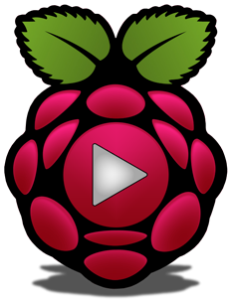


R A S P M E D I A

PROJECTE DE TECNOLOGIES DE LA INFORMACIÓ

Participants:

Álvarez Guerra, Adrián
Capellas Careta, Carles
Domínguez Purificación, Antonio
Masriera Quevedo, Miquel

Tutor:

Tous Liesa, Rubén

26 de Maig de 2014

Índex

1 Introducció

1.1 Motivació

Hem volgut aprofitar la oportunitat de treballar en un projecte de Tecnologies de la Informació per desenvolupar un producte que poguéssim utilitzar fàcilment en un entorn domèstic i sense requerir de coneixements tècnics. D'aquesta manera un cop el sistema estigués acabat podríem fer-lo servir en el nostre temps lliure i mostrar el resultat del nostre esforç a qualsevol persona aliena al món de les Tecnologies de la Informació.

Adicionalment volíem poder aplicar alguns dels conceptes de xarxes, sistemes i aplicacions que hem après al llarg de l'especialitat al projecte, però sense deixar de banda tecnologies recents com son les SBC (Single-Board Computer) i llenguatges de programació nous per a nosaltres (inicialment dubtavem entre Python i Node.js).

Així doncs, tot i que al principi no tinguéssim massa clar quin seria el resultat final, el que si que sabíem amb certesa era que volíem tenir un servidor propi (a poder ser un SBC de baix cost, abans que un servei remot com AWS), una aplicació mòbil, diferents dispositius, i orientar tot el projecte al voltant de la comunicació entre les diferents parts.

1.2 Desenvolupament

Per desenvolupar el projecte vam decidir dividir-lo en cinc apartats diferenciats: el descobriment de dispositius, l'aplicació Android, el codi del servidor, la reproducció de continguts i la configuració de la Raspberry Pi. D'aquesta manera cada membre del grup podia anar avançant amb l'apartat del qual era responsable i un cop enllestit unificar-ho tot en el producte final.

A la pràctica però no hem pogut realitzar el desenvolupament de forma aïllada ja que molts aspectes requerien de sincronització entre els diversos apartats i, per tant, a l'hora de treballar hem hagut de desenvolupar algunes parts de forma conjunta.

Pel que fa al desenvolupament de cada apartat en concret, hem invertit un temps inicial per aprendre com funcionaven els llenguatges/tecnologies que faríem servir. Com per exemple, en el cas del servidor, vam haver d'aprendre el funcionament de Node.js, que ens va fer perdre més temps de l'esperat en un principi, però a la llarga ens hem vist recompensats per les facilitats que ens ha aportat aquest nou llenguatge.

1.3 Conclusions

Hem assolit tots els objectius que ens vam proposar inicialment i hem disposat de temps suficient per afegir algunes ampliacions a les funcionalitats del projecte, com és per exemple, una pàgina web que permet gestionar el contingut del servidor desde qualsevol tipus de dispositiu que es pugui connectar a la xarxa i disposi d'un navegador amb suport a HTML5 (p.ex: ordinadors portàtils, equips de sobretaula, dispositius mòbils amb sistemes

operatius diferents d'Android, etc.).

També ens hem familiaritzat amb tecnologies que no aprenem a la universitat tals com Android, Node.js, Javascript, programació web, etc. de les quals alguns en teníem unes nocions bàsiques, però que cap de nosaltres havia tingut la oportunitat d'aplicar en un projecte gran i menys de relacionar-les entre elles.

Cal remarcar que hem tingut sort escollint les tecnologies que hem utilitzat, ja que vam apostar per aquestes una mica a cegues, sense saber com seria la integració entre elles ni conèixer el temps que necessitaríem per familiaritzar-nos-hi i al final el desenvolupament amb aquestes ens ha resultat molt menys problemàtic del que ho hagués pogut ser amb d'altres.

Hem arribat més lluny del que en un principi vam pensar que arribaríem en la implementació de funcionalitats del sistema i el fet de no necessitar cap element extern (p.ex: un servidor accessible desde qualsevol punt d'internet) fa que poguem utilitzar-lo sense cap cost i seguir-ne ampliant les funcionalitats pel nostre compte tot i deixar el projecte finalitzat per ara.

Per concloure aquesta introducció volem acabar dient que estem molt contents amb el resultat del projecte, tant pel producte que hem desenvolupat (que pràcticament és un element comercialitzable) com per l'experiència en si que ha representat el disseny, la gestió i el desenvolupament d'aquest.

2 App Android

L'aplicació nativa per dispositius Android és un dels punts fonamentals del projecte, ja que serà la interfície principal amb la qual l'usuari podrà interactuar amb el sistema. Per aquesta raó, el nostre principal objectiu a l'hora de dissenyar-la va ser oferir una interfície senzilla on la interacció fos el màxim intuïtiva possible i amb una estètica atractiva.
A continuació, descriurem les diferents pantalles de l'aplicació, el seu funcionament i una mica per sobre les classes que ens ofereix el SDK d'Android per poder implementar-les. No ens centrarem massa en aquest últim punt ja que la implementació és bastant complexa i preferim focalitzar-nos en el disseny/interacció amb l'usuari.

2.1 Pantalla Inicial

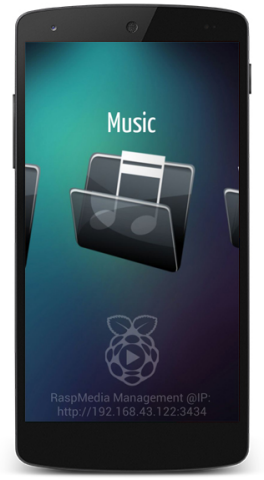
En aquesta primera pantalla se li mostra a l'usuari el logo de RaspMedia. Tot i això, no es tracta d'una pantalla "publicitària"sense sentit. En *background* es realitza el procés de descobriment de dispositius, que explicarem a continuació, per enregistrar la adreça IP de la Raspberry Pi. Un cop finalitzar aquest procés, apareix un botó *Start* per accedir al menú principal de l'aplicació.

