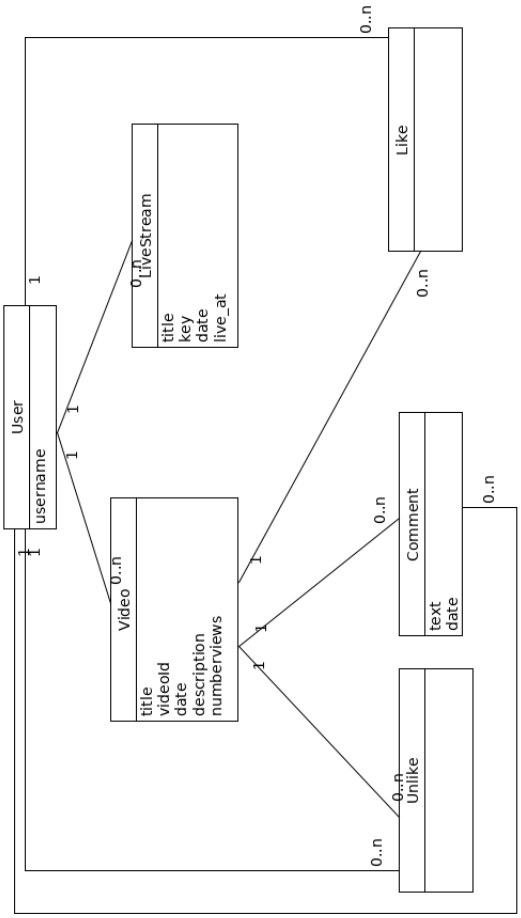

Figura 2: Diagrama de classes

Com es pot observar a la figura 2 hi ha Usuaris que poden tenir molts objectes `Video` i `LiveStream`. Un `Video` pot tenir molts `Like`, `Unlike` o `Comment`. I cada `Like`, `Unlike` i `Comment` és d'un sol `User`.

8.2 Casos d'ús

8.2.1 Usuari anònim

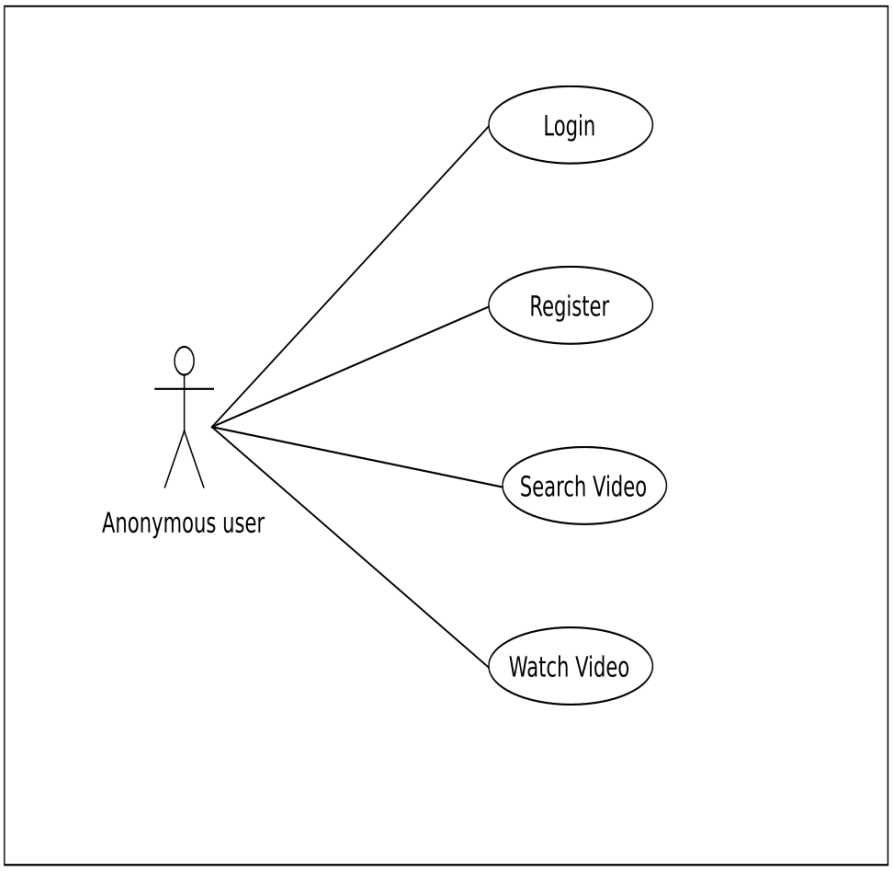

Figura 3: Diagrama de cas d'ús d'usuari anònim

8.2.2 Usuari registrat

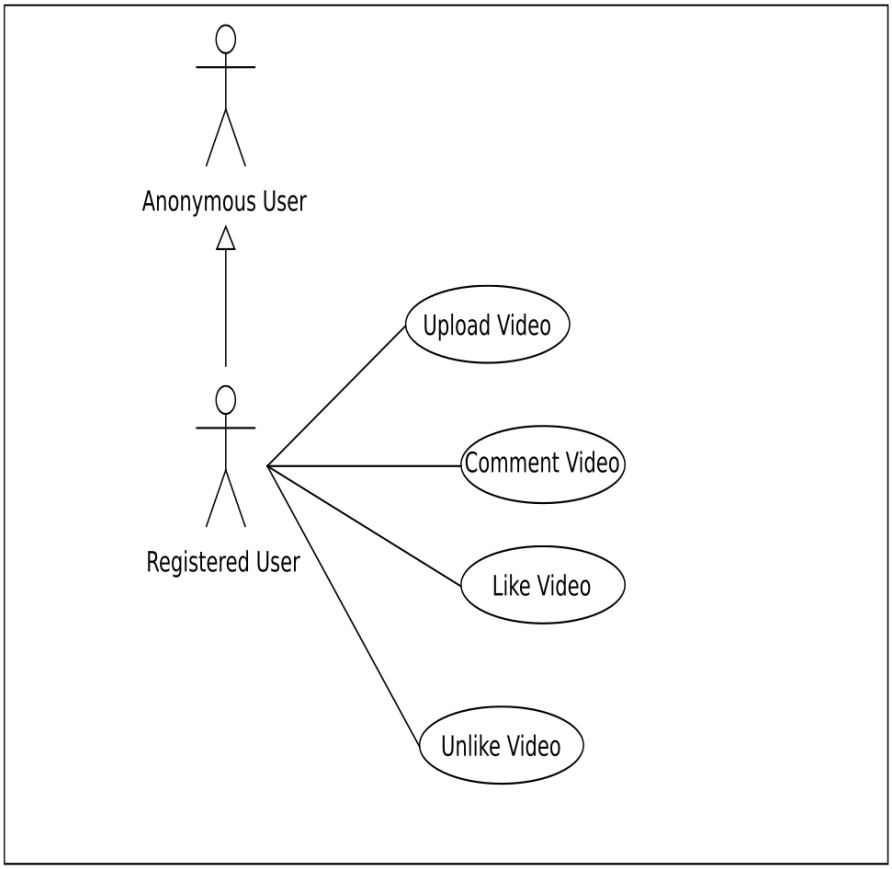

Figura 4: Diagrama de cas d'ús d'usuari registrat

8.2.3 Fitxa Login

Visió: Usuari

Actor: Usuari anònim

Descripció: Iniciar la sessió a un usuari concret

Pre: L'usuari al qual es vol accedir està registrat

1. Entrar nom d'usuari i contrasenya
2. Si la contrasenya pertany a l'usuari llavors
    (a) Es redirecciona a la mateixa pàgina on estava l'usuari

Post: L'usuari ha iniciat sessió i pot accedir a les funcionalitats d'un usuari registrat

8.2.4 Fitxa Veure Vídeo

Visió: Usuari

Actor: Usuari anònim/identificat

Descripció: Es reprodueix un vídeo i es mostren els comentaris

Pre: -

1. Seleccionar el vídeo a visualitzar
2. Mostra el vídeo i els comentaris

Post: -

8.2.5 Fitxa Cercar Vídeo

Visió: Usuari

Actor: Usuari anònim/identificat

Descripció: Es mostra un llistat de vídeos relacionats amb la cerca

Pre: -

1. Entrar paraules clau
2. Seleccionar tots els vídeos que contenen les paraules clau al títol
3. Per a cada vídeo seleccionat
   (a) Mostrar-lo al llistat

