E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

DISEÑO E IMPLEMENTACIÓN DE UN CANAL DE COMUNICACIÓN DIGITAL (DIGITAL SIGNAGE) CORPORATIVO

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Beatriz de Miguel Pérez
Tutor: Dr. Eduardo Magaña Lizarrondo
Pamplona, 25 de junio 2014

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN

El objetivo del proyecto es diseñar un sistema completo de comunicación digital aplicable a cualquier empresa que permita sustituir la cartelería tradicional (paneles o tablones de anuncios) por pantallas con contenidos informativos o publicitarios dinámicos.

El sistema podrá albergar contenidos de carácter relevante al interior de la organización en la cual se adecue el sistema como, por ejemplo, publicación de información corporativa, formación para empleados, información sobre el uso de instalaciones, instrucciones de trabajo, comunicados, etc.

Para el manejo del sistema y la actualización de contenidos se contará con un panel de gestión web que permitirá gestionar qué contenido se muestra en cada pantalla y en cada momento.

La información podrá presentarse en pantallas no interactivas, que presentan la lista de reproducción programada para ese terminal, o en pantallas táctiles donde el usuario pueda interactuar con el terminal y visualizar cualquier contenido que desee.

PALABRAS CLAVE:

Digital Signage, Cartelería Digital, Concerto, Android, multi pantalla empresarial

Tabla de contenido:

1. Introducción

1.1. Origen del proyecto

El presente proyecto final de grado está realizado durante el periodo de prácticas en una empresa de telecomunicaciones. La empresa está dedicada al desarrollo de aplicaciones móviles y requiere la implantación de un sistema Digital Signage para una PYME.

Se trata de un desarrollo desde cero donde previamente no se cuenta con ninguna referencia. Por ello, se requiere diseñar una solución que se adapte a los requerimientos del cliente.

1.2. ¿Qué es Digital Signage?

El concepto de Digital Signage o cartelería digital 5 se refiere a un sistema de comunicación que permite la distribución y reproducción de contenido digital (videos, imágenes, publicidad y todo tipo de información) en una red de pantallas. Estas pantallas pueden estar en una única ubicación o en distintos lugares y pueden ser de todo tipo, ya sean proyectores, vallas digitales, paneles táctiles, pantallas de plasma, etc.

Digital Signage permite mejorar la presentación y promoción de productos, facilita la interacción con los contenidos y la actualización de los mismos. Es por ello que cada vez más está sustituyendo a la cartelería tradicional donde el contenido es estático.

Estas redes se pueden encontrar en pequeños comercios, hoteles, restaurantes, lugares de ocio, entornos corporativos, e instituciones tales como centros de salud, entre muchos otros.

1.3. Objetivos

Los objetivos principales fijados en la realización del presente Trabajo Fin de Grado son los siguientes:

- Analizar posibles herramientas ya existentes de Digital Signage para el desarrollo del sistema.
- Definir dos tipos de pantallas donde se visualizará el contenido: pantallas no interactivas y pantallas táctiles. En las no interactivas, la información se presentará en una lista de reproducción programada para ese terminal. En las pantallas táctiles, donde el usuario puede interactuar con el terminal, el usuario podrá elegir qué contenido desea visualizar.
- Definir la arquitectura del sistema concretando los dispositivos a utilizar para la visualización en cada tipo de pantalla.
- Implementar un panel de gestión de contenido web en el que se pueda elegir el contenido que se visualiza en cada pantalla y en cada momento.
- El panel web debe permitir subir archivos de tipo PDF e imágenes.

- El servidor debe ser capaz de elaborar gráficos a partir de ficheros CSV enviados por el ERP de la entidad e incorporarlos al sistema. Dichos gráficos deben actualizarse de forma automática ante cambios en los CSV.
- Tener un producto real utilizable para el cliente final. Interaccionar con el cliente de la empresa para adaptar el producto a sus necesidades.

1.4. Estado del arte

En el mercado actual ya existe una gran variedad de empresas dedicadas a elaborar soluciones Digital Signage. Cada empresa se diferencia según el software y hardware empleado para su desarrollo y el tipo de contenido que pueden difundir.

La mayoría de empresas utilizan software y hardware propio, sin embargo, existen muchas herramientas de código libre (Open Source) para su desarrollo. Se pueden distinguir dos tipos de arquitectura: basada en tecnologías Web y basada en IPTV en la que el servidor distribuye el contenido mediante streaming.

Las soluciones basadas en IPTV precisan de una infraestructura de red más compleja, un alto costo para su implementación y no tienen sentido para contenidos casi estáticos. En este proyecto tiene sentido aplicar una de las soluciones de tipo Web donde el contenido corporativo se pueda gestionar a través de un portal mediante cualquier navegador Web.

De esta forma, para la implementación del sistema se ha realizado un estudio de las herramientas Open Source de Digital Signage basadas en web. A continuación se listan las principales herramientas encontradas:

- **Concerto:**

Concerto [2] es una solución Open-Source bajo la licencia GNU Public License v3. La arquitectura de esta solución es de tipo cliente – servidor tradicional. El cliente es básicamente una web que corre en un navegador. Esta herramienta permite subir el contenido a través de un portal web donde se gestionan las pantallas. Para comprobar las posibilidades que ofrece dicho portal, se puede encontrar en la página oficial de Concerto un link a un servidor de prueba [4]. Del mismo modo, Concerto facilita ejemplos para ver cómo es la visualización de contenidos en una pantalla [5].

En cuanto a los tipos de contenido que se pueden manejar, a través del portal web se pueden subir imágenes, ficheros PDF de una página, textos, entradas RSS y videos (en la versión 2). Además, para la instalación de un servidor propio basado en Concerto se tienen múltiples opciones descritas en su página web oficial [6]. En la parte cliente, Concerto cuenta con su propia aplicación Android para utilizarla como visualizador. El código de dicha app está disponible en su repositorio de git [3] y básicamente contiene un WebView que embebe la página web cliente.

- **Xibo:**

Xibo [8] es una solución Open Source bajo la licencia Affero GNU Public License v313 o cualquier versión posterior. La arquitectura de esta solución es también de tipo cliente – servidor tradicional. En su página oficial se indican los pasos a seguir para instalar el servidor de Xibo [9] descargando el código desde su repositorio Bazaar [10]. En cada ordenador asociado a una o más pantallas donde se quiera difundir el contenido es necesario instalar el cliente, el cual se encarga de la configuración de los parámetros necesarios para el correcto funcionamiento del sistema. La aplicación cliente puede correr bajo diferentes sistemas operativos. Xibo ofrece las siguientes posibilidades: del .NET Windows Client, del Python Ubuntu Client y del Android Client.

El cliente Ubuntu nació primero y por lo tanto es el que garantiza una instalación estable. El cliente Python tiene un mayor potencial para el futuro y con el tiempo se convertirá en el único cliente para Windows y Linux. El cliente de Android es un software comercial de su patrocinador y trae Xibo a bajo costo.

En cuanto a los tipos de contenido que soporta, la plataforma maneja: imágenes, texto, video en cualquier formato que soporte Windows Media Player, animaciones en formato SWF, sitios web embebidos, presentaciones en Microsoft Power Point y entradas RSS.

- **RiseVision:**

RiseVision [11] es un proyecto Open Source cuya licencia de código es MIT License . RiseVision proporciona una aplicación de usuario y una aplicación cliente de visualización que operan en Google App Engine. Dicha aplicación de usuario proporciona el medio de configurar y supervisar las pantallas administrando el contenido HTML que se entrega a la aplicación. Para comenzar a usarlo únicamente hay que iniciar sesión con una cuenta de Google y comenzar a utilizar su portal de administración de contenidos [12]. En las pantallas se debe tener instalado el cliente correspondiente que soporta diferentes sistemas operativos[14].

Risevision permite visualizar casi cualquier tipo de contenido: imágenes, textos, videos, páginas web embebidas, etc. Además ofrece la posibilidad de incluir distintas gadgets como puede ser Google Maps, Google Calendar, Twitter, etc o algunas Premium de pago como por ejemplo un Weather Widget.

Para crear aplicaciones personalizadas y crear gadgets propias, RiseVision cuenta con una API accesible desde su repositorio [13]. La API de RiseVision es RESTful lo que implica que puede ser usada en cualquier plataforma que use un entorno de desarrollo que soporte enviar y recibir datos sobre el protocolo HTTP. La API esta descrita en términos de: URIs, parámetros de entrada (pasados con la URI), métodos http (GET, PUT, DELETE, POST) llamados en la URI y respuestas (en XML para Core API y JSON para Viewer API).

Resumen de las plataformas:

En la siguiente tabla se recogen las principales características de las plataformas Open Source encontradas.

Plataforma	Concerto	Xibo	RiseVision
Formatos soportados	Imágenes, textos, RSS y videos (en versión 2)	Imágenes,PPT, textos,videos, HTML, swf y RSS	Imágenes, textos, videos, páginas web embebidas, gadgets de todo tipo, etc
Código del servidor	PHP+JavaScript+ HTML+CSS	PHP+JavaScript+ HTML+CSS	Servidor de Rise Vision en Google Cloud (código no disponible solo API)
Código del cliente (visualizador)	PHP+JavaScript+ HTML+CSS	.NET para Windows / Python para Ubuntu	No disponible
Plataformas soportadas para aplicación cliente	Web y Android	Windows, Ubuntu y Android (versión comercial)	Windows, Ubuntu, Chrome, RaspBerry Pi

Tabla 1 Resumen herramientas Open Source encontradas

2. Análisis técnico

2.1. Elección de dispositivos

En primer lugar hay que concretar el dispositivo a usar en el visualizador. Para poder emitir un canal Digital Signage en una pantalla de televisión será necesaria conectividad. Para ello, se tienen distintas opciones:

- Televisión Smart Tv

Smart TV es una denominación que las compañías de televisores han dado a sus modelos conectados y más avanzados. En ellos, además de la posibilidad de reproducir contenido desde diferentes fuentes, tienen conexión a Internet que puede ser vía WiFi y la posibilidad de instalar o usar aplicaciones diseñadas específicamente para un televisor. Podemos encontrar televisores de este tipo desde 350 € pero dependiendo de la gama y de las pulgadas de la pantalla se tiene un precio distinto.

- Conectar un dispositivo al televisor que dote de conectividad al mismo
    - Un ordenador

Actualmente existen muchos tipos de miniPC o barebones para sistemas operativos Linux o Windows como por ejemplo el XS35GS V3L de Shuttle. Según las prestaciones se tienen distintos precios. Los de baja gama rondan precios entre 100 y 300 euros y uno de los más famosos y más económicos es la Raspberry Pi. La Raspberry Pi es un ordenador del tamaño de una tarjeta de crédito que se conecta al televisor y a un teclado. Se trata de un pequeño ordenador que puede ser utilizado en proyectos de electrónica, y para muchas de las cosas que hace un PC de escritorio, como hojas de cálculo, procesadores de texto y juegos. También reproduce vídeo de alta definición. Sobre ella puede ejecutarse alguna de las múltiples distribuciones GNU/Linux que se han desarrollado para este mini ordenador. Para poder tener internet es necesario comprar un adaptador Wifi y su precio sumado al de un adaptador de este tipo es aproximadamente 70 euros.

    - Stick Android

Los llamados sticks internamente son dispositivos Android que generalmente tienen una salida HDMI para conectar a la HDTV y también un puerto para la alimentación así como puertos USB para entradas periféricas. Algunos tienen Bluetooth para añadir más conexiones. El precio de un stick oscila está entre los 50 y 150 euros.

    - Set-top box:

Los set top box son decodificadores Smart TV que pueden incorporar TDT. Ofrecen conexión a internet, posibilidad de grabación, TDT HD y otras opciones. Estos pequeños equipos incluyen un procesador y también suelen tener una versión de

Android. En cuanto al precio, por algo más de 50 euros ya se pueden encontrar equipos que según las prestaciones pueden llegar a costar más de 120 euros.

Comparativa:

Si comparamos las distintas opciones, se puede ver que cualquiera de las soluciones anteriores serviría para el objetivo de este proyecto. El diseño de aplicaciones para televisores Smart Tv suele ser específico para cada marca y sobre este dispositivo no se puede tener tanto control. Para un uso dedicado, como resulta en este caso, ya que el televisor solo mostrará el contenido del portal y no deberá mostrar otro contenido, no resulta por tanto adecuado. Los set top box comparados con los sticks y los mini ordenadores tienen un precio mayor y su uso está mas orientado a visualizar y grabar canales de televisión o reproducir contenidos vía streaming. En cuanto a los mini ordenadores, podemos ver como el precio es similar al de un stick Android, sin embargo, para programar aplicaciones se considera más sencillo utilizar el lenguaje de programación Android. Por este motivo se propone utilizar un stick Android.

Algunos de los sticks más conocidos son:

2.1.1.Tronsmart CX-919 [15]

- Android 4.2.2
- El procesador Rockchip RK3188 1,4 GHz de cuatro núcleos
- 2 GB de RAM y una capacidad de 8 GB, ampliable mediante elementos periféricos.
- con puertos USB y HDMI, así como ranura para tarjetas microSD
- integra tecnología con el estándar bluetooth 4.0
- dispone de una antena para facilitar la captación de la red WiFi, incluyendo asimismo un adaptador para obtener un puerto Ethernet.
- Precio aproximado: 40 €

2.1.2. Rikomagic MK802 IV[16]

- Sistema operativo Android 4.2
- Procesador: RK3188 ARM Cortex-A9 Quad Core
- Procesador grafico: Mali-400 MP Quad core
- HDMI: Output 720P, 576P, 480P, 1080P&2160P
- Memoria RAM: 2 GB
- Memoria interna: 8 GB
- Conectividad
  - Bluetooth 3.0
  - Wireless 802.11b/g/n, WAPI
- Alimentación USB
- Decodificar de video con soporte DLNA y Miracast
- Conexiones
  - 1 x USB (USB Estándar)
  - 1 x micro USB (USB Host)
  - 1 x micro USB (Alimentación)

    - 1 x Ranura microSD hasta 32GB
- Precio aproximado: 45 €

2.1.3.Cotton Candy(ICS 4.0)[17]

- CPU: Samsung Exynos 4210 Dual Core 1.2GHz Cortex A9
- GPU: Mali 400 MP4 (Quad-Core)
- ROM: microSD (hasta 64GB)
- RAM: 1GB
- Conectividad: HDMI, Wi-Fi, Bluetooth
- Precio aproximado: 120 €

Si comparamos los diferentes sticks se puede ver que tienen prestaciones similares en cuanto a CPU, ROM y RAM. Todos ellos se conectan por HDMI, ofrecen la posibilidad de conectar una tarjeta SD y disponen de conectividad Wifi. En el caso de Tronsmart CX-919, dispone de una antena externa por lo que en principio tiene una mejor recepción inalámbrica. En cuanto al precio, cabe decir que en función de la página donde se compre cada dispositivo hay variaciones, Cotton Candy tiene un precio mucho mayor. De esta forma, se propone utilizar el Tronsmart CX-919.