2.1.1 Descobriment de dispositius

Aquest ha sigut un dels aspectes on més possibilitats i variants hem contemplat. El nostre objectiu des de bon principi era que l'usuari hagués de fer el mínim esforç per utilitzar el sistema i, per tant, això requeria un descobriment dels dispositius on la resolució de les adreces IP es fes de manera automàtica i que l'usuari no les hagués d'introduir manualment.
Una de les primeres opcions que vam intentar implementar era una resolució de l'adreça IP mitjançant la MAC de la Raspberry PI. Com l'adreça MAC és coneguda i única podíem

escanejar la xarxa local en busca d'aquesta i, un cop trobada, podríem registrar fàcilment l'adreça IP de la Raspberry. El problema amb el que ens vam trobar es que per raons de seguretat obvies Java no permet visualitzat les adreces MAC de la resta de dispositius de la xarxa.
Amb aquest escenari vam decidir realitzar peticions HTTP a tots els dispositius actius a la nostra xarxa local. Aquestes peticions es realitzen sobre un port poc habitual a la resta d'aplicacions i l'App espera durant 50ms per la resposta del servidor. Si el codi de resposta equival a 200 llavors registrem l'adreça IP del servidor per utilitzar posteriorment. D'aquesta manera ens assegurem que només el servidor de la Raspberry Pi contesti amb les condicions anteriors a l'aplicació i, per tant, obtenim l'adreça IP de la Raspberry a la xarxa.
A més, per optimitzar aquest procés, guardem l'adreça IP trobada a una estructura de dades persistent de l'aplicació per tal de tenir-la enregistrada en futures execucions i estalviar-nos l'escaneig de la xarxa.

2.2 Menú Principal



Seguint amb la filosofia d'oferir una interacció intuïtiva, hem implementat un menú rotatori on es pot seleccionar entre les diverses seccions lliscant el menú a dreta i esquerra. Per fer-ho, hem utilitzam l'element *Fragment* que ens ofereix el SDK d'Android.
Les seccions disponibles són: *Music, Photos, Videos* i *Upload*.
A més, a la part inferior de la pantalla es mostra l'adreça IP enregistrada a la pantalla anterior per tal de poder accedir a la web de gestió del nostre projecte.

2.3 Música

A l'apartat de Música podem veure un llistat de les cançons disponibles a la Raspberry Pi. Per representar aquesta informació hem utilitzat l'element *ListView* on es mostra el títol de la cançó, el nom de l'artista, l'any de publicació i la imatge de portada de l'àlbum al que pertany. En el cas que alguna cançó en concret no tingui una imatge adjunta, es mostrarà una imatge preestablerta. Per recopilar les dades de cada cançó, l'aplicació efectua peticions HTTP al servidor i aquest li retorna una estructura *JSON* amb el següent esquema:

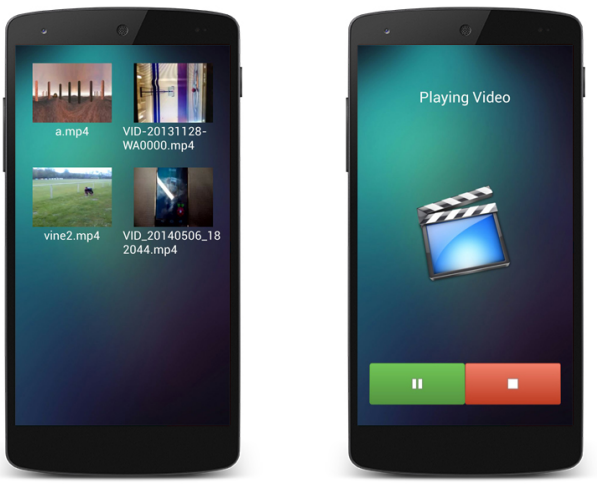
$$jsonSong = name,title,artist,year,pic$$

Els elements *name, title, artist* i *year* són del tipus *String* i el *pic* es tracta de la conversió en *base64* de la imatge.
Per últim, si l'usuari prem qualsevol cançó de la llista, apareixerà la pantalla de reproducció i s'enviarà una petició al servidor amb el nom de la cançó seleccionada per que la reprodueixi. En aquesta pantalla de reproducció hi han dos botons de *play/pause* per que l'usuari pugui controlar la reproducció i un botó de *stop* per finalitzar la reproducció i tornar a la pantalla amb la llista de cançons.

2.4 Fotos

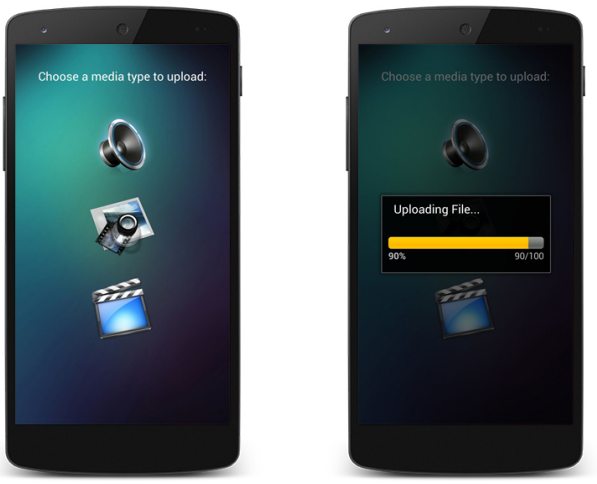
En el cas de las fotografies hem optat per l'element *GridView* del SDK d'Android amb dues columnes amb una imatge cada una. Per obtenir les imatges disponibles a la Raspberry Pi, l'aplicació realitza una petició HTTP al servidor i aquest li respon amb la conversió *base64* d'una miniatura de la imatge real.
Si l'usuari prem alguna de les imatges, s'entrarà a la pantalla de reproducció de fotografies. En aquest cas, l'usuari pot interaccionar amb quatre botons: anterior i següent per canviar entre les fotografies emmagatzemades, maximitzar/minimitzar per veure la fotografia a pantalla completa o normal i, per últim, el botó de *Stop* per finalitzar la reproducció de fotografies i tornar a la pantalla de selecció.

2.5 Vídeos



Tal i com fèiem a l'apartat de fotografies, hem seguit el mateix disseny amb *GridView* però en aquest cas es mostra també el nom del vídeo, ja que només una fotografia d'un *frame* aleatori del mateix pot no resultar suficient per la seva identificació.
Si l'usuari prem algun vídeo, apareixerà la pantalla de reproducció de vídeos. Aquesta, tal i com succeeix amb la de reproducció de música, té dues interaccions: botó de *play/pause* per que l'usuari pugui controlar la reproducció i un botó de *stop* per finalitzar la reproducció i tornar a la pantalla amb la llista de vídeos.