Post: S'ha llistat tots els vídeos que compleixen la cerca

8.2.6 Fitxa Pujar Vídeo

Visió: Usuari

Actor: Usuari identificat

Descripció: Es puja un vídeo

Pre: Vídeo amb format mp4

1. Seleccionar el vídeo
2. Entrar un títol i descripció
3. Pujar el vídeo amb títol i descripció

Post: S'ha pujat el vídeo

8.2.7 Fitxa emetre vídeo en directe

Visió: Usuari

Actor: Usuari identificat

Descripció: S'emet un vídeo en directe

Pre: -

1. Entrar un títol i descripció
2. Obtenir clau d'emisió
3. Emetre el vídeo utilitzant la clau

Post: S'està emetent el vídeo

8.2.8 Fitxa Comentar Vídeo

Visió: Usuari

Actor: Usuari identificat

Descripció: Es comenta un vídeo

Pre: S'està mirant el vídeo

1. Entrar comentari
2. Enviar

Post: S'ha afegit el comentari al vídeo

8.2.9 Fitxa Like Vídeo

Visió: Usuari

Actor: Usuari identificat

Descripció: Es marca el vídeo com agradat

Pre: S'està mirant el vídeo

1. Si el Like no estava donat llavors
   (a) Envia el like

Post: S'ha afegit un Like al vídeo

8.2.10 Fitxa Unlike Vídeo

Visió: Usuari

Actor: Usuari identificat

Descripció: Es és marca el vídeo com agradat

Pre: S'està mirant el vídeo

1. Si el Unlike no estava donat llavors
   (a) Envia el unlike

Post: S'ha afegit un unlike al vídeo

8.2.11 Fitxa Logout

Visió: Usuari

Actor: Usuari identificat

Descripció: Surt de la sessió

Pre: La sessió està iniciada

1. Es redirecciona a la mateixa pàgina on estava l'usuari

Post: L'usuari ha sortit de la sessió i no pot accedir a les funcionalitats d'un usuari registrat

8.3 Disseny d'interfície d'usuari

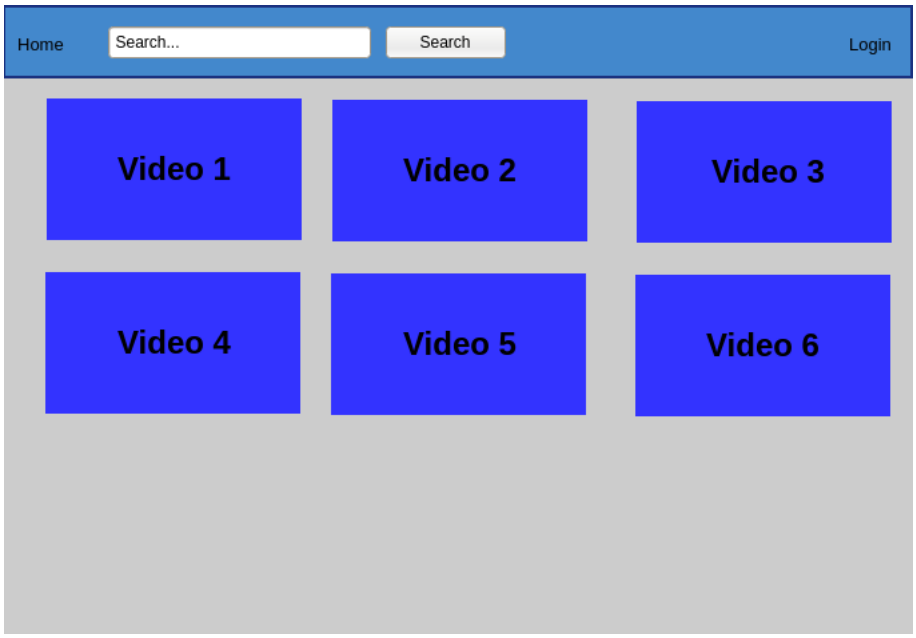

Figura 5: Pantalla principal

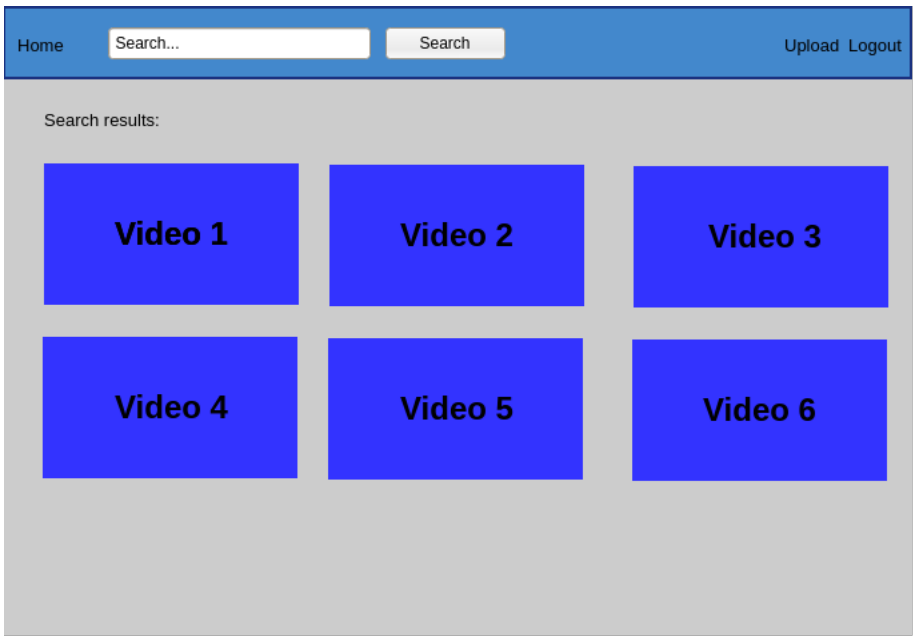

Figura 6: Cerca de vídeos

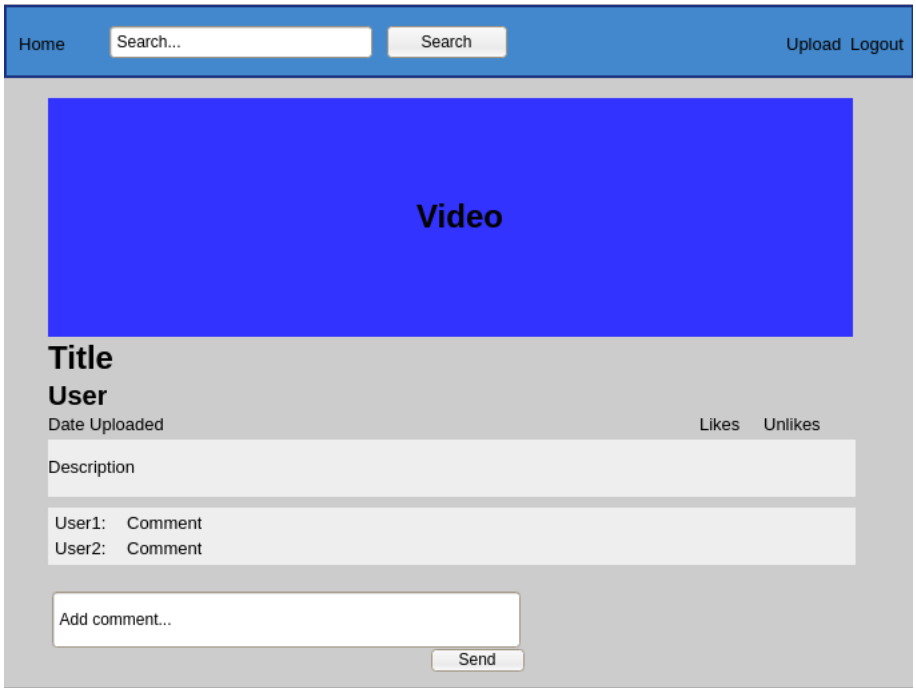

Figura 7: Visualitzar vídeo

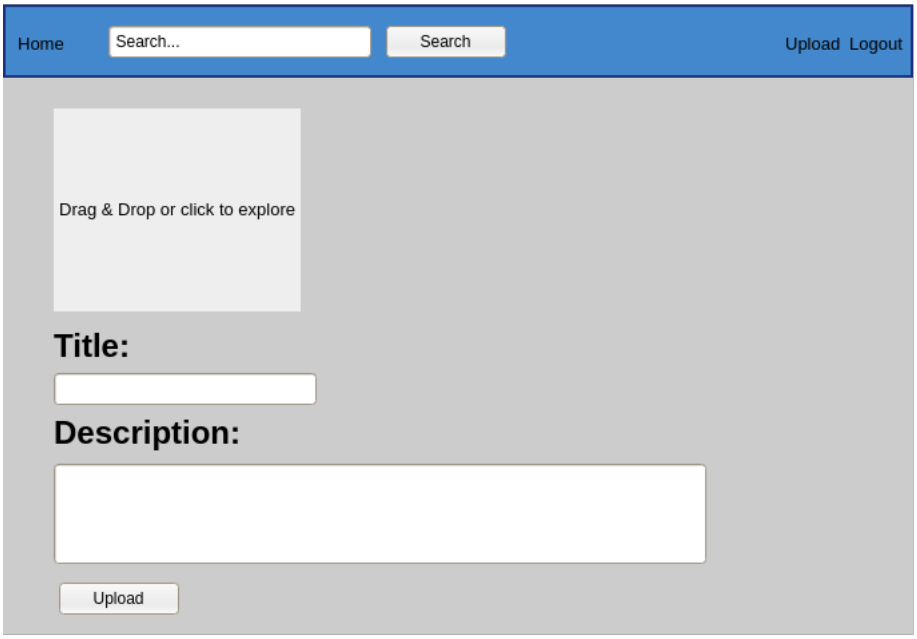

Figura 8: Pujar vídeo

8.3.1 Disseny API REST

Actualment existeixen moltes recomanacions alhora de dissenyar una API Rest. Però no existeix cap mena de document o recomanacions que siguin estàndards o oficials. Tot i això, hi ha una sèrie principis que són bastant utilitzats a API Rest d'aplicacions molt utilitzades com per exemple l'Instagram. A continuació s'exposa un breu de resum del que és una API RESTful i quins principis hauria de complir.