A su vez, las pantallas táctiles que se van a utilizar en este proyecto las define la empresa y son del siguiente modelo: Tablet Acer DA220HQLbmiacg (AIO Dual-Core OMAP 4428 1GB 6GB 21,5").

2.2. Elección de plataforma

Analizando las plataformas Open Source encontradas, se observa que no existen diferencias significativas en cuanto al tipo de contenido y la planificación del mismo. En cambio, si se diferencian en el tipo de lenguaje que emplean, en la apariencia de la interfaz y en qué sistema puede correr la aplicación cliente.

El punto de interés principal a la hora de elegir la plataforma es que el sistema sea personalizable y ampliable. Se pretende que constituya una base sobre la cual comenzar a desarrollar el proyecto y adaptarlo a las necesidades del cliente. En el planteamiento del proyecto se pretende realizar una versión para Tablet en la que se pueda acceder a todo el contenido subido al portal. A la hora de elegir la plataforma a usar hay por tanto que tener en cuenta la facilidad para incorporar dicha versión. También se debe poder incluir nuevos tipos de contenido al portal o nuevas pestañas.

En este sentido, RiseVision resulta difícil de personalizar debido a que su servidor opera en Google App Engine y aunque se tenga la API disponible para incluir gadgets personalizadas no se podría adaptar el sistema a lo que se requiere en este proyecto.

En el caso de Xibo, se tiene que el cliente está programado en .NET o Phyton. Concerto, sin embargo, está basado en lenguajes de programación web de los que se tiene un mayor conocimiento. Por esta razón y dado que el cliente debe correr en un stick Android se propone usar la plataforma de Concerto.

3. Diseño e implementación

3.1. Instalación de Concerto Server

En la web oficial de Concerto se proporcionan distintos métodos de instalación [6] y se han probado los tres siguientes:

- Paquete Debian: Con este método de instalación se obtuvieron una serie de errores y no fue posible llegar a finalizar la instalación y tener Concerto funcionando.
- Imagen de VirtualBox: Al descargar la imagen .oba e importarla a VirtualBox se tiene la versión 2 de Concerto accesible. La versión 2 de Concerto, todavía en versión Beta, fue testeada y en ella se detectaron una gran cantidad de errores (errores al subir contenidos, fallos en la inclusión de videos, etc).
- Instalación manual de versión 1.9.3: Este es el método con el que se ha conseguido una versión estable de Concerto instalada en un servidor propio [7].

3.2. Estructura del portal web de Concerto

3.2.1. Funcionamiento básico del portal

El funcionamiento básico de Concerto consiste en subir contenido a través del portal asignando ese contenido a un "feed" o categoría y estableciendo en qué fechas va a ser visible dicho contenido. Las pantallas se suscriben a ciertas categorías mostrando en bucle el contenido (el que corresponda según la fecha) de dichas categorías. La Figura 1 refleja cómo es este funcionamiento.

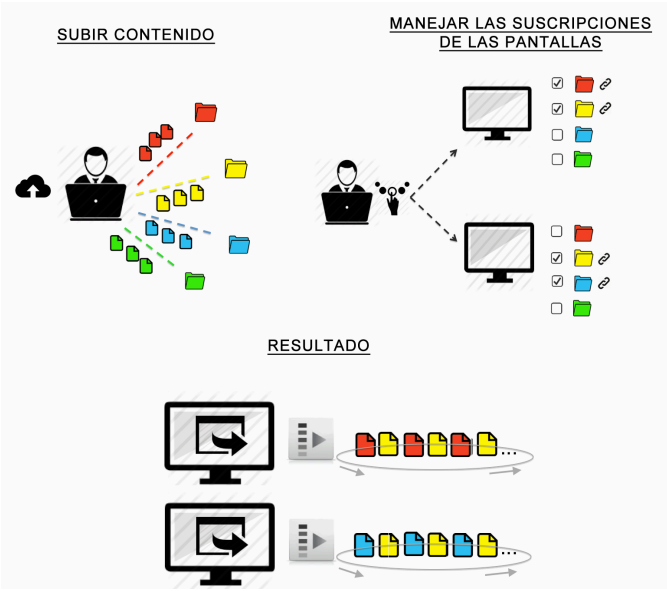

Figura 1 Esquema de funcionamiento de Concerto

Además, Concerto establece dos tipos de usuario: el administrador y el usuario básico. El administrador es el único usuario capaz de crear nuevas pantallas, categorías, usuarios y grupos de usuarios. El resto de usuarios pueden subir contenido a la plataforma y dependiendo a que grupo de usuarios pertenezcan podrán controlar ciertas pantallas y ciertas categorías de contenido. Controlar una pantalla implica decidir a qué categorías está suscrita y controlar una categoría implica moderar el contenido que alberga.

Existen tres tipos de categorías: estándar, restringida o privada.

- En la estándar todos los usuarios pueden ver el contenido que alberga y pueden añadir contenido a ella siempre que dicho contenido sea aprobado por un usuario que controle dicha categoría.

- En la restringida sólo los usuarios que pertenecen a su grupo controlador pueden añadir contenido.

- Por último, las categorías privadas sólo son visibles por los usuarios que pertenecen a su grupo controlador.

En el portal de Concerto se tienen diferentes pestañas que se observan en la Figura 2:

- **Dashboard:** pantalla inicial del portal donde se muestran todas las pantallas existentes en el sistema.
- **Add content:** pestaña para añadir contenido al portal. Dentro de la misma se tienen a su vez dos pestañas: Image o TickerText, para añadir ficheros de imágenes y añadir textos cortos respectivamente.
- **Browse Content:** pestaña para explorar el contenido de las distintas categorías o feeds.
- **Screens:** pestaña para explorar y administrar las pantallas del sistema.
- **Users:** pestaña para explorar y administrar usuarios.
- **User Groups:** pestaña para explorar y administrar grupos de usuarios.
- **Admin:** pestaña para el administrador del sistema.
- **InfoPages:** pestaña de ayuda con páginas de información de uso del portal.

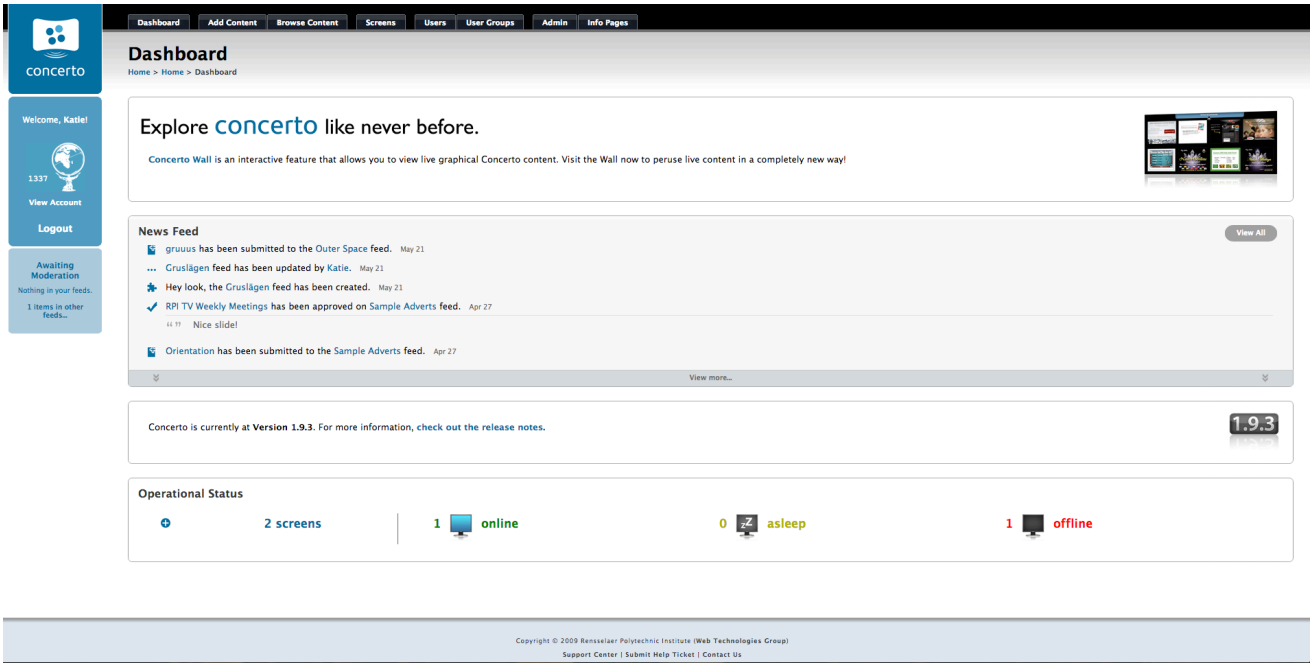

Figura 2-Captura de pantalla del portal de Concerto

3.2.2.Estructura del código

El código del servidor se estructura definiendo los siguientes directorios:

- Common:

En esta carpeta se alojan las clases php que definen los objetos modelo principales como por ejemplo la clase Screen.php. En todas las clases está definido un constructor, en la mayoría de ocasiones, con un id, un método create_construct en el cual se ejecuta la query necesaria para crear un nuevo objeto en la base de datos y un método set_properties para editar dicha entrada de la base de datos.

- Content

Es la carpeta donde se alojan las imágenes subidas por los usuarios.

- Screen

Carpeta donde se encuentra la página que solicita el visualizador.

- Admin

En esta carpeta se encuentra casi todo el código del portal web. En su interior tenemos el fichero index.php al cual se accede cuando se entra en el portal inicialmente. En index.php se define la clase base Controller.php de la que extienden todos los controladores de cada pestaña. Dentro de Admin se tienen varias carpetas:

  - app: donde se encuentra por carpetas el código correspondiente a cada pestaña (browse, content, feeds, frontpage, groups, moderate,

page_categories, pages, screens, templates, users, wall) cada una con su propio controller.php y sus propias secciones (edit, new,etc).

- css: donde se encuentran los archivos CSS para aplicar estilos al portal.

- images: donde se albergan los iconos utilizados en el portal.

- includes: donde se encuentra el código común a todas las pestañas formado por el menú lateral izquierdo (leftmenu.php), la barra de pestañas (menu_tabs.php) y el fichero que comprueba si se ha iniciado sesión en el sistema (login.php).

- js: donde se encuentran las librerías JavaScript usadas en el portal.

3.2.3. Base de datos

En la base de datos de Concerto se tienen las siguientes tablas:

- **content:** Tabla donde se almacenan los datos referentes al contenido subido por los usuarios del portal. En la estructura de la tabla se definen los campos id, name (nombre asignado al contenido), user_id (el del usuario que lo sube), content (nombre del fichero en caso de imagen o texto a mostrar en caso de Ticker Text), mime_type (tipo de contenido o image/* o text/*), type_id (tipo de contenido dentro de los que se definen en la tabla types), start_time (tiempo de inicio para la visualización del contenido), end_time (tiempo de finalización para la visualización), duration (duración del contenido en pantalla) y submitted (fecha de subida).
- **dynamic**: Tabla para añadir contenido dinámico. Está pensada para añadir RSS, para añadir uno se inserta en el campo type_field un 1, en path la URL del RSS, en el campo rules un array php serializado con las reglas para formatear dicha entrada en el campo y en update_interval el número mínimo de segundos entre actualizaciones.
- **feed:** Tabla que contiene las categorías de contenido. En la estructura de la tabla se definen los campos id, name (nombre de la categoría), group_id (corresponde al grupo de usuarios propietario de dicha categoría), type (tipo de categoría: pública, restringida o privada), dynamic_id (usado para RSS) y description (texto con la descripción de la categoría).
- **feed_content**: Tabla que relaciona los contenidos con sus correspondientes categorías. En la estructura de la tabla se definen los campos feed_id (id de la categoría), content_id (id del contenido), moderation_flag (null si está sin moderar, 0 si está rechazado y 1 si está aprobado), moderator_id (el moderador que ha moderado dicho contenido, duration (tiempo que se muestra en pantalla dicho contenido), display_count (número de veces que ha sido mostrado por categoría) y yesterday_count (número de veces que se ha mostrado en dicha categoría ayer).
- **field:** Tabla que contiene los campos o fields de una plantilla o template.

En la estructura de la tabla se definen los campos id, name (nombre del campo Graphics o Ticker), template_id (id del template al que pertenece dicho campo), type_id, style (contiene el css para modificar la vista del campo), left (coordenada horizontal izquierda en la plantilla en tanto por 1), top (coordenada vertical en la plantilla en tanto por 1), width (ancho del campo en tanto por 1) y height (altura del campo en tanto por 1).