2.6 Pujar Arxius



Per últim, hem afegit una secció a l'aplicació on l'usuari pot pujar les cançons, fotos o vídeos que tingui emmagatzemades al seu *smartphone*. Com es pot observar a la imatge superior, l'usuari pot escollir entre els tres tipus d'arxius a pujar i, segons el tipus seleccionat, s'obrirà la galeria pertinent per que pugui seleccionar l'arxiu desitjat. Un cop escollit, apareixerà una barra de progrés que indica el percentatge completat en la pujada de l'arxiu. Aquest últim element l'hem implementat mitjançant un *ProgressDialog* del SDK d'Android.
Un cop finalitzada, podem tornar enrere als apartats mostrats anteriorment i veure com realment l'arxiu s'ha pujat correctament.

3 Aplicació servidor i web

Aquesta secció tracta sobre la aplicació servidor que corre sobre la Raspberry, i amb la que interactuen tant la aplicació Android com una senzilla web a la que es pot accedir per gestionar el contingut multimèdia del servidor.
La intenció és donar una idea general les tecnologies utilitzades com la motivacó per a fer-les servir, el que han suposat de cara al desenvolupament del projecte.

3.1 Pàgina web

Tot i que la idea és que l'usuari utilitzi únicament la aplicació Android, vam decidir que seria una bona idea afegir una web, que servís per veure de quins continguts disposa el servidor, i poder pujar i esborrar arxius des d'un ordinador. Vam veure aquesta necessitat ja que la majoria de contingut que ens interessaria reproduir en un media center són pel·lícules i música, que normalment guardem a l'ordinador de casa.I el fet de passar-los al servidor manualemnt és incomode, sobretot per a usuaris poc experimentats.
La direcció de la web es mostra en tot moment (excepte quan s'està reproduint continguts) a la part inferior de la aplicació Android, com ja hem vist a la secció anterior.
Teníem la intenció de fer servir un domini, ja que sempre és més user-friendly, però ja que la direcció del servidor no és fixe, vam decidir dedicar-nos a altres aspectes del projecte i deixar això com a un futur extra, que al final no hem acabat implementant.
La web és molt senzilla, es divideix horitzontalment en dues parts, la de la esquerra és un formulari HTML que permet pujar arxius de la mateixa manera que la aplicació mòbil. I la de la dreta llista tots els arxius dels que disposem, dividits en tres categories, i ens permet seleccionar els que volguem per esborrar-los.

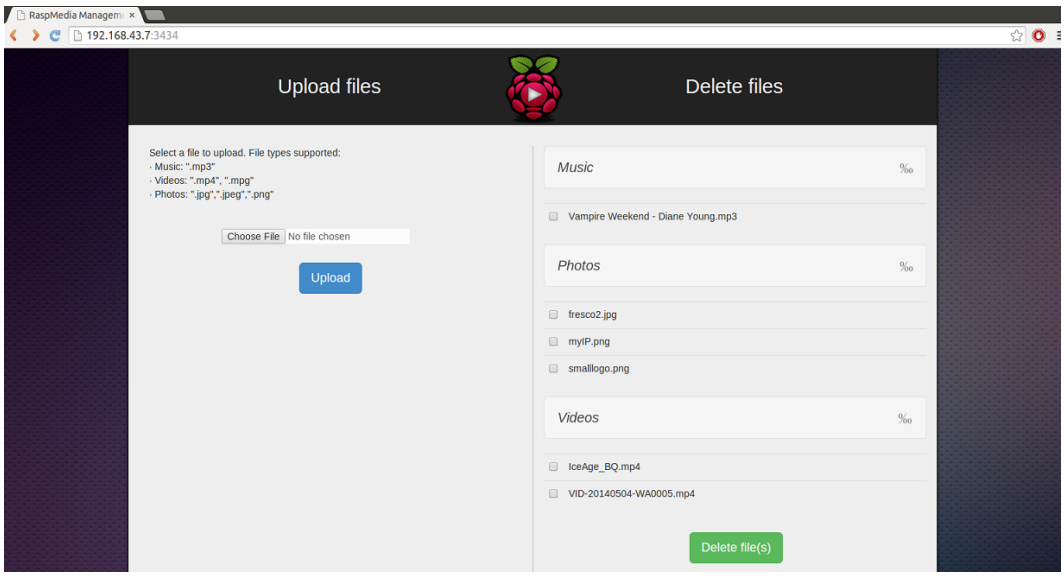


Figura 1: web de gestió de RaspMedia

Pel que fa a laimplementació, és una pàgina web programada en HTML, CSS i JavaScript. Pero hem utilitzat Bootstrap com a framework web, per a donar un millor aspecte, i jQuery i socket.io per a fer la web dinàmica i la comunicació amb el servidor, que es fa a través de websockets, cosa que expliquem a l'apartat 3.3.4. El codi es pot trobar al subdirectori web del directori serverCode. i a `https://github.com/miquelmasriera/RaspMedia/tree/master/serverCode/web`

3.2 Servidor: necessitats i solucions

La aplicació servidor, és bàsicament una aplicació que estarà executant la màquina i que tractarà les peticions que li arribin tant de la aplicació com de la web.
Ja que el hardware del que disposem té unes limitacions importants, i que volíem fer un servidor el més senzill possible però que complís amb les nostres necessitats vam, decidir utilitzar Node.js, abans que altres tecnologies com Apache, Tomcat, PHP o Django.