L'API Rest és una interfície que permetrà accedir a recursos mitjançant una url especifica.

8.3.1.1 Utilitzar dos URLs per a cada recurs

Un url per un element en concret i l'altre per la col·lecció d'elements. Per exemple:

/videos →per fer referència a la col·lecció de vídeos

/videos/1 →per fer referència al vídeo amb identificador 1

8.3.1.2 No utilitzar verbs pels recursos

Aquest principi és bàsic si es desitja construir una API fàcil de comprendre i amb el mínim d'URLs possible.

Per exemple podríem tenir:

/obtenirTotsElsVideos

/obtenirUnVideo

/crearUnVideo

/actualitzarUnVideo

Com es pot observar això no és gaire pràctic i és més fàcil l'exemple del punt anterior que suporta les mateixes operacions que aquest cas.

8.3.1.3 Utilitzar mètodes HTTP per operar sobre els recursos

Les URLs especifiquen sobre quin recurs o col·lecció de recursos estem operant i per especificar la operació sobre el recurs es poden utilitzar els verbs HTTP:

GET: Obtenir el recurs (read-only)

POST: Crear un nou recurs

PUT: Actualitzar un recurs existent

DELETE: Eliminar un recurs

Normalment els verbs més utilitzats són el GET i POST. El DELETE no s'acostuma a fer servir ja que normalment no ens interessa eliminar totalment un recurs i més aviat és més útil fer-lo no visible.

8.3.1.4 Utilitzar el Query String (?) per paràmetres complexos

És recomanable intentar mantenir les urls el més simple possible, però a vegades és més convenient i fàcil utilitzar paràmetres a la URL. Per exemple, en el cas que volguéssim obtenir la col·lecció de vídeos que estan categoritzats com a "Entreteniment" és podria fer:

GET /videos?categoria=entreteniment

8.3.1.5 Utilitzar els codis d'estat HTTP a la resposta

A l'aplicació a desenvolupar hi hauràn dos recursos els vídeos i els streams. Per tant la API Rest serà la següent:

GET /videos/last →Obté els últims 20 vídeos

GET /streams/last →Obté els últims 20 streams

GET /videos/best →Obté els 20 millors vídeos

GET /videos/id →Obté el vídeo id

POST /videos →Pujar un vídeo, a la resposta s'inclou un id que serveix per estabilr el títol i descripció del vídeo

PUT /videos/id →Actualitzar el títol i descripció del vídeo

PUT /videos/id/like →Establir un like

PUT /videos/id/unlike →Establir un unlike

POST /videos/id/comment →Crear un comentari

POST /login →Iniciar sessió

9 Implementació i proves

9.1 Client

El client s'ha desenvolupat utilitzant HTML, CSS i Javascript. Però alhora de programar s'utilitzarà Javascript i JSX. JSX és una extensió de la sintaxi de Javascript que permet crear elements de React. Com a recordatori el client es programa utilitzant la llibreria Redux i React. A continuació s'explica per separat com funciona el React per després poder entendre perquè ens pot fer falta utilitzar el Redux i finalment la implementació de l'aplicació.

9.1.1 React

React és basa en Components. Cada Component és una peça de interfície d'usuari. Els Components s'han de construir com peces reutilitzables i isolades. Els Components conceptualment són funcions Javascript que, donada una entrada anomenada "props", retornen elements React que descriuen el que ha d'aparèixer a la pantalla de l'usuari. La clau en desenvolupar una aplicació amb React és que aquests Components és poden composar: Un Component pot contenir un altre component.

Per crear un Component molt bàsic és faria de la següent forma:

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Com es pot observar la funció render ha de retornar un element React. En aquest cas és retorna utilitzant JSX.

La variable this.props conté tots els inputs del component i en aquest cas s'utilitza l'input name.

Els Components, doncs, són immutables: donat un input tornen un output

però en cap moment es modifiquen. L'estat del component està a this.state i és pot establir amb la funció this.state. Per exemple, és podria modificar l'estat del component al fer un click a un boto i a continuació el React detecta quins canvis s'ha de produir a l'output i els realitza. En cap moment es modifica el DOM directament i, en comptes, és modifica el DOM virtual de React.

Com es pot observar el React és molt útil alhora de construir interfícies però gestionar l'estat d'aquesta manera no és molt escalable. La solució a aquest problema és utilitzar Redux.

9.1.2 Redux

9.1.2.1 Perquè?

Les aplicacions SPA cada vegada són més complexes en el sentit que han de gestionar molts estats de l'aplicació. Els estats poden ser respostes del servidor, dades creades localment, etc.

Com es gestiona l'estat pot ser una tasca molt complicada ja que, per exemple, podríem tenir una vista que actualitza un model i aquest model podria modificar un altre model, el qual podria modificar una altra vista... Si s'arriba a aquest punt és impossible de controlar amb precisió l'estat de l'aplicació i, per tant, és difícil de desenvolupar sobre l'aplicació (corregir errors, afegir funcionalitats, etc)

El React és molt útil ja que separa el sincronisme i la mutació però no proporciona res per gestionar l'estat. El Redux té com a objectiu principal fer que l'estat sigui fàcil de preveure.

9.1.2.2 Conceptes bàsics

L'estat de l'aplicació conceptualment està descrit com un objecte com per exemple en una aplicació de tasques és podria definir de la següent forma:

{

```
  todos: [{
    text: 'Eat food',
    completed: true
  }, {
    text: 'Exercise',
    completed: false
  }],
  visibilityFilter: 'SHOW_COMPLETED'
}
```

L'estat no es pot modificar directament per evitar que diferents parts del codi puguin canviar l'estat de manera arbitrària i d'aquesta manera s'aconsegueix un únic flux de dades.

Per canviar l'estat s'ha de disparar una acció que defineix el que ha passat:

```
{ type: 'ADD_TODO', text: 'Go to swimming pool' }
{ type: 'TOGGLE_TODO', index: 1 }
{ type: 'SET_VISIBILITY_FILTER', filter: 'SHOW_ALL' }
```

Aquestes accions defineixen que s'ha d'afegir una tasca amb text "Go to swimming pool" , es canvia l'estat de la tasca amb índex 1 i es canvia el filtre de visualització perquè mostri totes les tasques completades i no completades (SHOW_ALL)

Finalment per ajuntar les accions i l'estat s'utilitzen unes funcions anomenades reducers. Simplement són funcions que donat un estat i una acció retornen el nou estat. Per exemple per gestionar l'acció amb tipus 'SET_VISIBILITY_FILTER' es crea el següent reducer:

```
function visibilityFilter(state = 'SHOW_ALL', action) {
  if (action.type === 'SET_VISIBILITY_FILTER') {
    return action.filter;
  } else {
    return state;
  }
}
```

Aquest reducer defineix que l'estat visibilityFilter de l'aplicació és el camp

'filter' de l'acció de tipus 'SET_VISIBILITY_FILTER'.

Aquest flux de dades queda il·lustrat a la figura 9:

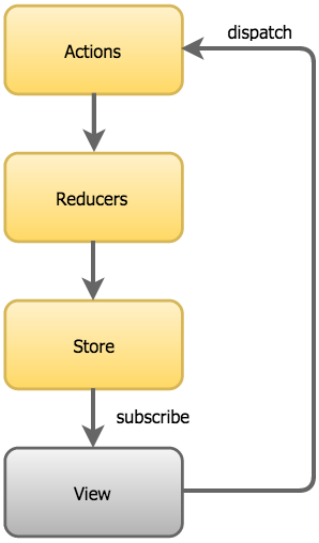

Figura 9: Flux de dades del Redux

9.1.3 Començant l'aplicació

Primer de tot es construeix la estructura bàsica de l'aplicació utilitzant el creador de projectes del React. Aquest no aporta el Redux però ja s'afegirà després ja que només consisteix en crear les accions i els reducers.

Per inicialitzar l'estructura bàsica de l'aplicació s'utilitza l'eina de React oficial `create-react-app`. Per utilitzar-la primer s'instal·la al sistema mitjançant el gestor de paquets `NPM` i a continuació s'executa per crear l'aplicació:

```
npm install -g create-react-app
create-react-app app
```

Aquestes comandes creen una aplicació bàsica amb Webpack per gestionar totes les dependències.

Ara l'aplicació consisteix d'un sol index.html el qual carrega el React i App.js és el punt d'entrada de l'aplicació.