- **group**: Tabla donde se almacenan los grupos que contiene los campos id y name (nombre del grupo).
- **newsfeed:** enlaza las notificaciones con los usuarios a los que va dirigido.
- **notifications**: Tabla donde se almacenan las nuevas notificaciones del sistema, cuando se edita o se modera o se crea un contenido, etc.
- **page**: Tabla donde se almacena el código HTML de las páginas de ayuda.
- **page_category**: Tabla donde se guarda el nombre del fichero php que hace de plantilla para la pestaña de ayuda.
- **position:** Tabla que relaciona qué campo de qué pantalla está suscrito a qué categoría y con qué peso, es decir, con qué frecuencia. En ella se tienen los campos id, screen_id, feed_id, field_id, weight, display_count y yesterday_count.
- **screen:** Tabla que contiene las pantallas del sistema. En la estructura de la tabla se definen los campos id, name (nombre asignado a la pantalla), group_id (grupo controlador de la pantalla), location (localización de la misma), mac_address (dirección MAC), template_id (id de la plantilla para dicha pantalla), last_updated (fecha de su última actualización), controls_display (flag indicador de si se controla on/off de pantalla), time_on (tiempo en el que se debería encender la pantalla), time_off (tiempo en el que se debería apagar) y display_count (número de contenidos que se han visualizado en la pantalla).
- **template**: Tabla que contiene las plantillas del sistema. En la estructura de la tabla están definidos id, name (nombre de la plantilla), filename (nombre del fichero imagen de la plantilla), height, width, creator, modified y hidden.
- **type**: Tabla donde se definen los tipos de contenido del sistema. Se define un id y un nombre (name). En esta tabla están introducidos dos tipos: Graphics y TickerText.
- **user**: Tabla que contiene los usuarios del sistema. En ella se definen los campos: id, username, password, name, email, admin_privileges y allow email.
- **user_group:** Tabla que relaciona los usuarios con los grupos a los que pertenecen.

3.3. Personalización de Concerto

3.3.1. Modificaciones y nuevas funcionalidades

3.3.1.1. Nuevas funcionalidades

Según los requerimientos del cliente es necesario hacer ciertos cambios en el portal de Concerto original. Concerto sólo tiene en cuenta un tipo de visualizador

que está pensado para reproducirse en una red de pantallas de televisión de forma que reproducen una playlist con el contenido y sin ningún tipo de interacción con el usuario que visualiza el contenido. A este tipo de visualizador se le va a designar el nombre "visualizador tipo Wall". Para este proyecto se necesita otro tipo de visualizador, al que se designará el nombre "visualizador tipo Tablet", en el que el usuario sea capaz de interactuar y consultar el contenido que desee a través de un menú de navegación. Así, el esquema de funcionamiento sería similar y quedaría como se refleja en la Figura 3.

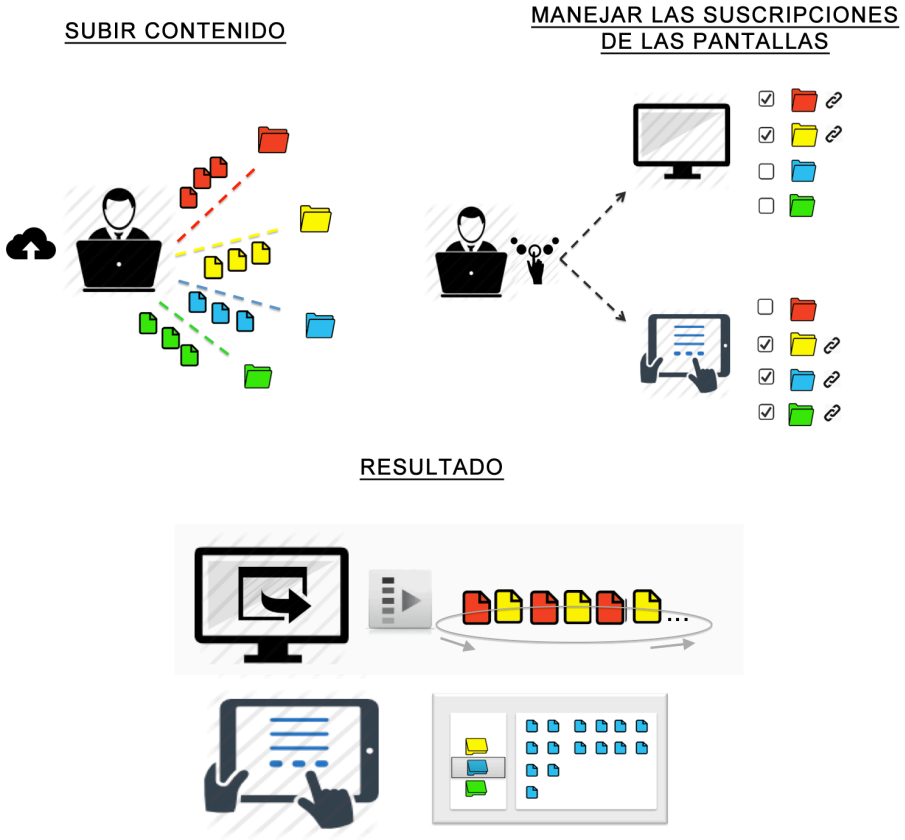

Figura 3 Esquema de funcionamiento del sistema incluyendo pantallas tipo "Tablet"

En la versión instalada de Concerto, al portal de administración se pueden subir archivos de tipo imagen en los formatos JPG, PNG, GIF y PDF de una página (que se convierten a imagen y se tratan como tal). También Concerto ofrece la posibilidad de añadir textos cortos a los que llama "TickerText". De esta forma, como nueva funcionalidad, se deben incorporar dos nuevos tipos de contenido: gráficos y ficheros PDF de varias páginas.

Los gráficos se pretende que se generen a partir de ficheros CSV que genera el ERP de la empresa. El ERP transferirá al servidor dichos ficheros vía FTP. Estos gráficos, podrán visualizarse tanto en pantallas tipo Wall como en las tipo Tablet de forma similar a otros tipos de contenido. Además, el usuario administrador podrá decidir qué tipo de gráfico aplicar a cada CSV subido por el ERP, pudiendo decidir entre un gráfico de tipo XY o diagrama de barras. A su vez, en dicho CSV se podrá marcar la escala que se desea aplicar al eje vertical de coordenadas así como los títulos de los ejes.

Los ficheros PDF de varias páginas sólo tendrá sentido que se muestren en los visualizadores tipo Tablet. Además, en el nuevo tipo de pantalla Tablet se podrán ver todos los tipos de contenido excepto el TickerText, ya que son textos generalmente cortos y en la Tablet no es interesante visualizarlos.

En definitiva, en nuestro escenario de red aparecerán los siguientes elementos: un servidor de contenido y FTP, el ERP de la empresa, algunas pantallas físicas, tanto Wall como Tablet, así como los hosts que se conecten al panel para administrar el sistema y subir contenidos. El diagrama de la arquitectura del sistema resultante quedaría como se muestra en la Figura 4.

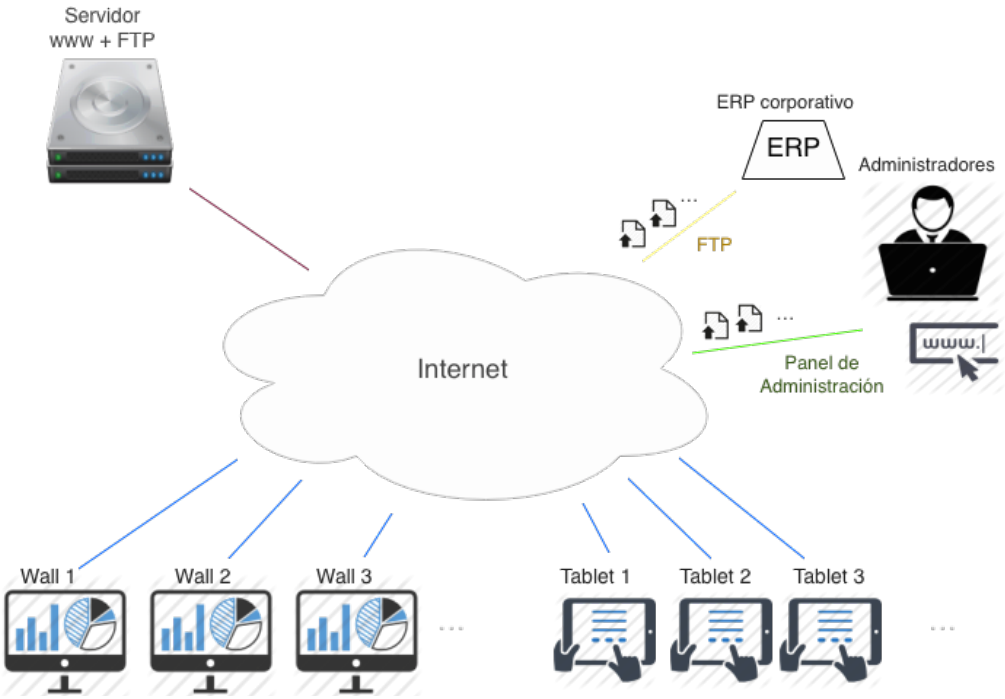

Figura 4 Diagrama de la arquitectura del sistema

Además, para el cliente en concreto para el que se realiza este proyecto, el idioma del portal de administración deberá ser Español. Sin embargo, se considera interesante de cara a implantar este mismo sistema en un futuro a otras empresas tenerlo en varios idiomas.

3.3.1.2.Modificaciones en las distintas pestañas de Concerto

Con el fin de adaptar Concerto a las necesidades del cliente, se realizan una serie de cambios en cada una de las pestañas de Concerto. Estos cambios se describen a continuación.

En la pestaña de **Dashboard** se elimina la información de notificaciones dejando únicamente el panel resumen donde aparecen todas las pantallas del sistema: conectadas y no conectadas. En la Figura 5 y Figura 6 se puede ver las diferencias entre el nuevo Dashboard y el que había en el portal de Concerto original.

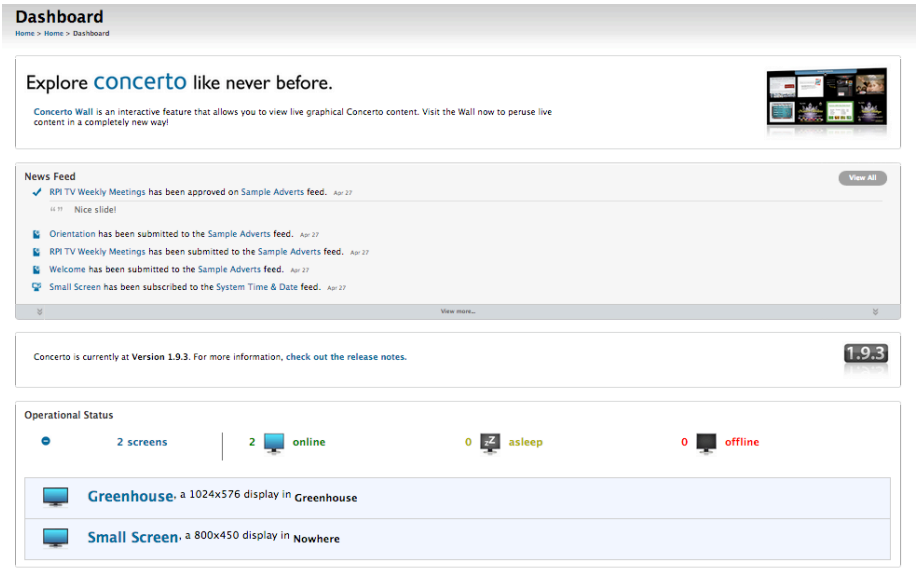

Figura 5 Captura de pantalla del Dashboard de Concerto

Figura 6- Captura de pantalla del nuevo Dashboard

Como se puede ver en la Figura 8 en la pestaña de **AddContent** se introducen dos nuevas pestañas: añadir documentos (para añadir ficheros PDF de varias páginas) y añadir gráficos (para añadir gráficas a partir de ficheros CSV subidos por el ERP de la empresa).

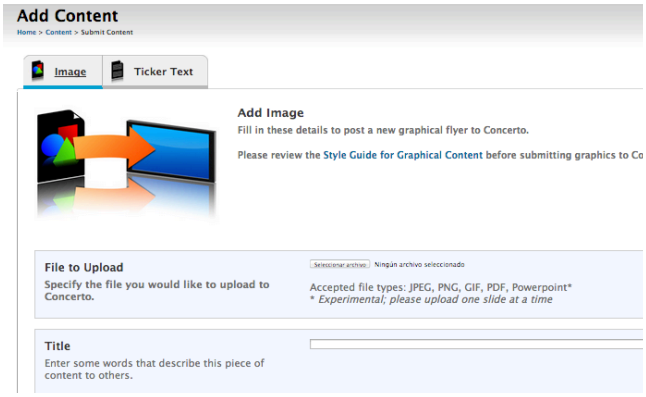

Figura 7 Captura de pantalla de la pestaña AddContent en Concerto

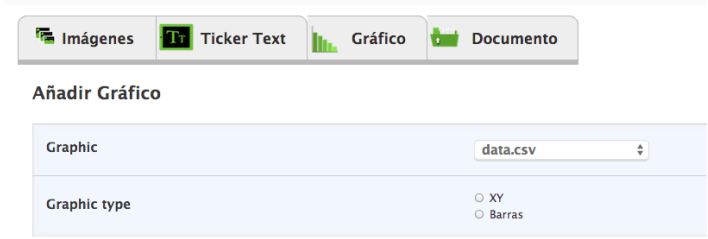

Figura 8- Captura de pantalla del nuevo AddContent

Además, se elimina de los formularios de creación de contenido el campo "Título", ya que se considera innecesario. Los títulos serán sustituidos por los nombres del fichero o por el comienzo del texto en el caso de los "TickerText" para que el usuario tenga que rellenar el menor número de campos posibles. Del mismo modo, con el fin de facilitar la experiencia del usuario se modifica el formulario para que se puedan subir varios ficheros al mismo tiempo con la misma configuración (fechas de visualización, duración y categorías asignadas). Cabe decir, que al subir varias imágenes a la vez, el usuario debe esperar cierto tiempo hasta que la subida se complete. Para retener al usuario en la página y evitar que piense que el sitio web no funciona se coloca una imagen de fondo animada para que el usuario sepa que se está subiendo su contenido. Para ello, se utiliza la librería spin.js [18] que facilita la inclusión de una imagen de carga. Básicamente, cuando el usuario envía el formulario, se captura dicho evento y se lanza una función JavaScript que carga en un div la animación de carga.