La idea de la que parteix Node.js és la de programar tot el back-end en Javascript, amb tot el que això comporta, una de les diferencies principals amb altres tecnologies, és que la aplicació corre un sol thread amb un loop d'esdeveniments, això permet que respongui molt bé a l'augment del nombre de connexions que rep el servidor, ja que s'estalvia el procés d'obrir i tancar connexions, i tractar-les individualment. El rendiment també és realment bo per a aplicacions de xarxa i en temps real: la màquina virtual de Javascript que utilitza és la V8 de Google, Node és asíncron i no bloquejant per a operacions d'entrada/sortida. Una altra raó determinat alhora de fer servir Node és la gran comunitat que té al darrera, hi ha milers de paquets que no només programa la comunitat, sinó que pràcticament tots son open-source, que ofereixen APIs per a qualsevol cosa que es pugui necessitar.

a El nostre servidor, respon a les peticions http get i post que li envien els altres actors del sistema, és a dir, la aplicació mòbil i la web.
Això significa que per a cada petició que li arribi (llistar música, reproduir arxiu, pujar arxiu ...) caldrà una funció específica al servidor, que s'encarregui de fer la feina, i retornar el feedback necessari. Totes les dades que viatgen del servidor al client (llistes, miniatures, etc) ho fan en objectes JSON, que ens permet seguir amb la idea de fer servir tant Javascript com sigui possible, i que segueixi sent molt senzill. A banda de rebre dades i enviar dades, el servidor fa molt més, ja sigui crear subprocessos per a executar altres programes com el reproductor o altres per a tractar imatges i vídeos per a generar miniatures, recórrer directoris, guardar arxius o interactuar amb el reproductor.

A continuació explicarem com hem dividit les tasques que ha de dur a terme el servidor en grups i com les hem implementat.

3.3 Servidor: implementació

Aquí explicarem en detall com hem programat cada una de les funcions del servidor[1] i perquè. El servidor esta programat amb Node.js, el principal paquet que fem servir és Express, que és el framework web més famós i usat per a aplicacions Node. La aplicació servidor crea una instància de express que inicialitza un servidor http al port 8000, i és sobre aquesta, sobre la qual afegim les funcions associades a cada petició. La aplicació té un arxiu main.js, que és el que conté tots els moduls en els que hem dividit la aplicació. per executar-la només cal executar aquest arxiu.

```
node main.js
```

3.3.1 Sistema de fitxers

Correspon a l' arxiu fsManager.js de la aplicació i fa servir el modul fs pròpi de Node.js. La idea és que crei i verifiqui que l'estructura del sistema de fitxers és la que toca. En el directori del servidor hi ha un directori per al codi anomenat serverCode, i tres directoris més, un per a cada tipus de contingut (imatges, videos i música) que és on es guardaran els arxius que es pugin al sistema, i els que es llegiran per a obtenir llistes i reproduir arxius.

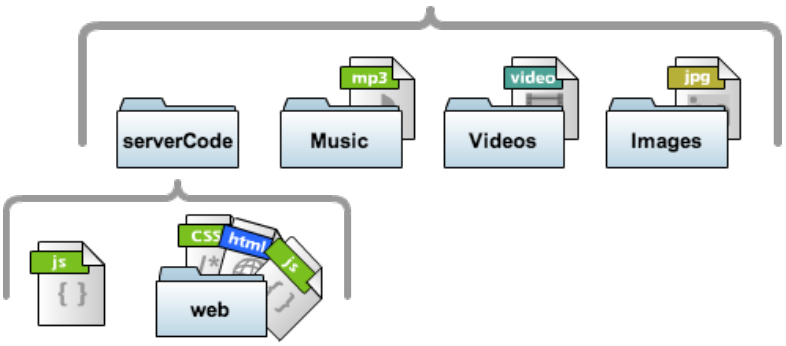


Figura 2: Esquema del sistema de fitxers del servidor

És el primer que s'executa quan arranca el sistema, comprova que hi hagi el directoris i subdirectoris que toca i que tinguin el nom adequat. En el cas de que en falti algun, es creen automàticament perquè el sistema pugui funcionar correctament.
En un principi pensàvem que tindria que ser molt més complexe, perquè la nostra intenció era tractar cada tipus d'arxiu per separat, i fer servir bases de dades tant per arxius com per sessions d'usuari, cosa que al final no ha sigut necessari.
Tots els paths que utilitzem a node son absoluts, ja que sinó, no es podria executar la palicació des de fora del seu propi directori. Ens hem trobat mab aquest problema ja que nosaltres la executem amb l'script d'arrancada del sistema.

[1]El codi font es troba a `https://github.com/miquelmasriera/RaspMedia/tree/master/serverCode`

3.3.2 Pujar arxius

En l'aplicació es correspon amb l'uploader.js i fa servir els moduls express i imagemagick. Aquesta part de la aplicació s'encarrega de rebre els fitxers que li arriben via web o aplicació Android, i els guarda al directori que toqui depenent del tipus d'arxiu que sigui. Els arxius arriben en una (o varies) peticions post, el servidor, comprova el tipus d'arxiu i decideix on emmagatezemar-lo. En el cas que es tracti d'una imatge, és just després de que sigui pujada que es crea una miniatura, i es guarda en un subdirectori del de imatges per a les miniatures, això fa que tot el procés de pujada sigui una mica més lent, però sacrificant aquests segons aquí, podem obrir la llista d'immatges practicament a l'instant. Anteriorment creavem les miniatures en el moment de seleccionar la opció de fotos de la aplicació, que ha de enviar a la aplicació una llista amb totes les miniatures de les fotos, i el problema que teníem, és que a mesura que creixia el nombre de fotos, també creixia considerablement el temps que trigava en enviar les miniatures.
Aquesta va ser una de les primeres funcionalitats que vam implementar, i ja que no vam trobar ninguna bona manera de fer-ho amb Node.js i els paquets que porta per defecte, vam esta provant bastants mòduls, i al final ens vam decantar per Express ja que era el més simple i senzill. En conseqüència d'això hem fet servir Express per a tota la aplicació, cosa que ha sigut un encert.