Per afegir el Redux s'instal·la la dependència `react-redux` mitjançant l'NPM:

```
npm install --save react-redux
```

Addicionalment utilitzarem el framework Bootstrap per facilitar la programació de la interfície i s'utilitzarà una llibreria anomenada Axios per realitzar les crides API dirigides al servidor. Com que les accions de Redux han de ser funcions pures i les crides API no sempre retornen el mateix output llavors també farà falta la llibreria redux-thunk per tal d'encapsular les crides API dintre de funcions pures. Aquestes llibreries/frameworks s'instal·len mitjançant l'NPM:

```
npm install --save react-bootstrap
npm install --save axios
npm install --save redux-thunk
```

Amb el que s'ha fet fins ara ja es pot començar a implementar el client utilitzant React i Redux. És a dir una SPA que faci crides API i mostri per pantalla el que faci falta. Per executar la aplicació en un entorn de proves s'executa la següent comanda per servir la aplicació a `localhost:3000`:

```
npm start
```

A mesura que s'ha anat desenvolupant la aplicació, s'han requerit certes funcionalitats extres. A continuació hi ha un llistat de llibreries que s'han afegit per tal de solucionar certs problemes:

- `react-router-redux` Per facilitar la utilització del router amb React i Redux
- `dashjs` Per reproduir vídeo DASH
- `react-hls` Per reproduir vídeo HLS
- `react-dropzone` Per navegar o arrossegar fitxers

9.1.4 Estructura de l'aplicació

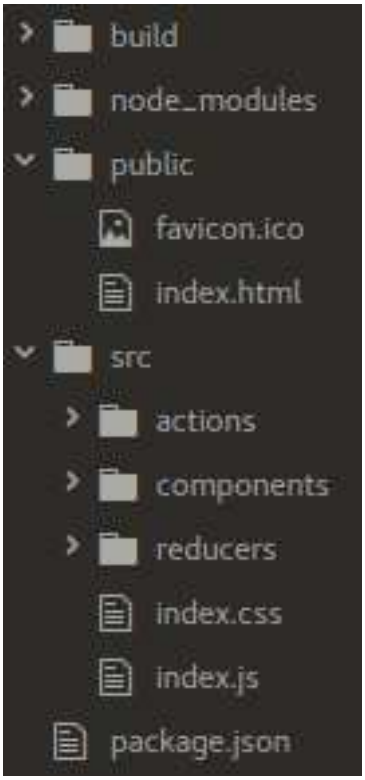

Figura 10: Estructura de directoris del client

- `build/` conté el build dirigit per entorns en producció
- `node_modules/` conté totes les dependències instal·lades que s'han especificat al `package.json`
- `public/` conté el favicon i el punt d'entrada html de l'aplicació
- `src/` conté tot el codi de l'aplicació separat en tres directoris `actions/`, `reducers/` i `components/`.

9.1.5 Actions

Les accions s'han definit en un sol fitxer ja que n'hi a relativament poques, i en el cas que l'aplicació comencés a créixer es podrien anar separant en mòduls. A continuació es dona un exemple d'acció que fa la crida api per obtenir un vídeo en concret:

```
export const RECEIVE_VIDEO = 'RECEIVE_VIDEO'

export const receiveVideo = json => ({
  type: RECEIVE_VIDEO,
  video: json.data
})

export const getVideo = (id, token) => {
    return function(dispatch) {
      let instance = axios.create({
        baseURL: base + 'videos/' + id + '/',
      });
      if (token !== null) {
        instance = axios.create({
          baseURL: base + 'videos/' + id + '/',
          headers: {'Authorization': 'Token ' + token}
        });
      }
      instance.get()
      .then(function(response) {
        dispatch(receiveVideo(response))
      })
    }
}
```

Com es pot observar hi han dos funcions. `receiveVideo` i `getVideo`. `getVideo` és la funció que es cridarà des del Component. Aquesta funció en realitat encapsula i retorna una altra funció que és la que fa la crida API mitjançant Axios. Això es degut a la naturalesa asíncrona de la crida API. Quan es rep la resposta és llança una acció que construeix la funció `receiveVideo`. Aquesta funció rep la resposta del servidor en format JSON i retorna la acció

de tipus `RECEIVE_VIDEO` i amb el camp `video` que conté les dades del vídeo. Aquesta acció ja està llesta per ser rebuda per el reducer corresponent.

9.1.6 Reducers

Els reducers també estan agrupats en un sol fitxer ja que n'hi ha relativament pocs. Si continuem amb l'exemple anterior el reducer encarregat de recollir la acció que s'ha llançat és el següent:

```
function video(state = null, action) {
  switch (action.type) {
    case RECEIVE_VIDEO:
    return {
      video: action.video
    }
    default:
    return state
  }
}
```

El reducer el que fa és guardar el camp `video` quan l'acció és del tipus `RECEIVE_VIDEO` al objecte `video`. Així de simple. L'únic pas que falta fer és connectar el reducer amb l'estat del Component.

9.1.7 Components

Aprofitant la composició de components s'ha desenvolupat la interfície en petits mòduls. El Component principal que és el punt d'entrada de l'aplicació és App.js i s'ha anat descomposant l'aplicació en Components. La composició dels components utilitzada es pot veure a la figura 11:

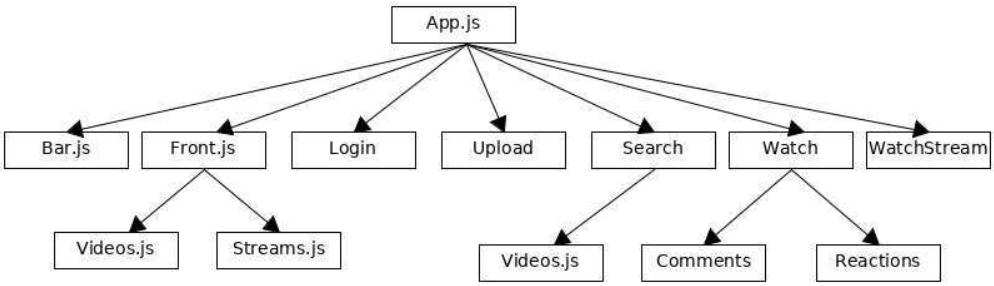

Figura 11: Composició de components

El Bar.js és la barra de navegació de dalt, que sempre es mostra. A sota la barra hi ha el contingut que es va canviat dinàmicament. El contingut potser qualsevol dels Components:

- Front.js: Mostra el llistat de vídeos i d'streams actius. Esta composat per Videos.js i Streams.js que són els components que llisten els vídeos i els streams respectivament
- Login.js: Mostra la pantalla de login
- Upload.js: Mostra la pantalla per pujar un vídeo
- Live.js: Mostra la pantalla per obtenir la clau que permet fer streaming
- Search.js: Mostra els resultats d'una cerca
- Watch.js: Mostra un vídeo. Aquest també està composat per Comments.js i Reactions.js que són els comentaris i els likes d'un vídeo respectivament
- WatchStream.js: Mostra un stream

L'App.js és bàsicament el que fa de Router:

```
class App extends Component {

  render() {

    return (
    <div className="App">
      <Router>
        <div className="container-fluid">
          <Bar />
          <div className="content">
            <Route exact path="/" component={Front}/>
            <Route exact path="/login" component={Login
               }/>
            <Route exact path="/upload" component={
               Upload}/>
            <Route exact path="/live" component={Live
               }/>
            <Route path="/videos" component={Search}/>
            <Route path="/watch" component={Watch}/>
            <Route path="/watchstream" component={
               WatchStream}/>
          </div>
        </div>
      </Router>
    </div>
    )
  }
}
```

El Router permet navegar per la SPA com si fos una MPA. Per exemple, quan és va a `/watch?video=1` es posa el Component `Watch` el qual processa la url per obtenir l'id del vídeo que és vol reproduir per mostrar-lo.

El component `Watch` per tal de "rebre" el vídeo llança la funció `getVideo` que s'ha vist a l'apartat d'Actions. El Reducer, tal i com s'ha descrit a l'apartat anterior, "obté" el vídeo i el posa a l'objecte video.

Per tal d'obtenir aquest objecte s'ha de connectar el Reducer amb el Com-

ponent de la següent forma:

```
const mapStateToProps = state => {
  const { video, token } = state
  return {
    video, token
  }
}

export default connect(mapStateToProps)(withRouter(Watch))
```

La connexió és realitza mitjançant la funció `connect` que rep la funció que defineix quins objectes es volen connectar (en aquest cas video i token). En aquest tros de codi a més es pot veure que també hi ha la funció `withRouter` que és necessària perquè les funcionalitats del Router funcionin en aquest Component.

9.1.8 Reproducció de vídeo

9.1.8.1 DASH

El reproductor DASH `dash.js` modifica el DOM directament. S'afegeix un element `video` a l'html:

```
<video id="videoPlayer" controls></video>
```

Mitjançant javascript es busca l'element i el dash.js fa la seva feina:

```
var url = "http://dash.edgesuite.net/envivio/
    EnvivioDash3/manifest.mpd";
var player = dashjs.MediaPlayer().create();
player.initialize(document.querySelector("#videoPlayer"
    ), url, true);
```

El problema és que amb React això no és pot fer d'aquesta manera ja que mai és modifica el DOM directament. Per solventar-ho React proporciona els Refs i, tot i que va en contra del flux de dades del React el cas de reproducció de

vídeo és un cas en el qual és necessari. Per crear el reproductor DASH, doncs, quan el Component és munta (`componentDidMount`) s'executa una funció que crea el reproductor `dash.js` mitjançant Refs. A continuació s'exposa el codi més rellevant.