En cuanto a las nuevas pestañas incorporadas, para incluir la pestaña de añadir documentos se sigue el esquema de programación de Concerto introduciendo un nuevo tipo de contenido con type_id=5 en la base de datos. Para ello se añade en el directorio admin/app/content un nuevo fichero new_document.php similar al new_ticker.php y se añade a la acción new en el controlador de esta pestaña.

De la misma forma, para incluir la pestaña de añadir gráficos se añade en el directorio admin/app/content un nuevo fichero new_graphic.php. En él se incluye los input del formulario que se deben completar a la hora de subir un gráfico. Cabe decir, que para recibir los ficheros CSV del ERP de la empresa cliente es necesario habilitar un servidor FTP. Dicho servidor se implementa con proftpd en el que se da acceso a la carpeta /content/csv del servidor web de manera que la empresa pueda alojar todos sus ficheros CSV en ese directorio.
En el formulario de subida de un gráfico aparte de introducir la duración, fecha de inicio y finalización y categoría a asignar, se elige un fichero CSV de la lista de ficheros subidos por FTP y se selecciona el tipo de gráfico que se desea, XY o de barras. Para guardar el tipo de gráfico, es necesario introducir otro campo en la tabla content de la base de datos llamado "graphic_type" que valdrá 0 o 1 en función del tipo de gráfico seleccionado. En el campo content de la tabla content se guardará el nombre del fichero CSV del cual debe provenir el gráfico en cuestión.

Además se modifica la clase content.php para que al subir una imagen se guarde una previsualización de la misma que será utilizada en la versión Tablet.

En la pestaña de **Browse**, donde se listan todas las categorías, se incluye un botón para añadir subcategorías y se modifica el listado para poder visualizar las mismas. En la Figura 9 y Figura 10 se ven los cambios realizados.

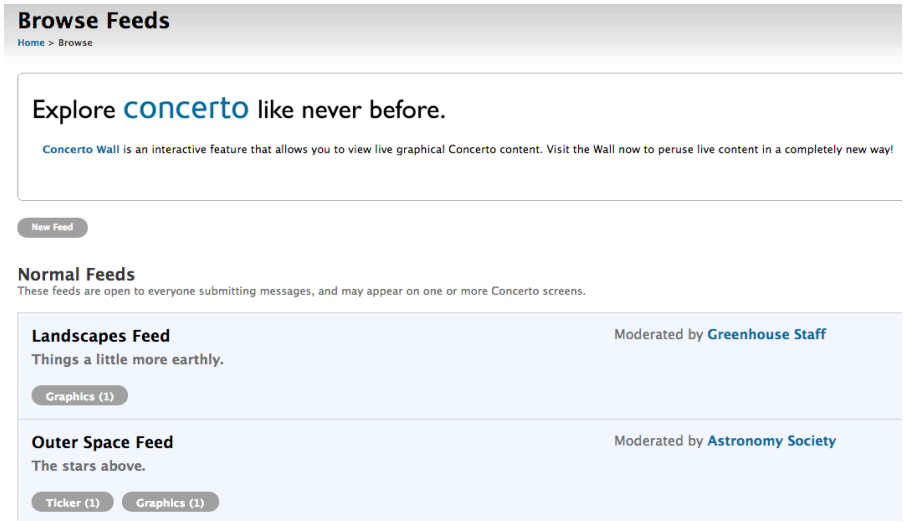

Figura 9 Captura de pantalla de Browse en Concerto

Figura 10-Captura de pantalla del nuevo Browse

Para añadir una subcategoría se añade en /admin/app/feeds los ficheros newsub.php, form_subfeed.php y editsub.php añadiendo las correspondientes acciones en controller.php. En newsub.php se carga un formulario donde se debe completar el nombre de la subcategoría y se selecciona cuál de las categorías existentes en el sistema va a ser la padre. De nuevo, es necesario añadir cambios en la base de datos introduciendo el campo father_id en la tabla feeds para poder guardar el id de la categoría padre. De esta forma, si se crea una categoría en vez de una subcategoría el campo father_id valdría NULL.

En la pestaña Browse, si se accede a una categoría se detalla una descripción de la misma: el estado de moderación, el contenido que alberga –clasificado en gráficos

y ticker (TickerText de dicha categoría)– y las pantallas activas, es decir donde se visualiza dicha categoría. Si se hace click en el enlace de Contenidos (o bien Ticker o bien gráficos) se listan todos los contenidos de dicha categoría. Como una categoría puede contener una gran cantidad de elementos, a esta vista se le añade paginado. El paginado se añade a través de la clase Paginator [23] que es un script de paginación desarrollado en PHP para dividir resultados de consultas extensas a una base de datos MySQL, en grupos de "n" registros por página. Paginator como se puede ver en la Figura 11 y la Figura 12 genera, además, una barra de navegación que contiene los enlaces a las diferentes páginas (<<anterior 1 2 3 4 siguiente>>).

En esta vista, se añade la posibilidad de editar ciertos parámetros de cada contenido, en concreto, su fecha de visualización, su duración y sus nombres. Además, se facilita la eliminación de contenidos incluyendo en el listado de contenidos checkboxes para poder eliminar varios ítems simultáneamente sin tener que entrar a la página de detalle de cada ítem. También se añade la opción de moderar contenidos desde el listado. Todos estos cambios se ven reflejados en la Figura 11 y en la Figura 12.

Figura 11 Página para explorar una categoría en Concerto

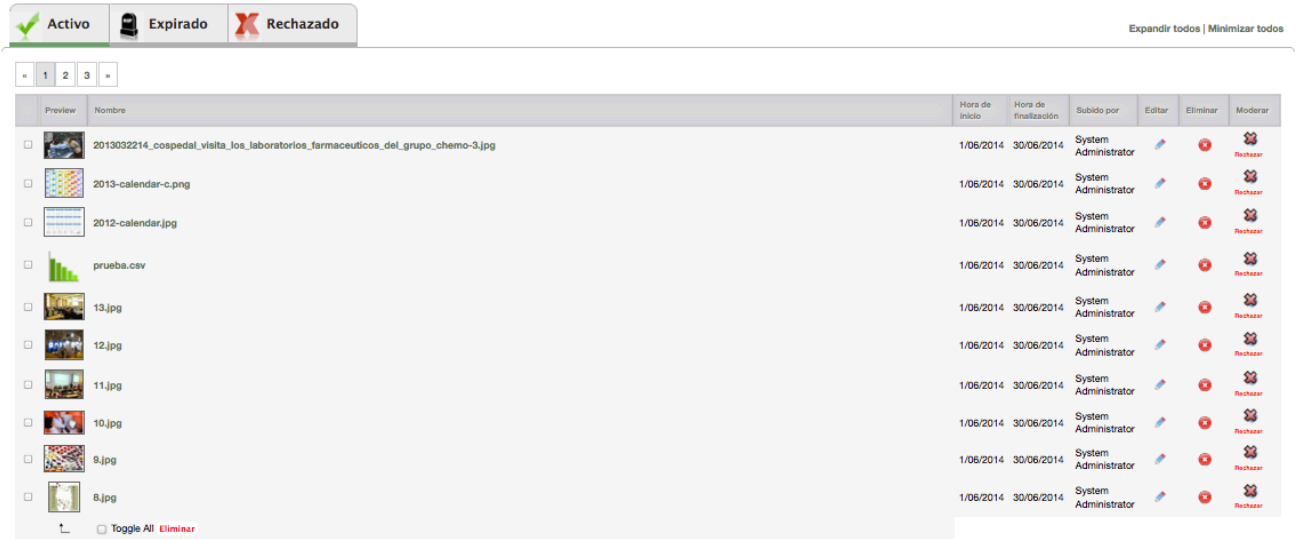

Figura 12 Cambios en la página de "Explorar categoría"

En la pestaña **Screens**, donde se listan las pantallas, se incluye un botón para añadir Tablets como se puede ver en la Figura 13 y la Figura 14 .El nuevo tipo de pantallas Tablet tendrá asignado en el campo type de la base de datos un type=1. A su vez, se modifica los campos del formulario de creación y edición de pantallas para que incorporen contraseña y no permitan nombres de pantalla con espacios o caracteres no autorizados. Además, se eliminan los campos ancho y alto, así como latitud y longitud de pantalla que se consideran innecesarios. También se elimina la opción de control on/off.

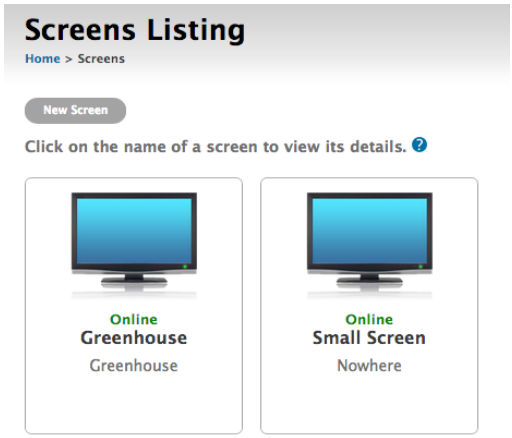

Figura 13 Captura de pantalla de pestaña Screens en Concerto

Figura 14-Captura de pantalla del nuevo Screens

Así, se sustituye el formulario para crear una nueva pantalla de Concerto (Figura 15) por dos formularios: uno para las pantallas tipo Wall (Figura 16) y otro para las pantallas tipo Tablet (Figura 17).

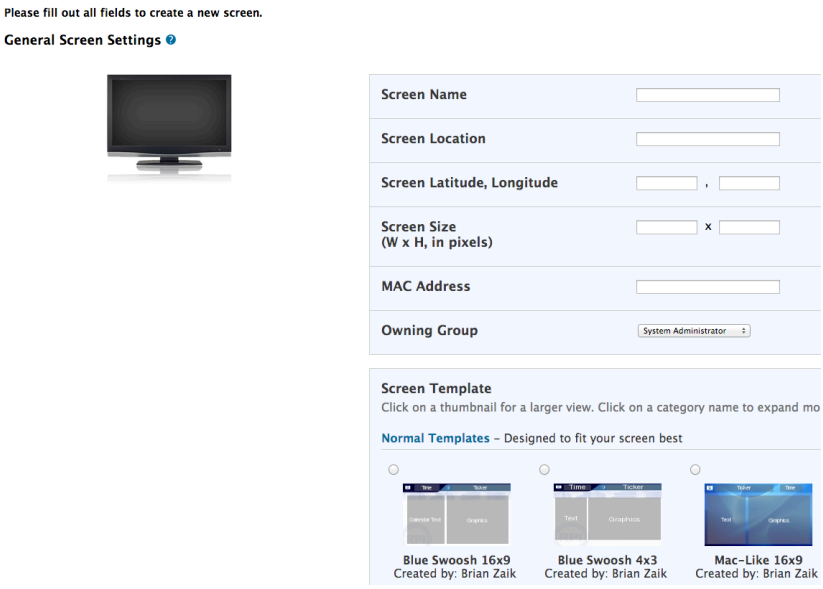

Figura 15 Formulario de creación de pantalla en Concerto

Figura 16- Formulario de creación de pantalla

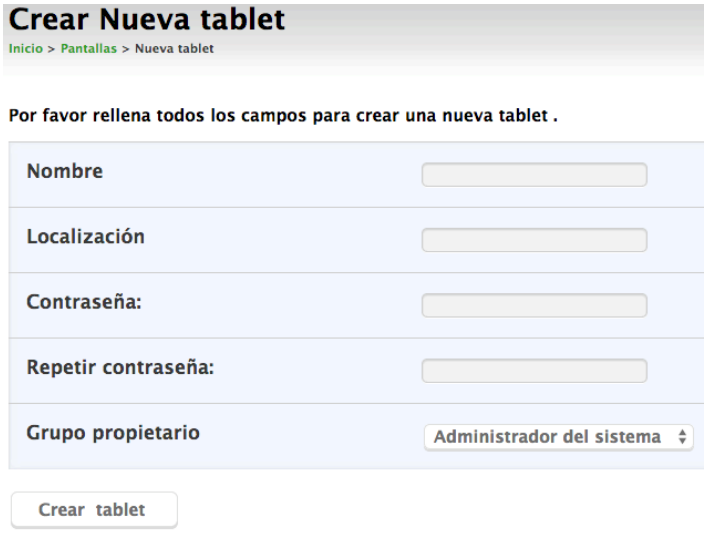

Figura 17-Formulario de creación de Tablet

En ambos casos el campo MAC de la base de datos se crea aleatoriamente. Este campo MAC se utilizará como token de sesión en las aplicaciones cliente.

Además, en el sistema se crea una categoría de contenido fija, la cual no se permite editar ni eliminar, llamada "Novedades" con id=0. Esta categoría se utilizará para que los usuarios introduzcan los contenidos que quieren que los usuarios de las pantallas tipo Tablet vean cuando nadie este interactuando con la misma. Al crear una Tablet nueva, se crea una subcategoría de Novedades para la misma con el nombre que se ha asignado a dicha Tablet y se suscribe de forma automática dicha Tablet a la categoría Novedades y a su subcategoría generada. Además, esta suscripción automática a Novedades no se permite modificar desde el portal. En las pantallas tipo Tablet se fija una plantilla con un único campo y fondo transparente. De esta forma, se pretende que las tablets tras un periodo de

inactividad del usuario, visualicen del mismo modo que en las de tipo Wall el contenido tanto de los elementos que haya en Novedades como en los de su propia subcategoría de Novedades. Cuando una Tablet se borra del sistema se borra su subcategoría de Novedades.

En la pestaña de **Users** se define otro tipo de usuarios de forma que se tiene:
un usuario administrador del sistema que puede crear y editar pantallas, usuarios, grupos de usuarios, categorías y contenidos. Además, puede moderar cualquier contenido y manejar las suscripciones de cualquier pantalla. Un usuario administrador de contenidos puede realizar lo mismo que el administrador del sistema exceptuando la creación de nuevas pantallas. Y por último, un usuario básico puede crear contenido y solo puede editar y eliminar contenidos que pertenezcan a una categoría del grupo de usuarios al que pertenece. Además, este usuario básico puede manejar las suscripciones de las pantallas que su grupo de usuarios controle.