3.3.3 Gestor de llistes

ListManager.js en el directori del servidor, i utilitza els moduls Express, imagemagick, musicmetadata i base64.
La principal funcionalitat d'aquesta part del codi, és respondre a les peticions de llistes d'arxius que conté el servidor multimèdia. Quan un usuari prem una carpeta de la aplicació (música, vídeos o imatges), es genera una petició cap al servidor, i aquest respon amb la llista de fitxers demanats.
Quan rep una petició el servidor obre el directori corresponent, i crea una llista d'elements JSON per a les dades de cada arxiu. En el cas de la música, es fa servir el mòdul musicmetadata per a obtenir les dades que es pugui d'una cançó (en el cas que es puguin llegir), si es pot obtenir una imatge, se codifica en base64 abans de envair-la a l'smartphone.

```
var jsonSong = { name:"-", title:"-", artist:"-", year:"-", pic:"-" }
```

Si el que volem és una llista dels vídeos, per agafar una imatge, executem el programa ffmpeg i obtenim un fotograma dels primers segons del video, i posteriorment la passem a base64.

```
var jsonVideo = { name:"", base64:"" }
```

I en el cas de les imatges, el procediment és molt semblant al dels vídeos, per a cada imatge del subdirectori de miniatures, la passem a base64 i posteriorment la afegim a la llista.

```
var jsonPhoto = { name:"", base64:"" }
```

Aquesta part de la aplicació ens va portar molta feina, ja que tot i que no és excessivament complexa, implica estar enviant dades entre el client i el servidor, crear objectes, fer servir mòduls i programes externs... En definitiva ens va portar temps de programar, i sobretot de decidir com faríem les coses. Per exemple, al principi en comptes de JSON fèiem servir arxius XML, però vam decidir canviar a Javascript ja que és més fàcil i més compatible amb els mòduls de Node.js.
També en aquesta part és on hem notat molt les millores que suposa l'overclock, sobretot en els temps en generar les miniatures de fotos i vídeos

3.3.4 Gestor d'esdeveniments de la app Android i la web

Aquí parlarem una mica del perquè hem decidit implementar D'aquesta manera la gestió dels esdeveniments entre client (mòbil o web), el servidor i el reproductor (que es troba al servidor també), el cas de la app Android s'explica en profunditat a l'apartat 5.4 d'aquest document.
El codi que implementa la gestió d'events és a eventManager.js.
El problema a resoldre era que els esdeveniments per al que fa a reproducció de continguts que genera el mòbil, tenen que passar a través del servidor fins al reproductor. La manera més correcte i senzilla que hem trobat per a que aquesta comunicació es dugui a terme és que el mòbil i el servidor es comuniquin de la mateixa manera que amb la resta de la aplicació, és a dir, amb peticions i respostes HTTP,i la comunicació sevidor-repoductor sigui utilitzant un modul de Node dissenyat específicament per a controlar el reproductor OMXPlayer anomenat omxctrl per a musica i vídeo, que proporciona una API molt senzilla i que funciona molt bé per a aquest propòsit, i gpicview per a les imagtes.
Per a alguns esdeveniments que requereixen interacció amb el teclat, les simulem, executant un procés de xdotool, el programa de Linux. Per posar un exemple, si el client (aplicació Android) genera una petició dient que vol reproduir el video1, el servidor la rep, i executa una funció de omxctrl (omx.play(video1)), i reotorna un resposta de confirmació al mòbil.
Tots els esdeveniments que associats a la web es troben a webManager.js.
Pel que fa a la comunicació entre la web i el servidor, hem decidit basar-la en websockets. Ja que un dels mòduls més famosos de Node.js és socket.io, que ofereix una API a alt nivell per a tractar comunicació en temps real. El servidor es posa a escoltar pel port 3434 i la web fa les peticions a aquest port.

4 Reproductor i gestió d'esdeveniments

4.1 Selecció del reproductor de vídeo i música

Durant tot el transcurs del projecte, mentre fèiem les parts anteriors, anàvem pensant què seria el més adequat per reproduir els continguts desitjats.

La opció que primer vam considerar va ser utilitzar reproductors de Linux com VLC, Rythmbox, etc... Però si ho fèiem així no podríem controlar la reproducció de vídeos, música o fotografies. Per permetre el control de la reproducció necessitàvem una comunicació entre el servidor i el reproductor, es per això que vam decidir-nos a fer un reproductor en HTML llançat des d'un navegador web.

Els avantatges que proporcionava aquest tipus de reproductor sobre els altres eren justament els que necessitàvem:

- La opció de poder personalitzar una interfície al nostre gust
- La possibilitat de comunicar servidor i reproductor
- Un major grau de portabilitat, etc ...

Una vegada feta aquesta decisió vam començar a buscar llibreries que complissin els nostres requeriments. La primera llibreria que vam trobar va ser *video.js*, semblava una bona elecció però a l'hora de començar a treballar amb ella van aparèixer els problemes: dificultats per treballar, les coses no anaven bé, controlar el reproductor no era tan fàcil com semblava només amb el primer cop d'ull, etc... La següent llibreria que vam trobar va ser *MediaElement.js* i ens vam adonar que realment semblava molt més fàcil d'utilitzar que no pas l'anterior, o això va ser la primera impressió que vam tenir, ja que a la seva pàgina principal es veuen un quants codis d'exemple que ens van semblar molt senzills.

4.2 Primer contacte amb MediaElement.js

A la poca estona de començar a treballar amb *MediaElement.js* ja vam notar que era més fàcil i simple, feien falta menys instruccions per fer el mateix. El funcionament del reproductor i el control d'aquest era tan senzill com utilitzar el tag de video d'HTML i donar-li una ID al reproductor per després poder treballar amb ell.

```
<video src=" " type="video/mp4" id="videoplayer"></video>
```

Una vegada creat el reproductor i amb la ID *videoplayer*, a la part de script del fitxer HTML es pot crear un objecte gràcies a la llibreria de *MediaElement.js*:

```
var player = new MediaElementPlayer("#videoplayer",{
        startVolume: 0.5
});
```

Quan tenim aquest objecte *player* ja podem treballar amb ell en JavaScript i JQuery. Uns del exemples de les funcions que tenim gràcies a *MediaElement.js* són:

```
player.setSrc(/* path to file */);
player.play();
player.pause();
player.enterFullScreen();
player.setVolume(0.5);
player.paused(); // returns a boolean
```

Ara que ja sabíem com controlar el reproductor ens faltaba comunicar-lo amb el servidor. Per aconseguir això vam utilitzar *socket.io*, un mòdul de *node.js* el qual ens va permetre connectar el codi del servidor escrit en *node.js* amb la part de JavaScript del fitxer HTML.

4.3 Visualització d'imatges

Per a la visualització d'imatges es pot fer amb el tag d'img d'HTML però s'ha de definir un *style* per redimensionar la foto al tamany de la pantalla.