L'HTML que renderitzarà el component és un element `vídeo`. Per utilitzar els Refs s'ha de posar el tag ref a l'element:

```
render() {
    return (
    <div>
        <video ref="dashplayer" controls></video>
    </div>
    )
}
```

Quan el component és munta crida a una funció del mateix component amb l'id del vídeo com a paràmetre:

```
  componentDidMount() {
    this.createDashPlayer(this.props.id)
  }
```

La funció `createDashPlayer` és la que utilitzarà els Refs mitjançant `this.refs`:

```
    this.video = this.refs["dashplayer"]
    this.player = MediaPlayer().create()
    this.player.initialize(this.video, url, true)
```

9.1.8.2 HLS

El reproductor HLS és més immediat ja que existeix un reproductor HLS encapsulat en un Component React anomenat `ReactHLS`. Nómes s'ha de passar la url com a paràmetre i el Component ja s'encarrega de la resta:

```
class Stream extends Component {
  render() {
```

```
        const url = "http://" + window.location.hostname +
            ":8083/live/" + this.props.username + "/index.
            m3u8"
        return (
        <div>
            <ReactHLS url={url} />
        </div>
        )
    }
}
```

9.2 Servidor

El servidor s'ha desenvolupat amb Python utilitzant el framework Django. Aquest segueix el patró MVT tot i que no s'utilitzaran els `Templates`. La part d'streaming de vídeo en directe s'ha implementat utilitzant el mòdul `nginx-rtmp` del servidor `Nginx`.

9.2.1 Començant l'aplicació

Abans de poder crear una aplicació s'ha de crear un projecte de Django. Un projecte pot tenir múltiples aplicacions i agrupa un conjunt de configuracions. Per crear un projecte al directori `app` s'ha d'executar la següent comanda:

```
django-admin startproject server
```

Aquesta comanda crea un projecte Django al directori server. Aquest directori es pot reanomenar ja que no afecta a la configuració del Django. A continuació ja es pot crear una aplicació utilitzant la següent comanda dins del directori del projecte que s'acaba de crear:

```
python manage.py startapp mediastream
```

Aquesta comanda crea l'aplicació mediastream. La estructura del projecte Django ha quedat de la següent forma:

```
manage.py
server/
    __init__.py
    settings.py
    urls.py
    wsgi.py
mediastream/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

La configuració del projecte està a `settings.py` i el punt d'entrada és a `urls.py`. La aplicació es desenvoluparà creant els models a `models.py` i afegint vistes al `views.py` sobre els models.

Per crear la API Rest fa falta una dependència anomenada `Django Rest Framework`. Es pot instal·lar mitjançant el gestor de paquets de Python `pip`:

```
pip install djangorestframework
```

Alhora de desenvolupar el Django proporciona un servidor de desenvolupament executant:

```
python manage.py runserver
```

9.2.2 API Rest

La API Rest amb el `Django` i el `Django Rest Framework` s'implementa de la següent manera:

1. És defineix la ruta a `server/urls.py` amb l'url i la `view` corresponent que està definida a `views.py`

2. La `view` processa la petició i obté les instàncies de `Model` requerits per la petició

3. Les instàncies es serialitzen com a objectes Python per a posteriorment ser convertides i enviades en format JSON

A continuació s'exposa una petita part de l'API Rest per veure com s'ha implementat.

Primer de tot es defineix el model, en aquest cas del Vídeo:

```
class Video(models.Model):
    title = models.CharField(max_length=200)
    videoId = models.CharField(max_length=200)
    date = models.DateField(auto_now=False,
        auto_now_add=True)
    description = models.CharField(max_length=1000)
    numberviews = models.BigIntegerField(default=0)
    user = models.ForeignKey(User, on_delete=models.
        CASCADE)
    enabled = models.BooleanField(default=False)
```

El Model permetrà crear instàncies i agrupar-les dins de conjunts (Querysets). Per fer la conversió de instància a objecte Python s'ha de crear un Serializer:

```
class VideoSerializer(serializers.
    HyperlinkedModelSerializer):
    user = UserSerializer(read_only=True)
    comments = CommentSerializer(many=True, read_only=
        True)
    likes = serializers.SerializerMethodField()
    unlikes = serializers.SerializerMethodField()
    currentuserlike = serializers.SerializerMethodField
        ()
    class Meta:
        model = Video
        fields = ('title', 'videoId', 'numberviews', '
            likes', 'unlikes', 'date', 'description', '
```

```
            user', 'comments', 'currentuserlike')
    def get_likes(self, obj):
        return obj.likes.count()
    def get_unlikes(self, obj):
        return obj.unlikes.count()
    def get_currentuserlike(self, obj):
        if 'request' in self.context:
            user = self.context['request'].user
            if obj.likes.all().filter(user__id=user.id)
                :
                return 'LIKE'
            elif obj.unlikes.all().filter(user__id=user
                .id):
                return 'UNLIKE'
        return '␣'
```

Al serializer es defineix quins camps del model es volen incloure amb la variable `field`. Per incloure altres Models dins del Model que s'està serialitzant es pot fer referència a un altre Serializer, com per exemple en aquest cas el serializers `UserSerializer` que serialitza instàncies `User` i el serializer `CommentSerializer` que serialitza instàncies `Comment`. Addicionalment es poden utilitzar funcions per establir camps del serializer com per exemple els camps likes, unlikes i currentuserlike que es defineixen en temps d'execució mitjançant les funcions `get_likes`, `get_unlikes` i `get_currentuserlike` respectivament.

Per implementar la API /videos/last es pot fer amb la següent `View`:

```
@api_view(['GET', 'PUT'])
@permission_classes((IsAuthenticatedOrReadOnly, ))
def VideoByIdView(request, videoId):
    id = videoId
    try:
        v = Video.objects.get(videoId=id)
    except v.DoesNotExist:
        return Response(status=status.
            HTTP_404_NOT_FOUND)
    if request.method == 'GET':
```

```
            v = Video.objects.get(videoId=videoId)
            v.numberviews += 1
            v.save()
            return Response(VideoSerializer(v, context={'
                request': request}).data, status=status.
                HTTP_200_OK)
        if request.method == 'PUT' and request.user:
            title = request.data.get('title')
            description = request.data.get('description')
            if  title and description and v.user == request
                .user:
                v.title = title
                v.description = description
                v.enabled = True
                v.save()
                return Response(VideoSerializer(v, context
                    ={'request': request}).data, status=
                    status.HTTP_201_CREATED)
            return Response(status=status.
                HTTP_400_BAD_REQUEST)
```

En aquest petit tros de codi es pot observar:

- La variable `permission_classes` permet definir el tipus d'usuari que pot utilitzar aquesta vista. En aquest cas és pot utilitzar com a usuari anònim amb peticions de lectura (GET) i si l'usuari està autenticat pot realitzar operacions d'escriptura (PUT).
- Es cerca el vídeo per id i a continuació en el cas del `GET` i es respon amb el vídeo serialitzat. En el cas del `PUT` s'obté el títol i la descripció del vídeo de la petició i s'actualitza el vídeo amb aquesta informació.
- En tots els casos es retorna l'estat HTTP adequat (404, 200 i 201)
- Els vídeo obtingut es serialitza amb `VideoSerializer`

Perquè aquesta view estigui disponible com una url s'ha de definir a `urls.py`:

```
urlpatterns = [
    url('^videos/(?P<videoId>[a-fA-F\d]{32})/$', views.
        VideoByIdView)
]
```

D'aquesta manera estarà disponible a la url /videos/videoId on mitjançant una expressió regular s'ha definit que el `videoId` és un valor format per números i lletres de 32 caràcters de longitud. Finalment es fa referència a la `View` que s'ha definit.

9.2.3 Streaming de vídeo en directe

La implementació del streaming de vídeo en directe s'ha solventat mitjançant el mòdul `rtmp-nginx`. Tot i què a l'apartat de Decisions s'ha explicat que utilitzaríem exclusivament MPEG-DASH per la reproducció de vídeo al client, en aquest cas l'stream RTMP s'ha convertit a HLS i, per tant, els clients utilitzaran HLS. Aquesta decisió s'ha pres degut a que s'ha observat que el rendiment de RTMP a HLS és molt millor que el de RTMP a MPEG-DASH per al mòdul `rtmp-nginx`. La configuració del Nginx és la següent:

```
rtmp {
    server {

        listen 1935;
        chunk_size 4000;

        application app {
            live on;
            deny play all;

            push rtmp://127.0.0.1:1935/hls;

            on_publish http://127.0.0.1/api/on_publish
                /;
            on_publish_done http://127.0.0.1/api/
                on_publish_done/;
```