En la pestaña de **Groups** no se realizan modificaciones. Por último las pestañas de **Admin** y **InfoPages** son eliminadas debido a que su funcionalidad no se considera interesante. La pestaña de InfoPages no se considera necesaria debido a que al cliente se le entrega un manual de usuario para manejar el portal de administración. Y en la pestaña Admin se podían añadir nuevas templates, en este caso, como las templates de pantallas para el cliente van a ser fijas no interesa y también permitía mandar emails que tampoco se considera necesario.

Idiomas

Se desea hacer un sitio web multilingüe, es decir, disponible en varios idiomas. Para ello, una solución es utilizar las funciones gettext incluidas en PHP. Gettext [20] es un conjunto de utilidades que forman parte del proyecto GNU y que facilita la traducción de programas.
Básicamente, la utilización de gettext en PHP es el siguiente:

- el programador indica en el código fuente de las páginas web cuáles son las cadenas de texto a traducir. Las cadenas que se quieran traducir, deben enviarse a la función gettext($cadena), aunque normalmente se utiliza el alias _($cadena).

- con alguna utilidad como Poedit [21], el programador crea archivos .po con la traducción de todas las cadenas (un archivo por idioma). A continuación se muestra un ejemplo del formato de un archivo .po para las traducciones a Español:

```
msgid "Screens"
msgstr "Pantallas"

msgid "Users"
msgstr "Usuarios"

msgid "User groups"
msgstr "Grupos de usuarios"
```

- al servirse las páginas webs, automáticamente se sustituyen las cadenas originales por las traducciones (si alguna cadena no está traducida, se muestra las cadenas en el idioma original)

La extensión Gettext está normalmente incluida en PHP (por ejemplo, lo está en XAMPP), aunque si no lo está, se puede utilizar la biblioteca php-gettext instalándolo con el comando:

```
sudo apt-get install gettext
```

Además es necesario tener los paquetes de los idiomas que se vayan a usar instalados. En este caso se debe tener el español que se instala con el comando:

```
apt-get install language-pack-es
```

Además para que php detecte el comando gettext es necesario modificar el fichero php.ini de configuración y añadir la extensión mediante la directiva:

```
extension = gettext.so
```

Suponiendo que nuestros ficheros de idioma se encuentran en el directorio /var/www/admin/locale/<código de idioma>/LC_MESSAGES/messages.po para indicar en qué idioma se debe servir la página se configura de la siguiente forma:

```
$language = "es_ES.utf8";
putenv('LANG='.$language);
putenv('LANGUAGE='.$language);
putenv('LC_ALL='.$language);
putenv('LC_MESSAGES='.$language);
setlocale(LC_ALL, $language);
bindtextdomain("messages", "/var/www/admin/locale/");
textdomain("messages");
```

Además de las cadenas de texto, hay que modificar los calendarios de los formularios que utilizan la librería DatePicker de JQuery [19] y añadirles ciertos parámetros y traducciones según el idioma. Por ejemplo en el caso de español, la semana no la muestra comenzando por Domingo sino por Lunes y se tienen que aplicar las traducciones pertinentes de días de la semana y meses. Para aplicar las modificaciones según el lenguaje se aplica el fichero .js correspondiente al idioma de la librería disponibles en su página [22].

3.4. Aplicaciones cliente

Una vez detallada la parte del servidor queda por describir la parte cliente, los visualizadores. La información subida a través del portal web se presenta de dos formas distintas, en el modo Wall y en el modo Tablet. Para cada modo se desarrolla una aplicación Android distinta. En ambas aplicaciones el usuario debe

identificar la pantalla a la que se desea conectar introduciendo los credenciales de dicha pantalla. En el modo Wall, se pretende visualizar el contenido de las categorías a las que esté suscrita la pantalla de forma cíclica y sin ningún tipo de interacción con el usuario. En el modo Tablet se debe poder navegar por todos los contenidos pertenecientes a las categorías a las que esté suscrito y tras un periodo de inactividad del usuario se debe visualizar, como en el modo Wall, los contenidos en bucle de las categorías Novedades para la Tablet.

3.4.1.Elementos comunes en ambas aplicaciones

La aplicación para Wall y la aplicación para Tablet tienen varios elementos en común. Por un lado las actividades y por otro un widget meteorológico.

Actividades:

Cada aplicación consta de cuatro actividades, en todas ellas se establece el tema de estilos de Android Theme.Light.NoTitleBar para que se oculte el ActionBar (barra superior de Android). Estas cuatro actividades se describen a continuación:

- **SplashScreenActivity:**

Actividad inicial que carga una Splash Screen. El concepto de SplashScreen se refiere a la primera pantalla que un usuario ve antes de ingresar al contenido principal de la aplicación. En este caso, dura 2 segundos en pantalla y muestra una imagen de bienvenida. En el AndroidManifest de cada aplicación se declara SplashScreenActivity como la actividad que se lanza al encender el dispositivo. Para ello se añade en la declaración de SplashScreenActivity del manifiesto lo siguiente:

```
<intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
      <category android:name="android.intent.category.HOME" />
      <category android:name="android.intent.category.DEFAULT" />
</intent-filter>
```

En la siguiente figura se aprecia la vista de esta actividad.

Figura 18 SplashScreenActivity

- **WifiActivity:**

Actividad que comprueba la conectividad del dispositivo a una red Wifi. Si el terminal no está conectado, muestra la pantalla para configurar una red y conectarse a la misma. Si el dispositivo ya tiene conectividad salta a la actividad LoginActivity. En la siguiente figura se aprecia la vista de esta actividad.

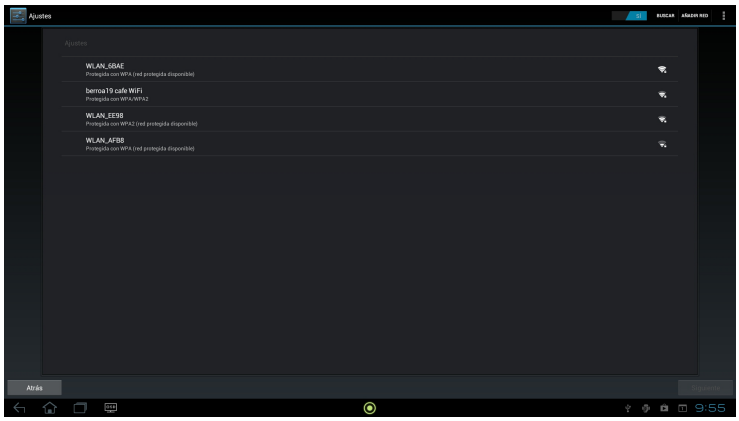

Figura 19 WifiActivity

- **LoginActivity:**

Actividad que comprueba los credenciales de la pantalla o Tablet según corresponda. Muestra un formulario en el que se introducen los credenciales necesarios para poder visualizar los contenidos asignados a esa Wall o Tablet. En ambas aplicaciones, para comprobar si los credenciales son correctos, LoginActivity realiza una petición al servidor enviando como variables POST el usuario y contraseña a la dirección:

```
http://serverdomain/admin/app/access_screen/index.php?tablet=<true/false>
```

El servidor devuelve un JSON indicando si los credenciales son válidos o no y si son válidos devuelve el campo MAC de dicha pantalla. A continuación se muestra un ejemplo de la respuesta JSON recibida si los credenciales son válidos:

```
{"status":"ok","user":"entrada","token":"248653"}
```

Y en caso de que sean inválidos:

```
{"status":"error","errors":{},"global":["Usuario y\/o contrase\u00f1a incorrectos"]}
```

La MAC será utilizada como token de sesión en ambas aplicaciones. Dicha MAC se guarda en el dispositivo en SharedPreferences. En la siguiente figura se aprecia la vista de esta actividad.

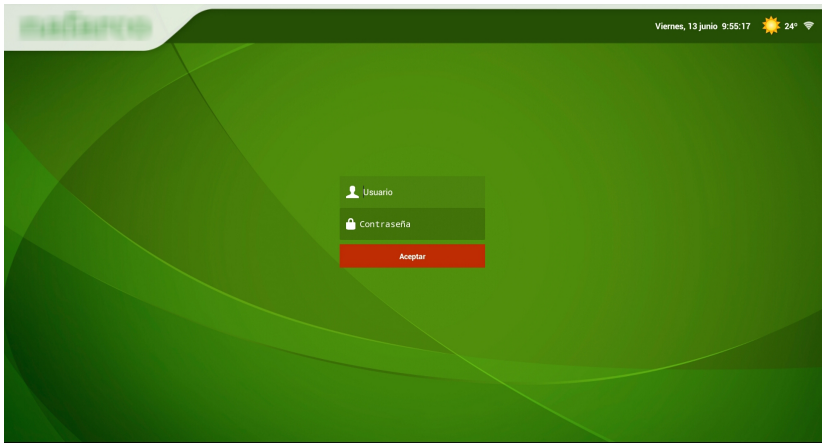

Figura 20 LoginActivity

- **TVActivity**

Actividad principal donde se muestra o bien la lista de reproducción de contenidos correspondiente a la pantalla (versión Wall) o el menú donde es accesible todo el contenido disponible para esa pantalla (versión Tablet).

Widget meteorológico:

Tanto en LoginActivity como en TVActivity se incluye en la parte superior derecha un widget con el tiempo meteorológico, la fecha y hora actual. Para realizar dicho widget, se crea una vista customizada WeatherWidget.java que contiene un layout en el que se tienen los siguientes elementos:

- DigitalClock: un widget de Android que muestra un reloj digital.
- TextView : un texto que muestra la fecha actual.
- ImageView+TextView: una imagen del tiempo meteorológico actual y un texto con la temperatura.

En la siguiente figura se muestra un ejemplo de la vista generada por WeatherWidget.

Figura 21 WeatherWidget

Para obtener las condiciones meteorológicas se utiliza la API de OpenWeatherMap [24]. La vista realiza una petición indicando la latitud y longitud en la que se encuentra el terminal de la siguiente forma:

```
http://api.openweathermap.org/data/2.5/weather?lat=<latitude>&lon=<
longitude >&units=metric
```

Después se parsea el JSON que recibe extrayendo el nombre del icono que debe mostrar y la temperatura actual. A continuación se muestra un ejemplo del JSON que recibe la aplicación:

```
{"coord": {
    "lon": -1.64,
    "lat": 42.82
  },
  "sys": {
    "message": 0.0209,
    "country": "ES",
    "sunrise": 1401078831,
    "sunset": 1401132807
  },
  "weather": [
    {
      "id": 801,
      "main": "Clouds",
      "description": "few clouds",
      "icon": "02d"
    }
  ],
  "base": "cmc stations",
  "main": {
    "temp": 286.71,
    "temp_min": 286.71,
    "temp_max": 286.71,
    "pressure": 971.13,
    "sea_level": 1033.78,
    "grnd_level": 971.13,
    "humidity": 86
  },
  "wind": {
    "speed": 3.47,
    "deg": 344.001
  },
  "clouds": {
    "all": 24
  },
  "dt": 1401115722,
  "id": 3114472,
  "name": "Pamplona",
  "cod": 200}
```


Se precisa que la aplicación refresque el tiempo meteorológico de forma que la petición se realice cada 10 minutos. La forma de hacer que un servicio se ejecute cada cierto tiempo es pidiendo al sistema operativo (Android) que avise cuando ese tiempo haya pasado. En Android esto se hace suscribiéndose al sistema de alarmas "AlarmManager". De esta forma, se establece una alarma que se repetirá cada 10 minutos con el método setRepeating. A este método, se le indica el intervalo de repetición y la clase, que extiende de BroadcastReceiver, que escucha dichas alarmas. Así se forma la estructura de la Figura 22.

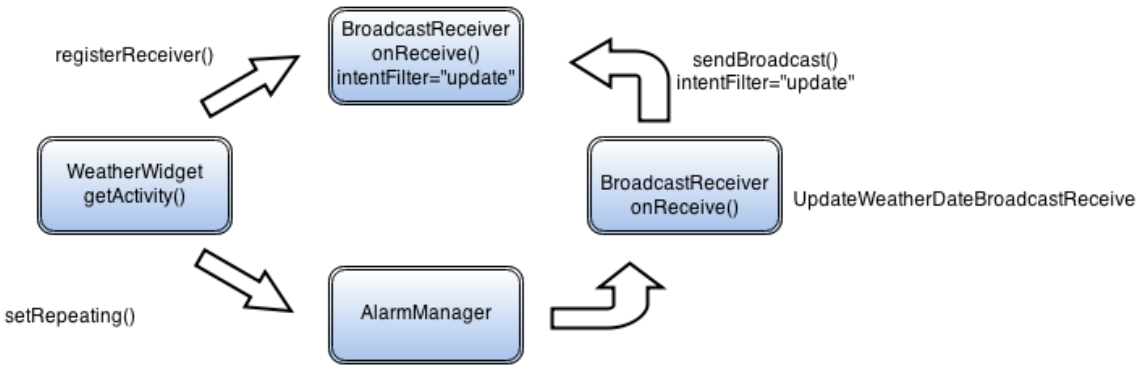

Figura 22 Estructura ejemplo para fijar una alarma repetitva en Android

3.4.2.Aplicación para Wall

En la actividad principal para la versión Wall, TVActivity.java muestra el widget meteorológico WeatherWidget y un WebView a pantalla completa que visualiza la URL de visualización de Concerto para pantallas:

```
http://serverdomain/screen/?mac=<mac>&width=<screen_width>&height=<scr
een_height>
```

Básicamente, se usa el código de Concerto basado en HTML5 y JavaScript. En la URL le pasamos como parámetros el ancho y alto de la pantalla en la que se visualiza. La página de visualización de Concerto utiliza un código JavaScript para refrescar la página. El funcionamiento básico consiste en primero comprobar si la MAC introducida es válida y si no lo es te redirecciona a la página de missing.php. Si la MAC es válida, se obtiene su id de pantalla y se le pasa a signage.js que es el fichero JavaScript principal que se carga en el $(document).ready de JQuery. Signage.js tiene dos métodos principales que son checkScreeen y fetchContent.