4.4 Gestió d'esdeveniments

Per aconseguir la connexió, els dos fitxers havien de crear un objecte socket al que els dos es connectaven. El servidor el crea i el reproductor només s'ha de connectar:

```
var socket = io.connect("http://localhost:xxxx");
```

Un cop connectats, quan el servidor rep una petició del dispositiu Android aquest pot simplement fer:

```
socket.emit("nom_event");
```

I a la part del reproductor és suficient amb tenir:

```
socket.on("nom_event", function () {
        /* accio a realitzar */
});
```

També és possible intercanviar missatges entre el servidor i el reproductor, en el nostre cas això ho utilitzem per indicar el path del fitxer a reproduir:

```
// Servidor:
socket.emit("changesource", "../music/sample1.mp3");
// Reproductor:
socket.on("changesource", function (message) {
        player.setSrc(message);
        player.play();
        player.enterFullScreen();
});
```

4.5 Resultat final

Tot això funcionava perfectament amb el resultat de la figura 1, però a l'hora d'integrar-ho amb la Raspberry Pi va haver-hi problemes ja que el Chromium de Raspberry Pi no suportava HTML5. Vam buscar un altre navegador web que si el suportés i vam trobar l'Epiphany. Amb aquest reproductor el reproductor que havíem fet funcionava, però anava molt lent: el navegador trigava aproximadament 30 segons en obrir-se i els videos es reproduien a 2-3 FPS.

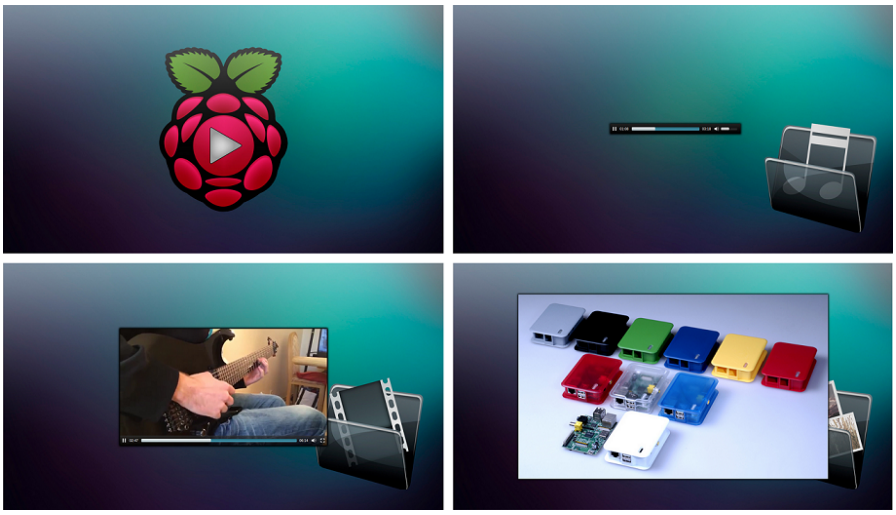


Figura 1: Reproductor HTML

Per solucionar això vam tenir que buscar una alternativa. Buscant i buscant ens vam decidir per OMXPlayer ja que, a més a més, vam trobar també un mòdul de *node.js* anomenat *omxcontrol* per poder controlar-ho. De forma que, mantenint els esdeveniments[2] provinents del dispositiu Android, ho vam adaptar tot per utilitzar-ho amb OMXPlayer i omxcontrol. És a dir, vam tenir que abandonar MediaElement.js, socket.io i la mínima personalització del aspecte que això oferia. Ara ja teníem solucionat la reproducció de video i música, ens faltava la visualització d'imatges. Per aconseguir-ho vam optar per *GPicView* el qual és un simple visor d'imatges, sense cap mòdul per poder controlar-ho remotament, així que vam tenir que fer ús de l'eina *xdootool*, un paquet per linux que permet llançar comandes que simulin la pulsació d'una tecla. El seu funcionament és molt senzill, les seves comandes són del format:

```
xdotool key <tecla_a_pulsar>
```

De forma que podem posar GPicView a pantalla completa amb:

```
xdotool key F11
```

D'aquesta forma podiem controlar l'OMXPlayer tant per música com per vídeo a través del mòdul *omxcontrol* i el GPicView a través de *xdotool*. De forma que al codi del servidor podem fer coses com:

[2]El codi font es troba a `https://github.com/miquelmasriera/RaspMedia/blob/master/serverCode/eventManager.js`

```
/* OMXPlayer */
var omx = require('omxctrl');
omx.play(/* path to file */);
omx.pause();
omx.stop();
```

Amb totes aquestes decisions fetes, només va quedar posar-ho tot a funcionar, de forma que la part del codi del servidor que s'encarrega de la reproducció necessita tres parts:

- Importació de mòduls
- Declaració dels paths on estàn els fitxers i declaració dels reproductor
- Funcions que responen a les peticions del dispositiu Android

Amb tot això ja vam aconseguir la reproducció dels formats proposats i el control complet de tots ells. De forma que el resultat final és:

- Reproductor OMXPlayer que reprodueix música i vídeo
  - Capaç de pausar la reproducció, reanudarla i aturar-la en qualsevol moment
- GPicView que visualitza tot tipus d'imatge
  - Capaç de entrar i sortir de pantalla completa, navegar a través de les imatges i sortir d'aquest mode en qualsevol moment

5 Configuració de l'entorn Raspberry

5.1 Instal·lació del sistema operatiu

El primer pas per configurar l'entorn de la Raspberry Pi consisteix en instal·lar-hi un sistema operatiu. Nosaltres hem escollit la distribució Raspbian (Debian Wheezy) disponible a la pàgina web oficial de Raspberry Pi `http://www.raspberrypi.org/downloads/` i hem treballat amb la versió de Gener de 2014 (l'ultima actualització disponible en començar el desenvolupament del projecte).

Donat que la Raspberry Pi no té cap tipus de memòria interna utilitza una targeta de memòria externa com a suport per l'emmagatzematge; la imatge del sistema operatiu que descarreguem s'haurà de gravar en una targeta SD. Treballant en un entorn Linux cal muntar la targeta SD i copiar-hi la imatge:

wget http://downloads.raspberrypi.org/raspbian_latest

unzip 2014-01-07-wheezy-raspbian.zip

sudo dd if=2014-01-07-wheezy-raspbian.img of=/dev/<TargetaSD>

Per qualsevol dubte en l'instal·lació del sistema podem visitar la pàgina `http://www.raspberrypi.org/documentation/installation/installing-images/README.md`, on s'explica detalladament el procés de instal·lació. Una vegada instal·lat el sistema operatiu ja podem provar-lo. Només cal posar la targeta SD a la ranura de la Raspberry i connectar el cable mini USB per alimentar-la.