```
        }


        application hls {
            live on;
            deny play all;

            allow publish 127.0.0.1;
            deny publish all;

            hls on;
            hls_path /usr/share/nginx/html/live;

            hls_nested on;
            hls_fragment_naming system;
        }
    }
```

Aquesta configuració fa disponible el servei de conversió de rtmp a hls al port 1935 del servidor. El client ha d'utilitzar una clau (key) alhora d'enviar l'stream. Aquesta clau identifica l'stream i s'utilitza per gestionar l'stream en directe al servidor Django. El funcionament és el següent:

1. El client rtmp envia l'stream a rtmp://server/hls/key
2. El servidor en temps real converteix l'stream rtpm a hls. En el moment en que comença a convertir-lo
   realitza un `POST http://127.0.0.1/api/on_publish/` amb el `key`. Aquest `POST` va dirigit al Django. La resposta del Django ha de contenir l'identificador del vídeo `id` que defineix on el guarda: `/usr/share/nginx/html/live/{id}`.
3. En el moment que para de rebre l'stream, l'Nginx envia al Django:
   `POST http://127.0.0.1/api/on_publish_done/`

El vídeo, resideix a la carpeta `live/{id}` del servidor i el client només l'ha de reproduir. Tal i com s'ha explicat el Django defineix el directori on es guarda l'stream i, per tant, pot indicar al client on trobar-lo.

10 Implantació i resultats

10.1 Implantació del servidor

Per realitzar el desplegament, primer s'ha de tenir en compte la diversitat de tecnologies necessàries perquè el servei funcioni correctament i quines són les funcions de cada una.

- Els vídeos i streams es serveixen estàticament
- El client és codi HTML, Javascript i CSS que s'ha de servir sense cap processament per part del servidor. És a dir, també és contingut estàtic
- El servidor API Rest, en aquest cas la funció del Django, ha d'estar disponible en una url concreta per no crear conflicte amb el contingut estàtic com per exemple el client
- Tot el contingut estarà servit o delegat pel servidor Nginx

El client React es pot posar a l'arrel del servidor. Per crear un `build` de l'aplicació implementada s'executa la següent comanda:

```
npm build
```

Aquesta comanda crearà codi optimitzat per producció (comprimit, compatible...) al directori `build/` del projecte. El codi resultant es posa a l'arrel de l'Nginx (`/usr/share/nginx/html/`).

En el cas del servidor Django només fa falta que estigui en un directori on l'Nginx tingui permisos de lectura i escriptura. S'ha posat al directori `api/` de l'arrel de l'Nginx.

Els directoris de l'arrel de Nginx on s'emmagatzemarà els vídeos i els streams són `videos/` i `live/`.

Per tant la configuració de Nginx ha de ser la següent:

- Tot el contingut estàtic es serveix pel port HTTP (80)
- Qualsevol petició dirigida a /api s'ha de redirigir al Django
- En el cas que no existeixi el contingut que demana la petició (404) s'ha de redirigir la petició al client React. Això es degut a que és una SPA i s'utilitza un Router per fer la navegació

La configuració del Nginx més rellevant per aconseguir la anterior configuració és la següent:

```
server {
        listen        80;
        proxy_read_timeout 300s

        location / {
            root   /usr/share/nginx/html;
            index  index.html index.htm;
        }

        location /api {
             rewrite ^/api/(.*) /$1  break;
             include      /etc/nginx/uwsgi_params;
             uwsgi_pass   django;
             uwsgi_read_timeout 600;
        }

        error_page  404              /index.html;
        # Redirigir els 404 al React
    }
```

Es pot observar com definim l'arrel de la web apuntant a l'index.html que és el client React i a més es defineix l'error 404 (pàgina no trobada) com index.html. Això és així perquè és una SPA i la navegació l'ha de gestionar el client.

A "location /api" es defineix que tot el que vagi a /api es rederigeixi al Django però sense la ruta "/api" (rewrite).

El Nginx no pot servir contingut dinàmic per si sol. En el nostre cas el contingut dinàmic és el Django. Per servir-lo es pot observar que s'ha utilitzat el `uwsgi` per aconseguir-ho. L'uwsgi és configura amb el fitxer `django.ini` que conté l'entorn de Django i per tal d'iniciar-lo es fa amb la següent comanda:

```
uwsgi --ini django.ini
```

10.2 Resultats

A continuació s'exposa mitjançant captures de pantalla els resultats de l'aplicació:

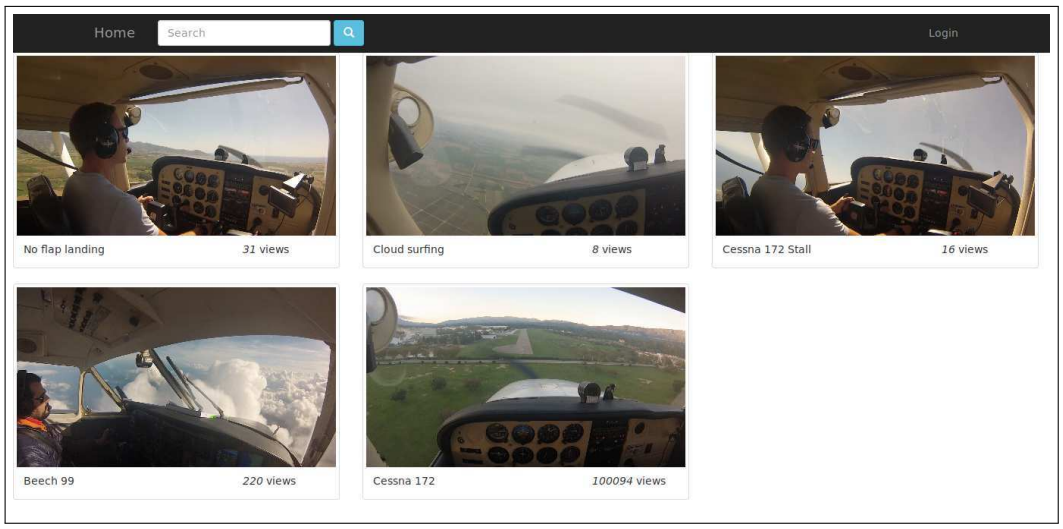

Figura 12: Pantalla principal per a usuari anònim

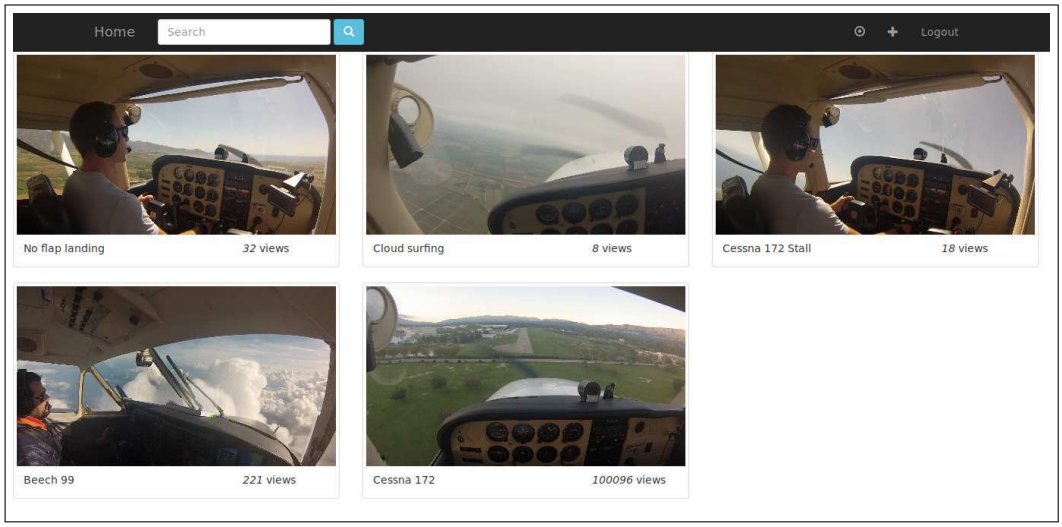

Figura 13: Pantalla principal per a usuari registrat

Com es pot observar a la figura 13, l'usuari registrat té com a opció de pujar un vídeo (signe positiu) i obtenir la clau d'emissió d'un vídeo en directe (signe de gravació).

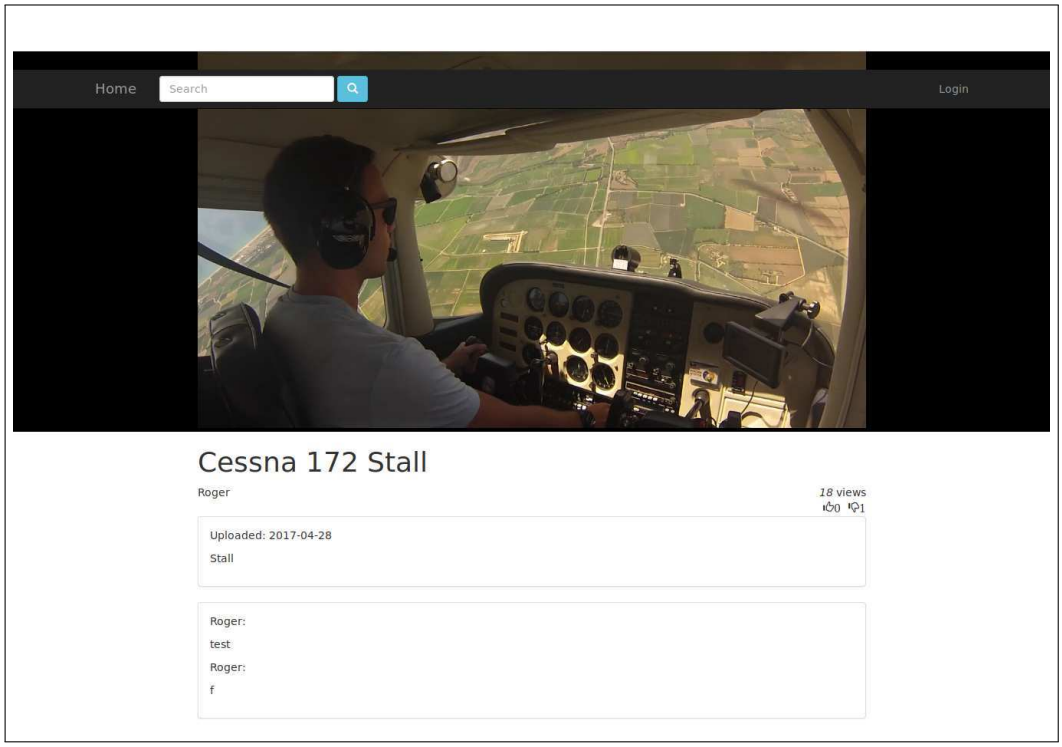

Figura 14: Vista de vídeo per a usuari anònim

Com es pot observar a la figura 14, l'usuari anònim pot veure el vídeo i les dades però no pot donar like, unlike o enviar un comentari.

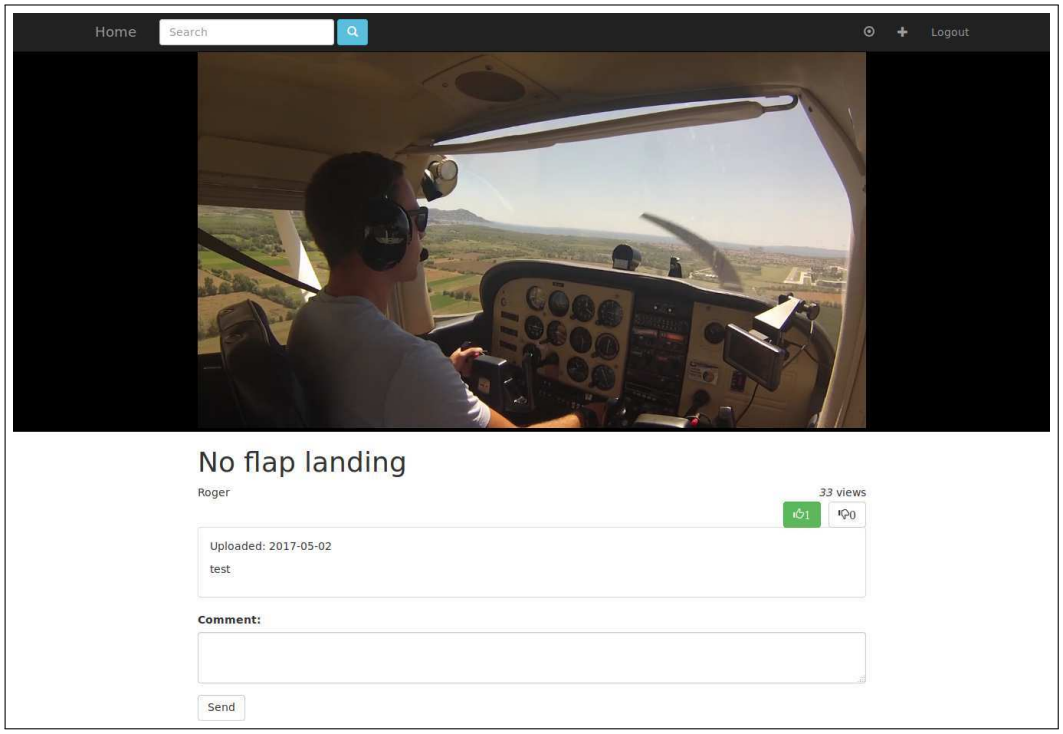

Figura 15: Vista de vídeo per a usuari registrat

Com es pot observar a la figura 15, si l'usuari està registrat llavors pot donar like, unlike o comentar. En aquest cas es veu com ha donat like.

Figura 16: Pujar vídeo

Com es pot observar a la figura 16, l'usuari registrat pot pujar un vídeo mitjançant aquest formulari.

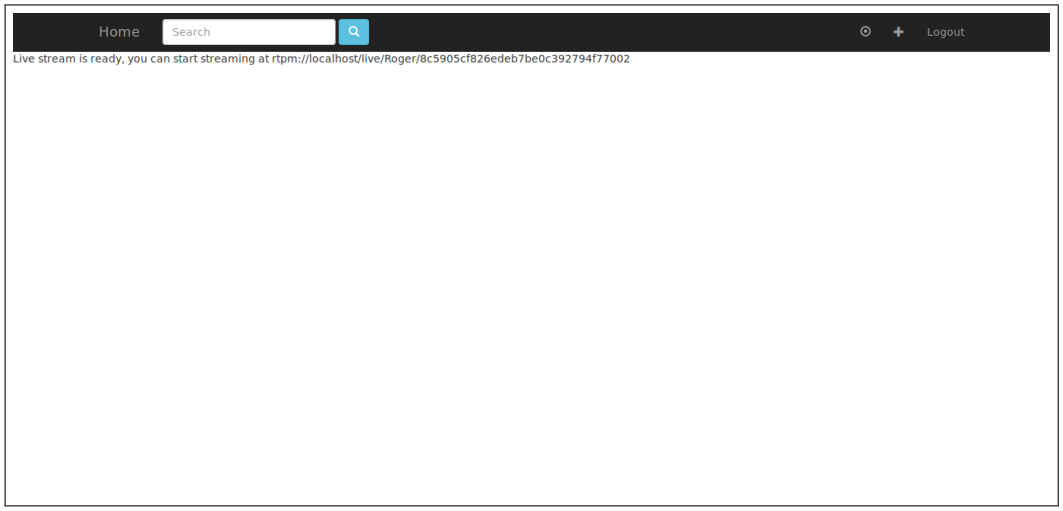

Figura 17: Obtenir clau

Com es pot observar a la figura 17, l'usuari registrat si fa click a la icona

superior amb el signe de gravació pot emplenar un formulari amb el títol i la descripció de la emissió en directe que vol fer. A continuació li surt per pantalla la URL amb la clau on pot començar a emetre el vídeo utilitzant qualsevol aplicació que suporti el protocol RTMP.

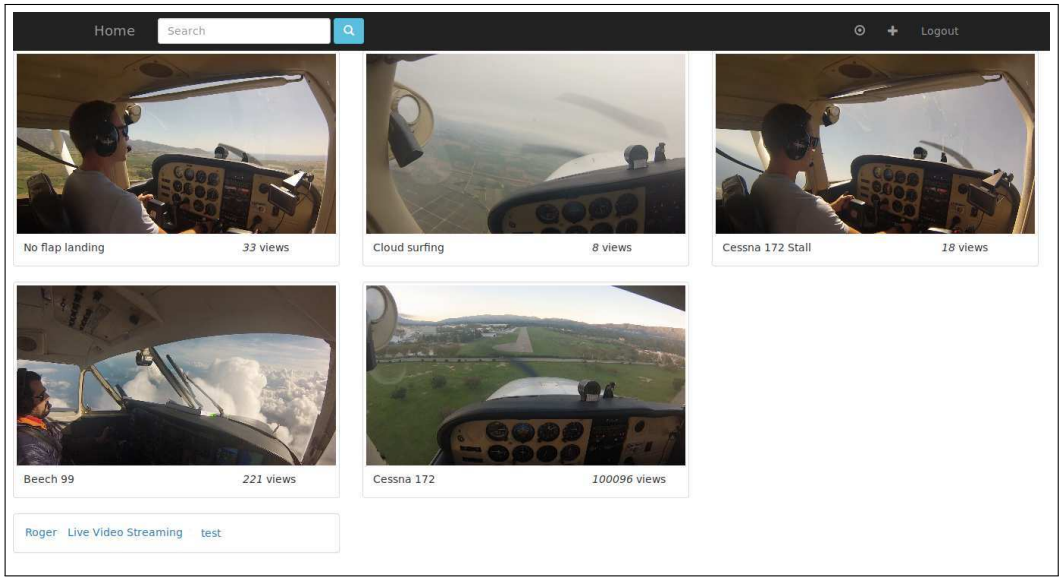

Figura 18: Stream en directe

Com es pot observar a la figura 17, un cop l'usuari de la figura 17 ha començat a emetre el vídeo, qualsevol usuari pot veure a la pantalla principal (a la part inferior) que hi ha un stream en directe. Si s'obre l'stream en directe es veurà l'stream de vídeo.

11 Conclusions

L'objectiu del projecte era implementar un servei d'streaming de vídeo a la carta i de vídeo en directe. Els resultats obtinguts són:

- S'ha implementat un servidor amb Nginx i Django
- S'ha implementat un client web SPA amb React i Redux
- S'ha guanyat coneixement sobre com funciona un servei d'streaming a tots els nivells

Per tant, els objectius del projecte s'han assolit.

Els requisits de l'aplicació eren oferir una aplicació que permetés cercar, visualitzar, comentar i compartir vídeos. Aquests requisits s'han assolit i es compleixen utilitzant els navegadors més populars com el Chrome, Safari i Firefox.

El resultat que s'ha implementat no ofereix amb detall totes les funcionalitats que proporcionen serveis similars com per exemple el Youtube. En aquest sentit el resultat del projecte no és una competència a aquests tipus de servei. Tanmateix, el resultat del projecte és útil per aprofundir en el coneixement de com funcionen aquests serveis i és una bona proposta d'aplicació per a qualsevol empresa o persona que tingui necessitat d'utilitzar el seu propi servei d'streaming.

11.1 Opinió personal

Tot i que s'han assolit els objectius del projecte, la aplicació encara està en un estat molt inicial i manca de certes funcionalitats extres. La causa és que un servei d'streaming complet comporta molta feina ja que involucra tota una arquitectura al darrera de processament de vídeo i distribució de contingut relativament complexa. A més, totes les tecnologies utilitzades en aquest projecte eren molt noves per mi i en la majoria de casos era el primer cop

que les utilitzava. Tot i això, el projecte ha suposat un guany d'experiència sobre aquestes tecnologies a canvi de hores d'estudi i treball. De la mateixa manera durant el transcurs del projecte que ha durat aproximadament uns 5 mesos he hagut de prendre decisions sobre com planificar el projecte i quina feina prioritzar per tal d'assolir els objectius del projecte.

12 Treball Futur

A continuació es llista una sèrie de millores que es podrien afegir a la implementació actual:

- Millorar la API Rest
- Utilitzar un segon servidor per tal de fer el processament del vídeo
- Utilitzar una CDN per tal de servir els vídeos
- Millorar la interfície d'usuari
- Implementar aplicacions per Android i iOS per tal de consumir la API Rest com el client web
- Desplegar l'aplicació en un servidor de hosting web dedicat
- Utilitzar TLS per a totes les connexions utilitzant un servei com `Let's Encrypt`

13 Bibliografia

1. Hanson, MD. (2000). Server Management.
2. Sventek, J. (1992). The Distributed Application Architecture.
3. Temple University (s.d.). Introduction to Distributed Systems, Middleware and Client-Server and Peer-to-Peer Systems.
4. Alonso, G., Casati, F., Kuno, H., Machiraju, V. (2004). Web Services.
5. Richardson, L., Ruby, S. (2008). RESTful web services.
6. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A. (2003). Overview of the H. 264/AVC video coding standard .
7. ITU-T (s.d.). Advanced video coding for generic audiovisual services.
8. Apostolopoulos, J.G., Wai-tian, T., Susie, J. (2002). Video Streaming: Concepts, Algorithms, and Systems.
9. (s.d.) Wikipedia.
10. (s.d.) Stackoverflow.
11. (s.d.). What exactly is client side rendering and hows it different from server side rendering. Recuperat de https://medium.freecodecamp.com/what-exactly-is-client-side-rendering-and-hows-it-different-from-server-side-rendering-bd5c786b340d
12. (s.d). Client Side vs Server Side Rendering. Recuperat de http://openmymind.net/2012/5/30/Client-Side-vs-Server-Side-Rendering/
13. (s.d). Single-Page Applications. Recuperat de https://msdn.microsoft.com/en-us/magazine/dn463786.aspx
14. (s.d). SPA vs MPA. Recuperat de https://medium.com/@NeotericEU/single-page-application-vs-multiple-page-application-2591588efe58
15. (s.d). SPA vs MPA. Recuperat de http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications

16. (s.d). SPA vs MPA. Recuperat de http://www.eikospartners.com/blog/multi-page-web-applications-vs.-single-page-web-applications

17. (s.d). Angular 2 concerns. Recuperat de https://www.infoq.com/news/2015/03/angular-2-concerns-addressed

18. (s.d). Angular. Recuperat de http://angular.io

19. (s.d). BackboneJS. Recuperat de http://backbonejs.org/

20. (s.d). React Recuperat de https://facebook.github.io/react/

21. (s.d). Why React. Recuperat de https://facebook.github.io/react/blog/2013/06/05/why-react.html

22. (s.d). Facebook MVC Flux. Recuperat de https://www.infoq.com/news/2014/05/facebook-mvc-flux

23. (s.d). Reasons to migrate to mariaDB. Recuperat de https://seravo.fi/2015/10-reasons-to-migrate-to-mariadb-if-still-using-mysql

24. (s.d). MongoDB. Recuperat de https://www.mongodb.com/compare/mongodb-mysql

25. (s.d). DASH vs HLS. Recuperat de https://bitmovin.com/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/

26. (s.d). Why YouTube & Netflix use MPEG-DASH in HTML5. Recuperat de https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/

27. (s.d). Twitch API. Recuperat de https://dev.twitch.tv/docs/v5/guides/using-the-twitch-api

28. (s.d). Apache vs Nginx. Recuperat de https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations

29. (s.d). Apache vs Nginx. Recuperat de https://www.nginx.com/blog/nginx-vs-apache-our-view/

30. (s.d). Web Server. Recuperat de https://w3techs.com/technologies/overview/web_server/all

31. (s.d). React. Recuperat de https://facebook.github.io/react/docs/

32. (s.d). Redux. Recuperat de http://redux.js.org/docs/introduction/

33. (s.d). Django. Recuperat de https://docs.djangoproject.com/en/1.11/

34. (s.d). Djano Rest Framework. Recuperat de http://www.django-rest-framework.org/

35. (s.d). dash.js. Recuperat de https://github.com/Dash-Industry-Forum/dash.js

36. (s.d). RESTful API Design, Best Practices in a Nutshell. Recuperat de https://blog.philipphauer.de/restful-api-design-best-practices/

14 Annexos

14.1 Com DASHificar un vídeo

A continuacó es mostra com es "dashifica" un vídeo MP4 anomenat 1.mp4.

14.1.1 Una sola qualitat

```
ffmpeg -i 1.mp4 -c:v libx264 -b:v 4000k -r 24 -x264opts
    keyint=48:min-keyint=48:no-scenecut -profile:v main
    -preset medium -movflags +faststart -c:a aac -b:a
   128k -ac 2 out1.mp4
MP4Box -dash-strict 4000 -rap -bs-switching no -profile
    dashavc264:live -out 1.mpd out1.mp4#audio out1.mp4#
   video
```

14.1.2 Multiples qualitats

```
ffmpeg -y -i inputfile -c:a aac -ac 2 -ab 128k -c:v
   libx264 -x264opts 'keyint=24:min-keyint=24:no-
   scenecut' -b:v 1500k -maxrate 1500k -bufsize 1000k -
   vf "scale=-1:720" outputfile720.mp4
ffmpeg -y -i inputfile -c:a aac -ac 2 -ab 128k -c:v
   libx264 -x264opts 'keyint=24:min-keyint=24:no-
   scenecut' -b:v 800k -maxrate 800k -bufsize 500k -vf
   "scale=-1:540" outputfile540.mp4
ffmpeg -y -i inputfile -c:a aac -ac 2 -ab 128k -c:v
   libx264 -x264opts 'keyint=24:min-keyint=24:no-
   scenecut' -b:v 400k -maxrate 400k -bufsize 400k -vf
   "scale=-1:360" outputfile360.mp4
MP4Box -dash-strict 4000 -rap -bs-switching no -profile
    dashavc264:live -out 1.mpd out1.mp4#audio out1.mp4#
   video
```

15 Manual d'instal·lació

Per tal de poder desplegar el servidor fan falta les següents dependències:

- Python3
- pip (Python package index)
- uwsgi
- NodeJS
- NPM (Node package manager)
- Nginx
- MariaDB (o MySQL)

A l'arrel del projecte hi han dos directoris `client` i `server`. A més s'inclou la configuració de l'Nginx (`nginx.conf`). Primer s'explicarà els passos a seguir per tal de desplegar el servidor i a continuació s'explicarà els passos a seguir per tal de generar el `build` del client i on s'ha de posar.

15.1 Servidor

Utilitzar el fitxer de configuració de l'Nginx:

```
# cp nginx.conf /etc/ngnix/
```

Instal·lar el Django:

```
pip install django djangorestframework django-cors-
    headers MySQL-python
```

Crear els directoris videos, thumbs, live i api a l'arrel de l'Nginx. A més copiar el contingut de la carpeta server a dins de la carpeta api:

```
# mkdir /usr/share/nginx/html/api
# cp -r server/* /usr/share/nginx/api/
# cd /usr/share/nginx/html
# mkdir videos thumbs live
```

15.2 Client

El client s'ha de construir (build) i posar a l'arrel de l'Nginx:

```
# cd client
# npm install
# npm run build
# cp -r build/* /usr/share/nginx/html/
```

15.3 Execució

L'Nginx ha d'estar en execució. Com que s'utilitza el uwsgi pel Django fa falta crear un servei per tal de que s'executi. Alternativament es pot anar a l'arrel del projecte Django (/usr/share/nginx/html/api/) i executar la següent comanda:

```
# uwsgi --ini django.ini
```

L'aplicació ja està disponible a la url http://localhost