El método checkScreen realiza una petición POST a content.php pasándole el id de la pantalla y esté le devuelve un JSON con los campos de dicha pantalla. Además, checkScreen incluye la imagen de fondo de la plantilla haciendo un append de una etiqueta img cuyo atributo src carga lo que devuelve template.php?id_screen=<?>.

El método fetchContent es el que añade contenido a cierto campo o field. Para ello realiza una petición POST a content.php pasándole como parámetros los ids de la pantalla y el campo en concreto. En content.php se calcula el timeline de contenidos que dicho campo va a visualizar. En este apartado, se modifica el timeline que se construye para pantallas de tipo Tablet (type=1) para que solo muestre los contenidos de las categorías Novedades y sus subcategorías. Dependiendo de si el contenido que devuelve content.php tiene como mimetype text o image incluye al código HTML una etiqueta img o un div.

Para poder visualizar además de las imágenes y los textos, las gráficas, fue necesario modificar el fichero signage.js e incluir otro tipo de mimetype que fuera /html/ de forma que cuando se tratara de una gráfica, añadiera un iframe que cargara la página: /admin/app/content/graphic.php?id=<id del contenido>

Se añade un iframe en vez añadir el código HTML debido a que el código JavaScript necesario para dibujar las gráficas sino no se ejecutaría. Para dibujar dichos gráficos en graphic.php se han probado diferentes librerías php y JavaScript. Algunas de ellas generaban una imagen con el gráfico a partir de los datos. Este tipo de librerías se descartan debido a que al generar una imagen dicha imagen debería ser almacenada en el servidor y se debería realizar un script que actualizará cada cierto tiempo dicha imagen para poder captar los posibles cambios del csv. En este sentido, es más sencillo generar la gráfica con código JavaScript de forma que el gráfico siempre esta actualizado. En cuanto a las librerías JavaScript, se probaron varias librerías como: Raphael.js [25], d3.js [26]y Google Charts [27]. Finalmente, se utilizó Google Charts, la que resulto más sencilla

a la hora de personalizar los ejes y ajustar la escala, ya que la apariencia de los gráficos en todas las librerías probadas era similar.

Así, partiendo del id del contenido graphic.php se obtiene el fichero CSV y se parsea trasladando sus datos a un formato de array. Los ficheros CSV deben contener en la primera línea los títulos de los ejes (título del eje x, título de la primera serie de datos y título de la segunda serie de datos) y en la segunda línea, si se desea, la escala a aplicar (se introduce de la forma [ESCALA; valor mínimo del eje y; valor máximo del eje y;]). En las Figura 23 y Figura 24 se puede ver un ejemplo de los gráficos que se obtienen a partir de un CSV con los datos de la Tabla 2 tanto XY como de barras.

año	objetivo	producción
1918	39.9	60
1919	39.48	50
1920	38.1	70
1921	42.86	70
1922	42.95	80
1923	40.59	60

Tabla 2 Datos de ejemplo de un fichero csv

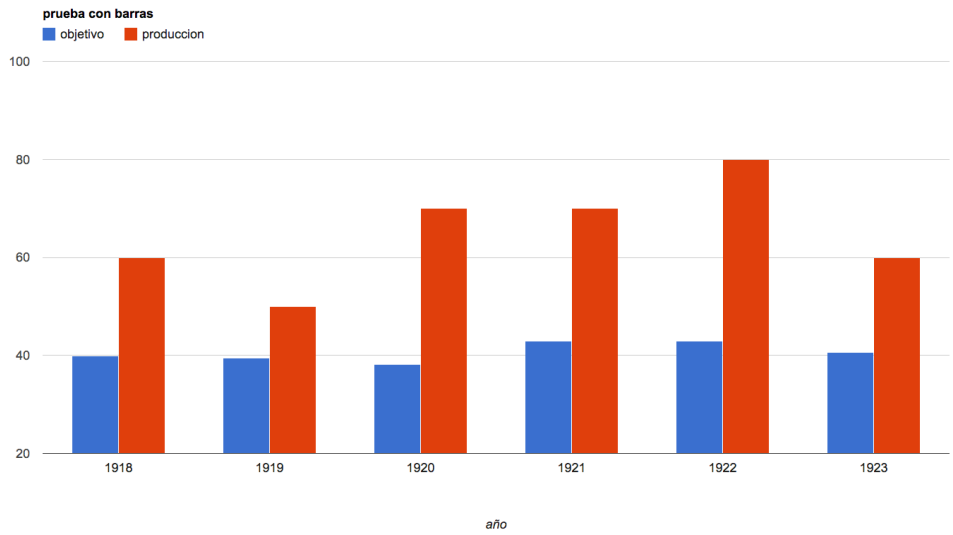

Figura 23 Ejemplo de un diagrama de barras generado a partir de un fichero csv

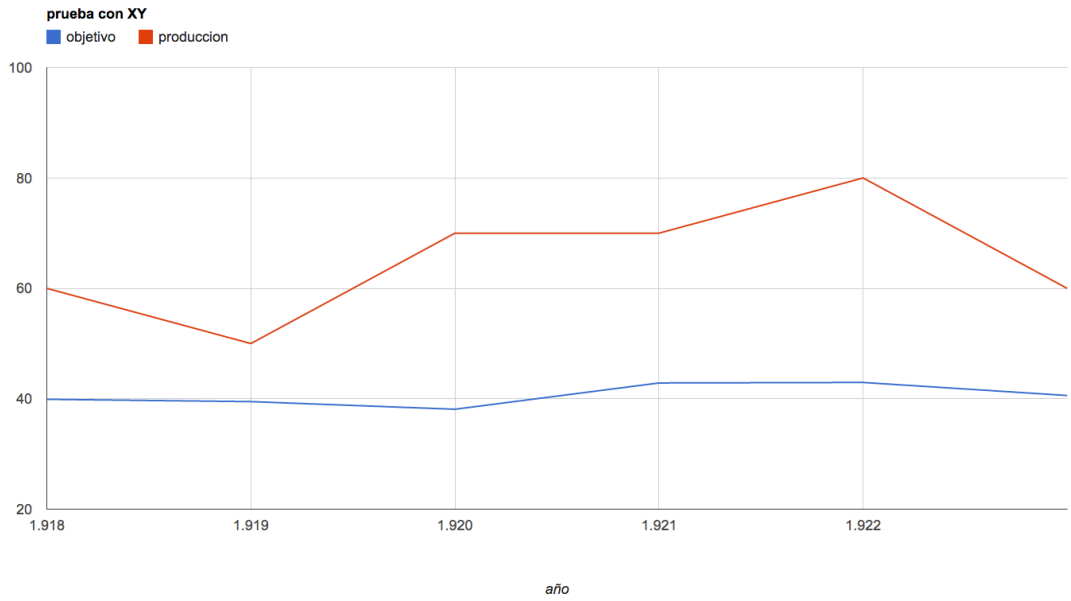

Figura 24 Ejemplo de un gráfico XY generado a partir de un fichero CSV

3.4.3.Aplicación para Tablet

En la actividad principal para la versión Tablet se pretende que el usuario pueda acceder a todas las categorías de contenido a través de un menú lateral. Al hacer click en una categoría se espera que se visualicen todos los contenidos de dicha categoría organizados en páginas. Al hacer click sobre cualquier contenido se espera ver el detalle de dicho contenido en grande. Además, en esta vista se añaden flechas de navegación para que el usuario pueda desplazarse hacia el detalle del ítem anterior o posterior sin tener que volver al listado de contenidos de la categoría.

De esta forma, en TVActivity.java se tienen diferentes partes. La estructura del layout XML que muestra TVActivity es la mostrada en la Figura 25.

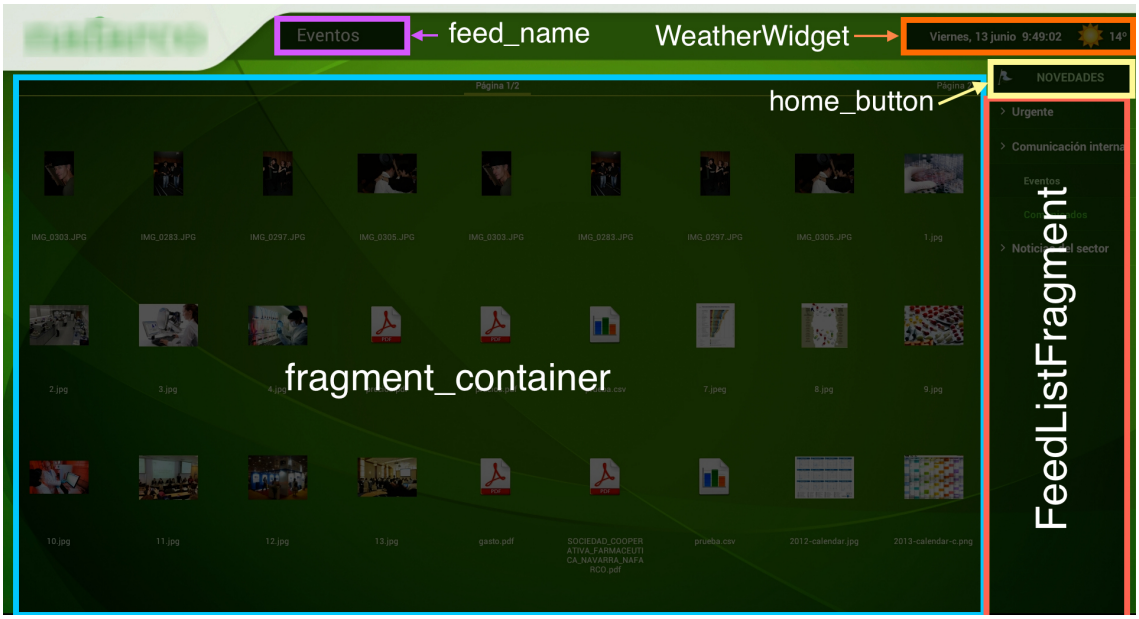

Figura 25 Estructura del layout de TVActivity para tablets

En la figura se distinguen las siguientes áreas:

- **fragment_container:** campo donde se ve el listado de ítems que pertenecen a una categoría o el detalle de cierto ítem o bien el logo corporativo al inicio de la app.

- **feedListFragment:** un menú lateral para navegar por las categorías a las que esté suscrito la pantalla.

- **feed_name:** título de la categoría que se está mostrando

- **weatherWidget:** widget meteorológico desarrollado para las dos aplicaciones cliente.

- **home_button:** botón fijo que carga en fragment_container las Novedades para la Tablet.

A continuación se detalla cómo se ha implementado cada uno de estos campos.

Los fragmentos [28] en Android son interfaces de usuario que representan parte de una actividad, en otras palabras se puede decir que un fragmento es una sub-parte de una actividad. Los fragments corren en el contexto de la actividad en la que se alberguen. En TVActivity se van a embeber 5 fragmentos distintos: FeedListFragment, , IndexLogoFragment, ViewPagerFragment, DetailItemFragment y HomeFragment.

FeedListFragment

Contiene el menú lateral donde se ven todas las categorías disponibles. El fragmento está añadido estáticamente en el layout XML de la actividad. Dicho

fragment realiza una petición al servidor para obtener el listado de categorías en formato JSON a la URL:

```
http://serverdomain/tablet/listfeeds.php?mac=<mac>
```

Con AlarmManager se vuelve a fijar un periodo de refresco para que la Tablet vuelva hacer la petición al servidor y actualice la lista de categorías. A continuación se muestra un ejemplo del JSON enviado por el servidor:

```
{
  "status": "ok",
  "feeds": [
    {
      "id": "26",
      "name": "RRHH",
      "childs": [
        {
          "id": "29",
          "name": "Cita con expofarma",
          "isSubscripted": "true",
          "childs": false
        }
      ],
      "isSubscripted": "false"
    },
    {
      "id": "34",
      "name": "pruebas",
      "childs": false,
      "isSubscripted": "true"
    }
  ]
}
```

Dicho JSON se parsea y FeedListFragment forma el listado con un ExpandableListView. Para definir qué layouts aplicar para las categorías y para las subcategorías se genera un adaptador para la lista. En cada ítem de la lista se fija un listener (OnGroupClickListener y OnChildClickListener para categorías y subcategorías respectivamente) que lanza el método changeContentFragment(feed_id) de TVActivity. Dicho método, changeContentFragment, realiza una petición al servidor a la siguiente URL:

```
http://serverdomain/tablet/listitemsfeed.php?feed_id=<feed_id>
```

Esta petición devuelve un JSON que se parsea y posteriormente se obtiene un array de objetos de tipo ContentItemGridView. Un ejemplo del JSON se muestra a continuación:

```
{
  "status": "ok",
  "items": [
    {
      "id": "295",
      "name": "Expofarma1",
      "path": "295.jpg",
      "mime_type": "image\/jpeg"
    },
    {
      "id": "298",
```

```
      "name": "Expofarma2",
      "path": "298.jpg",
      "mime_type": "image\/jpeg"
    }
  ]
}
```

Un objeto de tipo ContentItemGridView está definido por un id, un nombre del contenido, un fichero asociado o ruta y el tipo de contenido, con las variables asociadas id, name, path y mime_type respectivamente. Cuando la petición se realiza con éxito y se obtiene el array de objetos ContentItemGridView, entonces se llama al método changeToViewPagerFragment(ArrayList<ContentItemGridView> items). Este método remplaza el Fragment existente en el LinearLayout con identificador fragment_container por una nueva instancia de ViewPagerFragment.

fragment_container:

Constituye un LinearLayout que va a contener distintos tipos de fragment como se refleja en la Figura 26. Inicialmente se añade IndexLogoFragment el cual muestra el logo corporativo. Cuando el usuario hace click sobre una categoría ese fragment se sustituye por ViewPagerFragment y cuando el usuario pincha sobre un ítem se reemplaza ViewPagerFragment por DetailItemFragment. Tras un periodo de inactividad del usuario se remplaza el fragment que haya por HomeFragment.

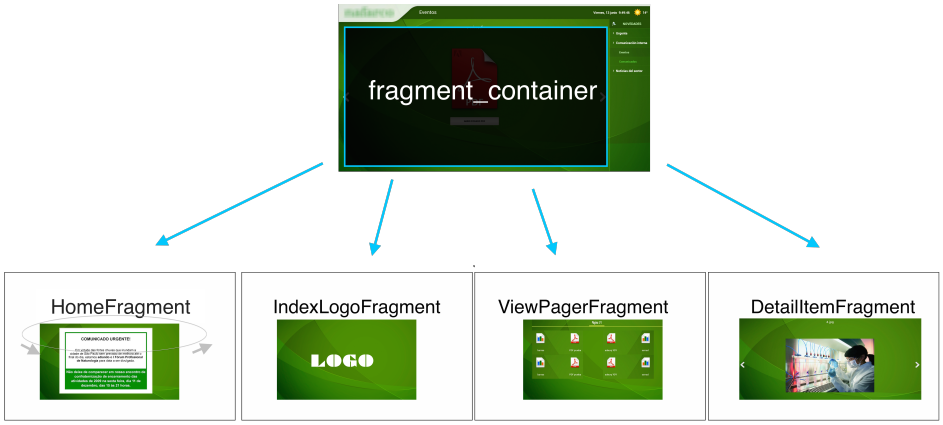

Figura 26 Tipos de fragment que aloja fragment_container

ViewPagerFragment:

El layout que muestra ViewPagerFragment está constituido por un ViewPager y un TitlePageIndicator (elemento indicador de páginas de la librería ViewPagerIndicator [29]). Cuando se genera una nueva instancia de ViewPagerFragment este recibe como parámetro un ArrayList de objetos de tipo ContentItemGridView.

En ViewPagerFragment se calcula el número de páginas que debe formar según el número de objetos que tenga el array de ítems. Conocido el número de páginas, se forma un nuevo array para pasarle al adaptador del ViewPager. En cada posición de dicho array se introduce el array de ítems correspondiente a cada página.

En la Figura 27 se refleja como el adaptador ViewPagerAdapter se encarga de generar una vista propia llamada ContentPageView por cada página y pasarle el array de ítems correspondiente a esa página.

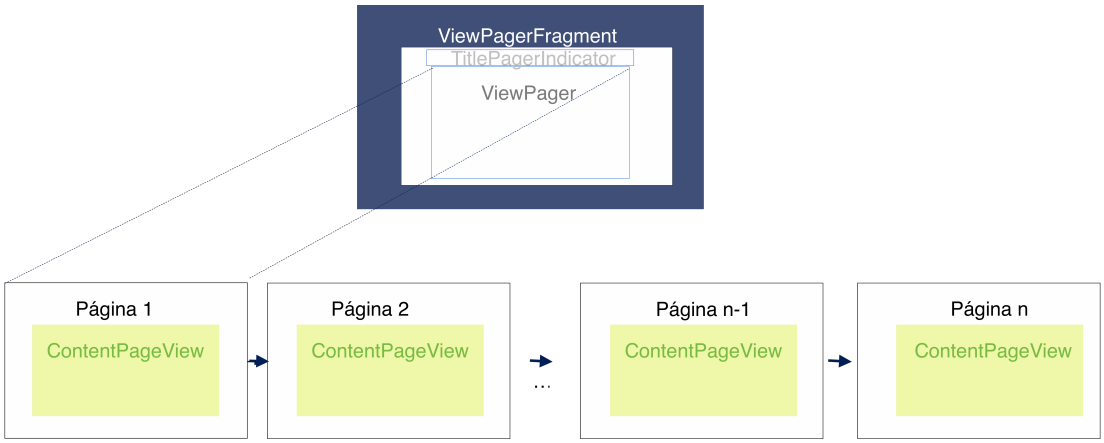

Figura 27 Esquema de funcionamiento del adaptador del ViewPager en ViewPagerFragment

ContentPageView:

ContentPageView consiste básicamente en un GridView. Cuando se genera una nueva instancia de ContentPageView se pasa como parámetro el array de ítems de dicha página. Se define un adaptador para el GridView para establecer qué layout aplicar a cada ítem de la página.

Cada ítem consiste en una imagen y un texto. En el caso de la imagen se define una vista propia llamada ImageViewPicasso. Esta vista propia es necesaria debido a que el contenido puede ser o bien pdf, gráficos o imágenes. En las imágenes mostramos una preview de dicha imagen y en el resto un icono. En la siguiente figura se ve un ejemplo de la visualización de ContentPageView.

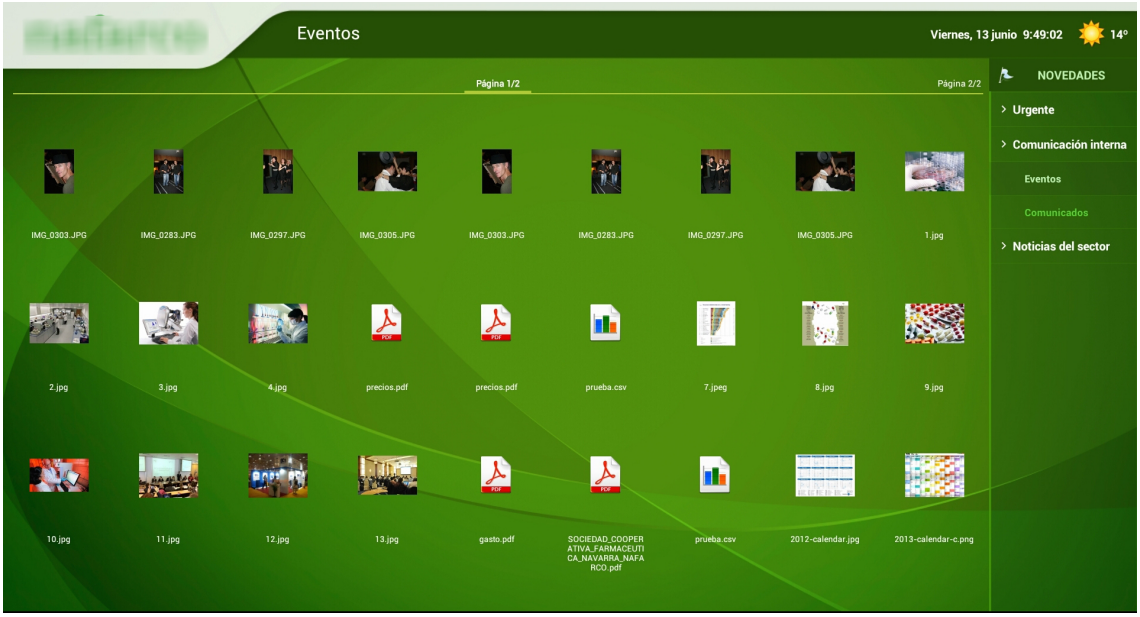

Figura 28 Vista generada por ContentPageView

En ImageViewPicasso se ha definido un método setImage(String image_url,boolean isUrl, int drawable_id) que permite fijar en el ImageView o bien un drawable o una imagen a partir de una URL. Para cargar una imagen desde una URL se ayuda de la librería Picasso[30].

Para utilizar la librería Picasso simplemente hay que incluir el jar descargable desde la web en la carpeta "libs" y añadirla al Build Path del proyecto. La forma de utilizarla es mediante un método estático como se muestra en el siguiente código de ejemplo:

```
Picasso.with(context).load(image_url).into(image,new Callback(){
@Override
public void onError() {
        progressBar.setVisibility(View.GONE);
@Override
public void onSuccess() {
        progressBar.setVisibility(View.GONE);
});
```

- "**with**", recibe el contexto de la app
- "**load**", puede recibir varios parámetros, en este caso se trata de la URL de la imagen a cargar
- "**into**", recibe la ImageView en la que se cargará la imagen y el callback que consta de dos métodos onError, que decide qué acciones realizar si la URL no carga, y onSuccess, que decide qué hacer si la descarga de la imagen se realiza con éxito.

Así, se define que si no se consigue descargar la imagen, fije un icono por defecto. Además la vista contiene un ProgressBar (spinner de carga) que se muestra hasta que la petición a la URL para descargar la imagen sea resuelta y se oculta tanto si falla la descarga, como si se realiza con éxito.

A su vez se añade un Listener al gridview sobreescribiendo el método OnItemClick de forma que al hacer click sobre un elemento se llama al método changeToDetail(id,name) de TVActivity para cambiar el fragment actual de fragment_container por un fragment de tipo DetailItemFragment. En la llamada al método le indicamos el id del ítem seleccionado y el array total de ítems de la categoría a la que pertenece. En el caso de que el ítem sea un PDF, no se llama a changeToDetail sino que se llama a la actividad que visualiza los PDF como se detalla más adelante.

DetailItemFragment:

DetailItemFragment carga un ViewPager con todo el array de ítems de la categoría y muestra en primer lugar la página del ítem seleccionado. Como se ve en la Figura 29 por cada ítem genera una página que carga la vista DetailPageForPDFsView (si el ítem es un PDF), DetailPageForImageView (si es una imagen) o DetailPageForGraphicsView si es un gráfico.

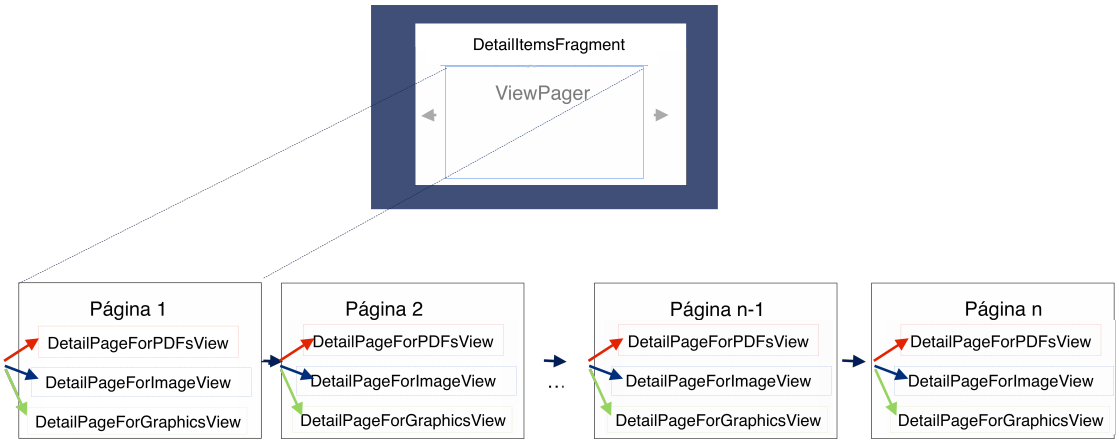

Figura 29 Esquema de funcionamiento del adaptador del ViewPager de DetailItemFragment

DetailPageForImageView:

Si el ítem es una imagen muestra un texto con el título del contenido y la imagen en grande cargando un ImageViewPicasso a partir de la URL:

```
http://serverdomain/tablet/showcontent.php?id= <ítem_id>
```

En la siguiente figura se ve un ejemplo de la visualización de DetailPageForImageView.

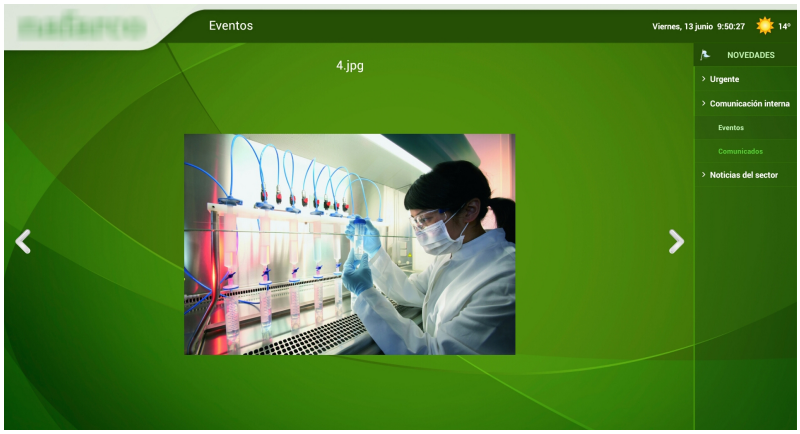

Figura 30 Vista generada por DetailPageForImageView

DetailPageForGraphicsView:

Si el ítem es un gráfico muestra un texto con el título del contenido y una vista propia llamada ConfiguratedWebView que básicamente consiste en un WebView, es decir, un navegador web que carga la URL que se le indique. En este caso carga la URL:

```
http://serverdomain/tablet/showcontent.php?id= <ítem_id>
```

En la vista ConfiguratedWebView se define que en el WebView que contiene active JavaScript y establezca un fondo transparente (por defecto es blanco) y que hasta que la URL se cargue en su totalidad muestre un ProgressDialog. En la siguiente figura se ve un ejemplo de la visualización de DetailPageForGraphicsView.

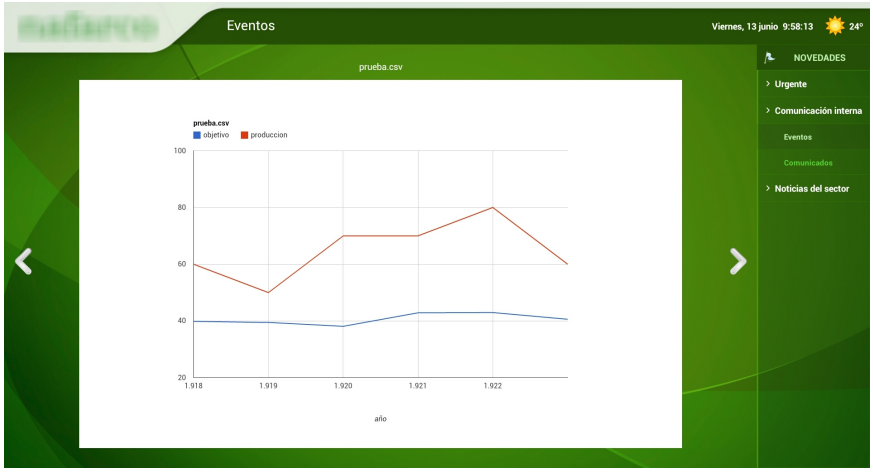

Figura 31 Vista generada por DetailPageForGraphicsView

DetailPageForPDFsView:

Si el ítem es un PDF muestra un icono de un fichero PDF y un Button para abrir dicho fichero. En la siguiente figura se ve un ejemplo de la visualización de DetailPageForPDFsView.

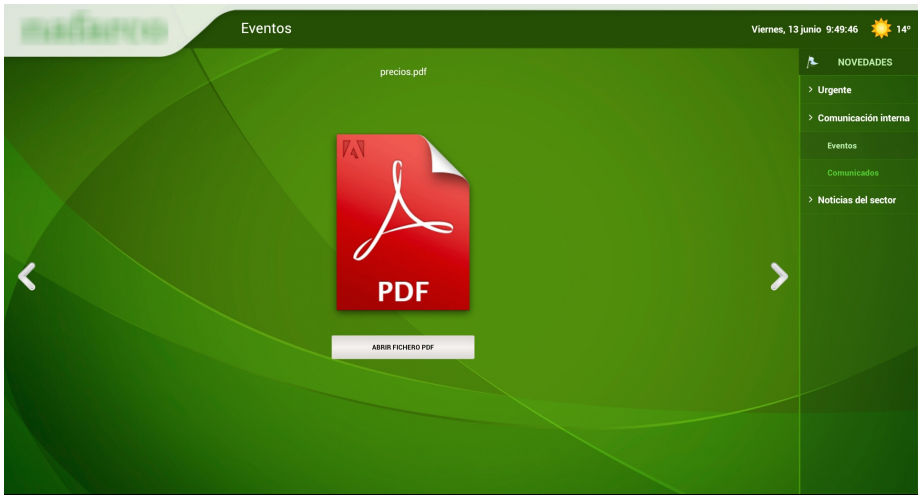

Figura 32 Vista generada por DetailPageForPDFsView

En Android para visualizar un PDF es necesario tener descargada alguna aplicación que lo soporte, como por ejemplo AdobeReader, ya que de por si no es capaz de mostrarlos. De esta forma, una de las opciones para mostrar un PDF es recurrir a una aplicación externa que lo abra, sin embargo, en este caso no nos interesa salir del contexto de nuestra aplicación luego ésta no es una opción. Por tanto, para poder cargar ficheros de este tipo se recurre a librerías externas que permitan mostrar los PDF dentro de tu aplicación. En el proceso de búsqueda se encuentran muchas librerías de pago, sin embargo sólo se ha intentado aplicar una gratuita. La mayoría de librerías encontradas, algunas de ellas JavaScript como pdf.js, han sido probadas y se han obtenido resultados muy negativos.

Finalmente tras un proceso de pruebas, se decide incorporar la librería MuPDF [31] que es Open Source excepto para usos comerciales. En este proyecto sí se va a destinar su uso a un fin comercial, por ello, se consulta vía correo electrónico a través del enlace que facilitan en su página el precio de la librería. En la contestación aseguran que no tienen un modelo de negocio para cobrar su uso en este caso en concreto por lo que se puede utilizar en principio de forma gratuita.

En la documentación de MuPDF [32] se explica cómo aplicar esta librería en Android. Las instrucciones de su página oficial indican cómo compilar la librería con JDK para obtener un fichero .lib pero no se detalla claramente cómo se incluye al proyecto, ni cómo utilizarla. Por ello, se decide seguir las indicaciones de otra fuente no oficial partiendo del proyecto ejemplo de Android encontrado [33]. Dicho proyecto contiene una aplicación que descarga desde una URL un PDF y realiza un Intent, es decir, una llamada a una actividad llamada MuPDFActivity que, utilizando clases de la librería MuPDF, genera la vista del PDF. En el PDF se puede hacer Zoom, pasar páginas deslizándose de izquierda a derecha y viceversa e incluso buscar contenido haciendo click en el bóton superior de búsqueda. Se puede ver un ejemplo de la vista generada por MuPDFActivity en la Figura 33.

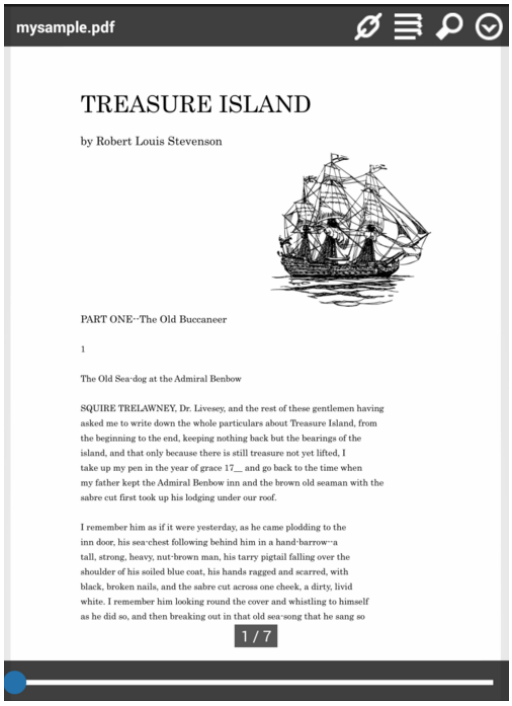

Figura 33 Vista generada por MuPDFActivity

De esta forma, se genera un proyecto basado en el comentado anteriormente que se define como DigitalSignage-MuPDF y se importa como librería externa. Así, una vez incorporada la librería, al pulsar el botón para abrir el PDF se descarga el PDF en concreto de la URL:

```
http://serverdomain/content/images/ <ítem_id>.pdf
```

Tras descargarlo, se guarda en la tarjeta de memoria del dispositivo con nombre "<id>.pdf ". Si el PDF ya había sido descargado anteriormente no vuelve a descargarlo. Una vez se tiene el PDF guardado, se realiza un Intent a la actividad MuPDFActivity comentada anteriormente. En el layout de dicha actividad se incluye un botón en la parte superior izquierda para que el usuario pueda volver a la vista anterior cuando termine de ver el PDF. Dicho botón simplemente lanza el método OnBackPressed() que es el método al que llama Android cuando un usuario pulsa el botón back de la barra de sistema del dispositivo.

HomeFragment:

Por último, queda por describir HomeFragment que es el fragment que se carga en fragment_container tras un periodo de inactividad del usuario. Para detectar la inactividad del usuario se utiliza la clase CountTimerDown que ejecuta una cuenta atrás y cuando llega a cero ejecuta el método changeToHomeFragment que remplaza el fragment actual de fragment_container por un HomeFragment. Si el usuario toca la pantalla, se detecta a través del método de Android OnUserInteraction() y se reinicia la cuenta atrás. HomeFragment consta simplemente de un ConfiguratedWebView que carga la URL de visualización de pantallas siguiente:

```
http://serverdomain/screen/?mac=<mac>&width=<screen_width>&height=<scr
een_height>
```

4. Conclusiones y trabajos futuros

4.1. Conclusiones

A la conclusión de este proyecto, hacemos balance entre los objetivos planteados al inicio del mismo y los que se han llegado a conseguir.

El objetivo principal de este proyecto siempre fue el de conseguir un producto real utilizable por el cliente. Una vez finalizado este proyecto, se puede afirmar que dicho propósito ha sido conseguido con éxito, pues se tiene un sistema Digital Signage que funciona según los requisitos iniciales del cliente. Cabe decir, que el sistema aún no ha sido implantado a falta de incorporar nuevos cambios que solicita el cliente para el que se ha desarrollado este proyecto.

En el inicio de este proyecto se analizó cómo definir la arquitectura del sistema y para ello se realizó un estudio de las soluciones existentes en el mercado. Dada la gran cantidad de posibles tecnologías para el desarrollo de soluciones de Digital Signage, la investigación constituyó una etapa muy importante. La idea inicial era empezar desde cero, sin embargo, al ver la complejidad de un sistema de este tipo se propuso partir de una solución OpenSource existente que ayudara a implantar el sistema de forma rápida. En este sentido, cabe decir que indudablemente, este proyecto no podría haber sido terminado si se hubiera decidido partir de cero, es decir, sin aplicar y reutilizar una solución ya existente.

Se realizaron una gran cantidad de pruebas antes de elegir la plataforma a utilizar y al encontrar la plataforma de Concerto y partir de ella se consiguió una base sobre la que avanzar muy rápidamente. La principal funcionalidad añadida al sistema de Concerto ha sido la de incorporar un nuevo tipo de pantallas donde el usuario elige qué contenido visualiza.

La elaboración de este proyecto me ha servido para valorar que cualquier inicio de investigación se hace mucho más dificultoso de no existir estudios anteriores. Es decir, resulta más productivo trabajar sobre proyectos ya realizados ampliando o mejorando las conclusiones. Aunque cabe decir que trabajar sobre un código no realizado por ti mismo resulta una tarea ardua en sus inicios.

Además, valoro el aprovechamiento de librerías externas como una medida muy importante a la hora de agilizar al máximo el trabajo y mejorarlo, ya que estas han sido testeadas con anterioridad.

La realización de las aplicaciones cliente (para Wall y para Tablet) me ha permitido ampliar mis conocimientos de programación en la plataforma Android (trabajar con librerías externas, peticiones a Web Services, fragments, hacer widgets, etc). Al mismo tiempo la colaboración del personal de la empresa me ha permitido mejorar mis métodos de programación, enseñándome a organizar el proyecto separando el código en diferentes paquetes, utilizando un nombrado de strings adecuado, etc, en definitiva, a realizar un código más limpio.

4.2. Trabajos futuros

El prototipo de sistema desarrollado fue diseñado de tal forma que en un futuro se puedan realizar mejoras y actualizaciones. A continuación se listan una serie de trabajos futuros que mejorarían la solución propuesta en este proyecto:

- Incluir el protocolo HTTPS para garantizar la seguridad de los datos que maneja el sistema en la comunicación entre pantallas cliente y servidor.
- Ofrecer la posibilidad de incluir videos a través del portal web de administración.
- Ofrecer la posibilidad de elegir el orden deseado en el que tienen que aparecer los contenidos en el modo Wall.
- Incluir un buscador en la aplicación cliente de Tablet para buscar ficheros de manera sencilla.
- Incluir la opción de ordenar el contenido en las Tablet de distintas formas: orden alfabético, por fecha de subida o por fecha de inicio de visualización.
- Hacer que las aplicaciones cliente funcionen offline a través de la caché de la aplicación, guardando recursos para la aplicación en el sistema del cliente, de modo que estén disponibles incluso cuando Internet no lo esté.

5. Referencias bibliográficas

[1] Definición de "Digital Signage" (Wikipedia): http://es.wikipedia.org/wiki/Se%C3%B1alizaci%C3%B3n_digital

[2] Página web oficial de Concerto Digital Signage: https://www.concerto-signage.org/

[3] Repositiorio del código de Concerto: https://github.com/concerto/concerto_gtv

[4] Demostración del portal de administración de contenidos de Concerto: http://demo.concerto-signage.com/admin/index.php/

[5] Demostración de la visualización en pantalla http://demo.concerto-signage.com/screen/?mac=00:00:00:00:00:EE

[6] Guía de instalación de Concerto: https://github.com/concerto/concerto/wiki/Installing-Concerto-2

[7] Pasos a seguir para realizar una instalación manual de Concerto: https://github.com/concerto/concerto_v1/wiki/Installation

[8] Página web oficial de Xibo Digital Signage: http://xibo.org.uk/

[9] Pasos para instalar el servidor de Xibo Digital Signage: http://wiki.xibo.org.uk/wiki/Install_Guide_Xibo_Server

[10] Link para descargar Xibo: https://launchpad.net/xibo

[11] Página web oficial de RiseVision: http://www.risevision.com/

[12] Portal de administración de RiseVision: http://rva.risevision.com/

[13] Repositorio de RiseVision: https://code.google.com/p/risevision/

[14] Instaladores de la aplicación cliente de RiseVision: http://www.risevision.com/player/?utm_source=tour&utm_medium=player-link&utm_campaign=Player

[15] Página oficial del stick Tronsmart CX-919: http://www.tronsmart.com/Item/63

[16] Página oficial del stick Rikomagic MK802: http://www.rikomagic.co.uk/

[17] Página oficial del stick Cotton Candy: http://www.fxitech.com/cotton-candy/what-is-it/

[18] Libería JavaScript para generar spinners de carga: http://fgnass.github.io/spin.js/

[19] Librería JavaScript para introducir fechas en formularios: http://jqueryui.com/datepicker/

[20] Función PHP gettext: http://php.net/manual/es/book.gettext.php

[21] Poedit, editor de traducciones para Gettext: http://poedit.net/

[22] Librería DatePicker de JQuery según el idioma: http://jquery-ui.googlecode.com/svn/tags/latest/ui/i18n/

[23] Clase Paginator.php para añadir paginado: https://github.com/wallabag/wallabag/blob/master/inc/3rdparty/paginator.php

[24] API para obtener el tiempo meteorológico: api.openweathermap.org

[25] Librería JavaScript de gráficos Raphael.js: http://g.raphaeljs.com/

[26] Librería JavaScript de gráficos d3.js: http://d3js.org/

[27] Librería JavaScript de gráficos de Google Chart: https://developers.google.com/chart/

[28] Definición de fragmentos en Android: http://developer.android.com/guide/components/fragments.html

[29] Librería ViewPagerIndicator: http://viewpagerindicator.com/

[30] Librería para carga asíncrona de imágenes en Android: http://square.github.io/picasso/

[31] Página oficial de la librería MuPDF: http://www.mupdf.com/

[32] Documentación de la librería MuPDF para visualizar PDFs: http://www.mupdf.com/docs/how-to-build-mupdf-for-android

[33] Proyecto ejemplo de uso de la librería MuPdf en Android: http://iti.hatenablog.jp/entry/2013/11/14/113953