Important: Durant la fase de configuració necessitarem tenir la Raspberry connectada a una pantalla amb entrada HDMI i utilitzar un teclat amb sortida USB per navegar per la línia de commandes. Es recomana connectar el hardware abans d'arrencar la Raspberry ja que, per exemple, si no detecta un dispositiu amb HDMI durant la incialització del sistema desactiva la sortida HDMI.

El primer cop que arranquem el sistema apareixarà el menú de configuració (més tard podem tornar a accedir a aquest menú executant la comanda raspi-config amb permisos de root) en el qual podem, per exemple, canviar el fús horari, seleccionar el mode d'inici del sistema (escriptori gràfic o línia de commandes, canviar la contrasenya de l'usuari per defecte, etc.)

El primer cop que entrem al sistema però farem dues coses; assegurar-nos que el protocol ssh està activat per poder accedir remotament a la Raspberry més tard (opció ssh del menú) i expandir el sistema de fitxers (opció expand_rootfs). Un cop fet això caldrà reiniciar la Raspberry i la següent vegada que el sistema arranqui accedirem directament a una línia de commandes on es demanarà identificació.

L'usuari per defecte del sistema operatiu Raspbian és 'pi' i la seva contrasenya, 'raspberry'. Per més informació: `http://www.raspberrypi.org/wp-content/uploads/2012/12/quick-start-guide-v1.1.pdf`.

5.2 Configuració de xarxa

Per donar una millor experiència d'usuari hem decidit utilitzar un connector WiFi USB de manera que la Raspberry no hagi d'estar connectada al router de la xarxa domèstica a través d'un cable. Cal tenir en compte que no tots els connectors son compatibles amb el sistema operatiu Raspbian (en aquesta pàgina hi ha una llista dels que si que ho son; `http://elinux.org/RPi_USB_Wi-Fi_Adapters`).

Per instal·lar el connector WiFi cal connectar-lo a la Raspberry mentres aquesta està apagada. A continuació arranquem la Raspberry i si el dispositiu és un dels connectors compatibles la Raspberry el reconeixarà de forma automàtica.

lsusb: comprovem que el connector WiFi ha estat reconegut.

lsmod: comprovem que el controlador del dispositiu està instal·lat.

iwconfig: comprovem que el connector funciona correctament.

Un cop instal·lat el connector WiFi només falta configurar la nostra xarxa inalàmbrica. La manera més ràpida de fer-ho és modificant el fitxer /etc/network/interfaces i afegir-hi el SSID i la contrasenya de la nostra xarxa.

És probable que a l'instal·lar el connector WiFi el sistema operatiu hagi modificat aquest fitxer afegint-hi un parell de línies pel connector WiFi (interficie wlan0) però si no ho ha fet les entrarem nosaltres. L'aspecte final que ha de tenir el fitxer és aquest:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0


iface wlan0 inet dhcp
        wpa-ssid "ssid"
        wpa-psk "password"
```

A partir d'ara cada cop que arrenquem la Raspberry Pi i estigui dins l'abast de la nostra xarxa inalàmbrica es connectarà al router automàticament i adquirirà una adreça IP dinàmica mitjançant DHCP.

Si tenim previst utilitzar la Raspberry en més d'una xarxa inalàmbrica aquesta configuració no és la més adequada ja que haurem de cambiar el SSID i la contrasenya de la interfície cada vegada que cambiem de xarxa. Fixem-nos però que la interfície tindrà assignats el SSID i la contrasenya de la última xarxa on haguem estat treballant, per tant, no podrem accedir remotament a la Raspberry per fer aquest canvi des de la nova xarxa.

En el cas de voler utiltzar la Raspberry en vàries xarxes podem optar per una altre tipus de configuració que permet emmagatzemar les dades de vàries xarxes inalàmbriques i durant l'arrencada del sistema comprova si alguna d'aquestes xarxes està disponible.

Nosaltres no hem optat per aquesta possibilitat perquè sempre utilitzavem la mateixa xarxa i perquè és una solució més complexa. Per fer-ho cal guardar les dades de les xarxes en el fitxer /etc/wpa_supplicant/wpa_supplicant.conf amb el format següent

```
network = {

    ssid="SSID de la xarxa"
    psk="Contrasenya"
    proto=WPA
    key_mgmt=WPA-PSK

}
```

i modificar el contingut de etcnetworkinterfaces per què tingui aquest aspecte

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
iface wlan0 inet dhcp

wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Per més informació sobre aquest tipus de configuració es pot consultar la pàgina web `http://www.howtogeek.com/167425/how-to-setup-wi-fi-on-your-raspberry-pi-via-the-command-line/`.

5.3 Configuració de l'entorn d'usuari

Per tal que l'usuari no hagi d'interactuar amb el sistema operatiu hem de configurar-lo per tal que a l'arrencar s'identifiqui automàticament i inici el mode gràfic. Els passos a seguir que expliquem a continuació estan detallats en aquesta pàgina web `http://elinux.org/RPi_Debian_Auto_Login`.

Per evitar que l'usuari s'hagi d'identificar a l'inici n'hi ha prou de cambiar la línia de commandes a la qual accedirà; editem el fitxer /etc/inittab i cambiem la línia

```
1:2345:respawn:/sbin/getty 115200 tty1
```

per la següent :

```
1:2345:respawn:/bin/login -f pi tty1 </dev/tty1 >/dev/tty1 2>&1
```

Un cop l'usuari s'identifica de forma automàtica només queda iniciar el mode gràfic. Per aconseguir-ho executarem la commanda startx (amb permisos de root per poder executar qualsevol commanda una vegada iniciat el sistema) cada cop que el sistema arranqui; afegim la commanda següent al fitxer /etc/rc.local:

```
su -l pi -c startx
```

En aquest punt la Raspberry Pi arrencarà en mode gràfic però el que es mostrarà per pantalla serà el clàssic escriptori de qualsevol entorn gràfic de sistema operatiu. La nostra intenció es adaptar aquest entorn d'escriptori per donar-li l'aparença d'una interfície gràfica i estalivar-nos així tot el temps que comporta el desenvolupament d'una interfície gràfica pròpia (temps que hem pogut dedicar a la implementació de les funcionalitats del sistema). Hem aplicat els canvis següents:

- Dissenyar una imatge d'alta resolució i establir-la com a fons de pantalla. La manera més fàcil de fer-ho és iniciar la raspberry en mode gràfic i amb un ratolí connectat, clicar amb la dreta sobre l'escriptori, escollir la opció "Desktop Preferences" i canviar la foto per la nostra.

- Eliminar tots els fitxers de l'escriptori (fent-ne una còpia de seguretat); d'aquesta manera la pantalla quedarà neta i es veurà bé la foto de fons:

  ```
  cp Desktop DesktopOld
  rm -rf Desktop/*
  ```

- Amagar el cursor del ratolí; a l'iniciar el sistema el cursor surt centrat al mig de la pantalla. Per tal que no interfereixi en la visualització de continguts el configurarem per què desaparegui si no s'està fent servir mitjançant l'eina unclutter:

  ```
  sudo apt-get install unclutter
  unclutter -display :0 -noevents -grab
  ```

- Configurar la barra de tasques per que s'amagui automàticament, maximitzant l'area de visualització de continguts. Un cop més la manera més fàcil de fer-ho és des de l'escriptori gràfic, fent clic dret sobre la barra de tasques, seleccionant la opció "Panel Settings" i activar la opció "Minimize panel when not in use"de la pestanya "Advanced".

- Per defecte Raspbian utilitza la sortida de audio analògica; si reproduim música o video a través de la sortida HDMI no s'escoltarà cap so. Per indicar-li al sistema que volem utilitzar la sortida HDMI com a sortida d'audio per defecte cal modificar el fixter /boot/config.txt.

  Concretament cal modificar la línia 'hdmi_drive=1', i donar-li valor 2. Podem trobar més informació sobre aquest fitxer de configuració a `http://raspberrypi.stackexchange.com/tags/config.txt/info`.

5.4 Preparació del servidor

Un cop tenim l'entorn gràfic configurat només falta preparar el sistema per tal que pugui executar el codi del servidor multimèdia a l'arrencar. Com que vam decidir utilitzar el llenguatge de programació node.js ens ha calgut instal·lar l'administrador de paquets de node (que inclou un intèrpret per aquest llenguatge) per poder executar el codi:

```
wget http://node-arm.herokuapp.com/node_latest_armhf.deb
sudo dpkg -i node_latest_armhf.deb
sudo apt-get update
sudo apt-get upgrade
```

Adicionalment el nostre codi fa servir mòduls de node.js que no venen en la instal·lació per defecte de node; també ens ha fet falta instal·lar-los. Hem de tenir en compte que per tal que el codi es pugui executar aquests mòduls han d'instal·lar-se desde el mateix directori on estiguin els fitxers de codi i utilitzar la opció –save per indicar a l'adminstrador de paquets de node que, a part de instal·lar el mòdul, afegeixi les dependències corresponents a l'esquema JSON de l'aplicació.

npm install <NomMòdul> –save

base64: Mòdul utilitzat per transferir fitxers dins de peticions HTTP transformant el seu contingut en una cadena de caràcters base 64.

express@3.5.1: Mòdul utilitzat per executar un servidor de forma senzilla i programar-ne el seu comportament. Utilitzem la versió 3.5.1 perquè alguna de les opcions que té aquesta versió desapareix a les versions successives.

imagemagick: Mòdul utilitzat per tractar les imatges de portada dels arxius de música així com per generar les miniatures de les fotos.

musicmetadata: Mòdul utilitzat per extreure les dades de les etiquetes MP3 dels arxius de música.

ffmpeg: Mòdul necessari per poder reproduir audio.

omxctrl: Mòdul de control del reproductor OMXPlayer, reproductor per defecte del sistema operatiu Raspbian.

A part d'aquests mòduls de node la nostra aplicació també fa servir altres aplicacions que cal instal·lar a la Raspberry:

```
sudo apt-get install ffmpeg

sudo apt-get install imagemagick

sudo apt-get install xdotool
```

A partir d'aquest punt ja es pot executar el codi de servidor i accedir-hi tant des de l'aplicació Android com des de qualsevol navegador web (tot i fa falta l'aplicació Android per conèixer l'adreça a la qual accedir desde el navegador).

L'última cosa que queda per fer és indicar-li al sistema que executi el codi de servidor automàticament a l'arrencar. Aquesta tasca que sembla trivial ens ha portat més problemes dels que tocarien.

La primera idea que vam tenir va ser crear un script que s'encarregués de inicialitzar tot el que fes falta en l'arrencada del sistema i cridar aquest script des de el fitxer /etc/rc.local propi dels sistemes Debian.

No sabem perquè pero la opció d'afegir commandes al fitxer /etc/rc.local no funciona. Descartada aquesta possibilitat vam decidir posar el mateix script d'inicialització al directori /etc/init.d i utilitzar la comanda update-rc.d per generar enllaços al script que farien que aquest s'executes a l'iniciar el run level 2 (que és el run level per defecte).

Igual que abans aquest plantejament tampoc funciona quan es porta a la pràctica. Després d'investigar altres possibilitats vam descobrir que quan s'inicia el mode gràfic de la Raspberry s'executa l'script /etc/xdg/lxsession/LXDE/autostart. A diferència de les alternatives anteriors aquesta funciona i és la que fem servir actualment; només cal afegir una línia en aquest fitxer cridant al nostre escript de inicialització (amb el path absolut).

Pel que fa a l'script d'inicialització, anomenat RaspIni.sh, l'única cosa que fa es executar el servidor. Podriem haver posat directament una crida al codi de servidor al fitxer /etc/xdg/lxsession/LXDE/autostart però quan configuravem aquesta part del sistema encara feiem servir el reproductor HTML i com que la inicialització del sistema constava de vàries crides vam decidir crear un script, que hem mantingut per si més endavant volem tornar a afegir-hi més coses.

5.5 Millorar el rendiment

Per defecte el processador de la Raspberry treballa a 700 MHz. Aquesta freqüència no és suficient per executar l'entron gràfic amb fluidesa; després de fer unes quantes proves hem arribat a la conclusió que augmentant la freqüència a 900 MHz (com a mínim) la velocitat de resposta de la Raspberry augmenta notablement i l'experiència d'usuari millora molt.

Per modificar la freqüència de treball del processador executem raspi-config amb permisos de root i accedim a la opció '7 Overclock'. Dins aquest apartat trobarem un menú amb l'aspecte següent on podem seleccionar la freqüència desitjada: