UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITECNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

ESCOLA POLITÈCNICA
SUPERIOR DE GANDIA

“Configurador para equipos de recepción de televisión satelital 'on road'”

TRABAJO FINAL DE GRADO

Autor/a:
Luis Martínez López

Tutor/a:
Jordi Bataller Mascarell
Salvador Ferrairó Castelló

GANDIA, 2018

Resumen

La recepción de televisión vía satélite precisa, evidentemente, que el aparato receptor esté correctamente orientado a un satélite emisor; en concreto, a aquél que emita los canales en los que un usuario está interesado. Desde el punto de vista del usuario, es justamente el tipo de canal, idioma o temática la que determina qué satélite utilizar. En el caso de equipos móviles (en autobuses, trenes o barcos), determinar qué satélites son utilizables y qué canales tienen disponibles se complica, ya que no todos los satélites son visibles en todas las zonas geográficas. En este trabajo, hemos desarrollado una aplicación que encuentra qué canales están disponibles en un trayecto determinado. El usuario de esta aplicación marca sobre un mapa el recorrido sobre el cuál desea obtener información, qué idiomas son de su interés y qué tipo de receptor tiene. Con estos datos, el programa presenta la lista de canales sintonizables en el trayecto que cumplan los requisitos.

Abstract

Satellite television reception needs, obviously, reception equipment to be correctly oriented to the emitter satellite; specifically, to which emits channels the user is interested in. From the user viewpoint, the reason that determines which satellite will be used is the channel kind, language or topic. On mobile equipment case (buses, trains or ships), determining which satellites are usable and which channels are available is complicated, because all satellites are not visible in every geographic area. In this work, an application has been developed, which finds available channels for a determined route. User marks over a map the path that wants to get information about, which languages is interested in, and which type of receiver has. With this data, the program shows a list with the tunable channels that meet the requirements.

Palabras clave, keywords

Televisión, satélite, transporte, canales, base de datos.

Television, satellite, transport, channels, database.

Contenido

1. Introducción

Como respuesta a la actual demanda de disponer de medios de entretenimiento de forma ubicua, también en medios de transporte que habitualmente no disponían de este tipo de servicios o lo ofrecían de forma muy limitada, la empresa Azimut Electronics lanzó su línea de negocio 'Azimut On Road'. Estos productos permiten embarcar sistemas de entretenimiento en medios de transporte (autobuses, trenes, barcos, etc.). Entre los servicios que ofrecen estos sistemas (como internet, películas a la carta, prensa digital) se encuentra la televisión por satélite, por su variedad, internacionalidad y omnipresencia (en cualquier punto del planeta habrá, al menos, un satélite visible).

Las características de la tecnología satelital determinan la necesidad de visibilidad directa entre el emisor (los satélites geoestacionarios que actúan como repetidores de los centros de producción) y el receptor (el usuario que detecta la señal con su antena parabólica para disfrutar del contenido audiovisual ofrecido). Esta cuestión limita la oferta de canales comerciales dependiendo del área geográfica en la que el usuario se encuentre, o en nuestro caso, por la que vaya a moverse.

La necesidad concreta es poder determinar qué canales (incluyendo información relevante como idioma y determinados datos técnicos) estarán disponibles en todos los puntos de un recorrido. Hacerlo manualmente buscando la información, siendo ésta muy cambiante además, para cada satélite es tedioso y probablemente no actualizado.

Es por este motivo que en este proyecto pretendemos automatizar esta búsqueda, proponiendo de manera instantánea paquetes de canales comerciales disponibles y compatibles con una misma configuración. Así pues, vamos a desarrollar una herramienta cuyo fin último será ofrecer la posibilidad de que el cliente (la empresa de transporte) dada la disponibilidad de canales para una ruta, elija qué paquete de canales desea recibir, de forma que los técnicos de Azimut configuren correctamente el sistema de recepción para ese trayecto.

La organización de esta memoria es la siguiente. En el capítulo 2, "Análisis de requerimientos", detallamos qué características concretas y funcionalidad debe tener la aplicación. El capítulo 3, "Herramientas", recoge los programas y bibliotecas utilizados en el desarrollo, así como la versión de los mismos. A continuación, en el capítulo 4, "Diseño", presentamos la estructura del programa, detallando cómo son los módulos que lo componen, cómo se relacionan y qué debe hacer cada uno. El capítulo 5, "Implementación", explica el código que implementa el diseño del capítulo 4. Aquí presentamos los recursos desarrollados, cómo se concatenan las llamadas a funciones y algunos problemas resueltos durante el desarrollo. Posteriormente, el capítulo 6, "Guía de uso", pretende ser un manual, en primer lugar, para la instalación, pero también para el uso de la aplicación. Finalmente, en el capítulo 7, "Conclusiones y trabajo futuro" resumimos qué objetivos hemos cumplido en este proyecto y plantemos una serie de mejoras futuras.

2. Análisis de requerimientos

En este capítulo se describen las características que debe tener la aplicación, así como los objetivos concretos que debe cumplir, siguiendo las indicaciones de Azimut Electronics.

Desde el principio, el *software* está pensado con dos tipos de interacción entre el mismo y el usuario: introducir el dibujo de las huellas de cobertura e introducir la ruta de un bus para conocer los canales de televisión que encontrará disponibles. Una tercera actividad que debe tener lugar en el sistema es la importación a la base de datos propia de la información contenida en el archivo CSV provisto por un tercero.

2.1. Canales disponibles en la ruta

La imagen que sigue (Figura 1) es el esbozo de lo que debía ser la interfaz para introducir la ruta sobre la cual queremos saber los canales satelitales disponibles en los idiomas seleccionados.

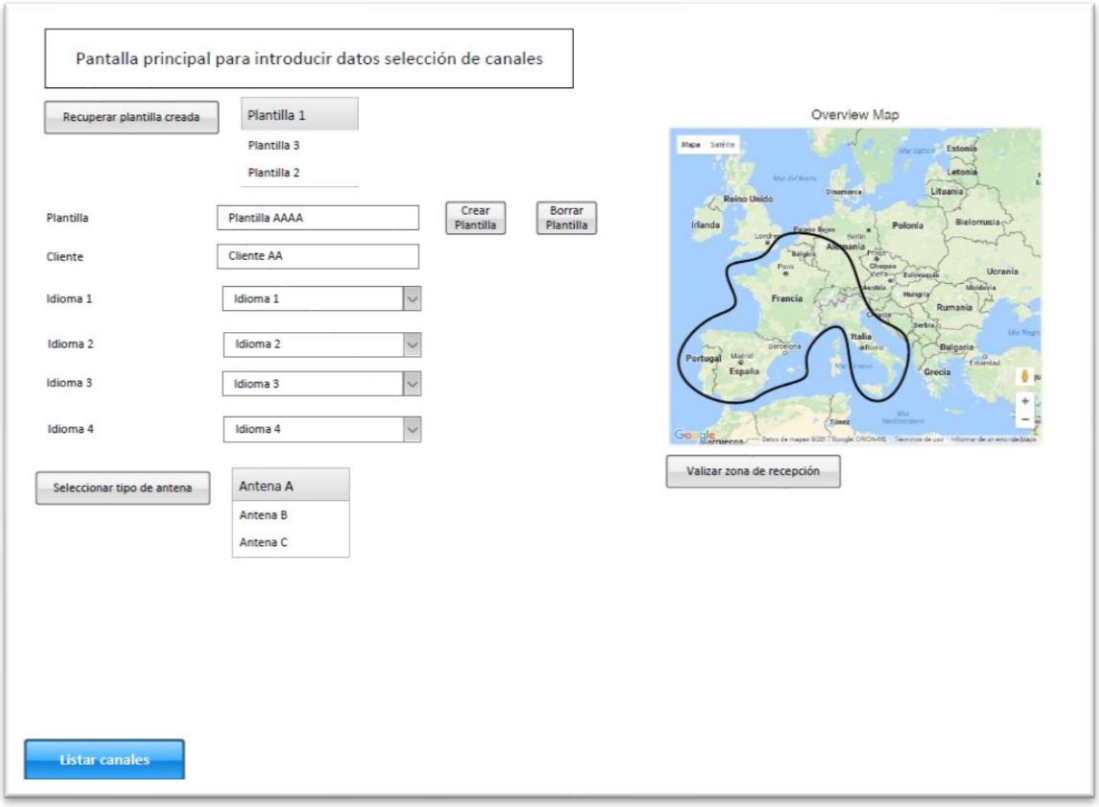

Figura 1

Este esquema original fue modificado, dejando el desarrollo del código que implementaría los campos "Plantilla" y "Cliente" para una fase posterior. Otros cambios son meramente visuales, respecto a la organización de los elementos que forman la interfaz de usuario.

La información que debe listar como *output* el sistema son los siguientes campos: nombre del canal, web del canal, idiomas del canal, satélite al que corresponde el canal, sistema de codificación o FTA (si el canal se emite en abierto o con qué encriptación), tipo (TV o radio), frecuencia, polaridad y TID (identificador de transpondedor).

2.2. Áreas de cobertura

A continuación, la figura 2 corresponde al primer boceto de lo que iba a ser la interfaz para introducir las áreas de cobertura de los satélites.

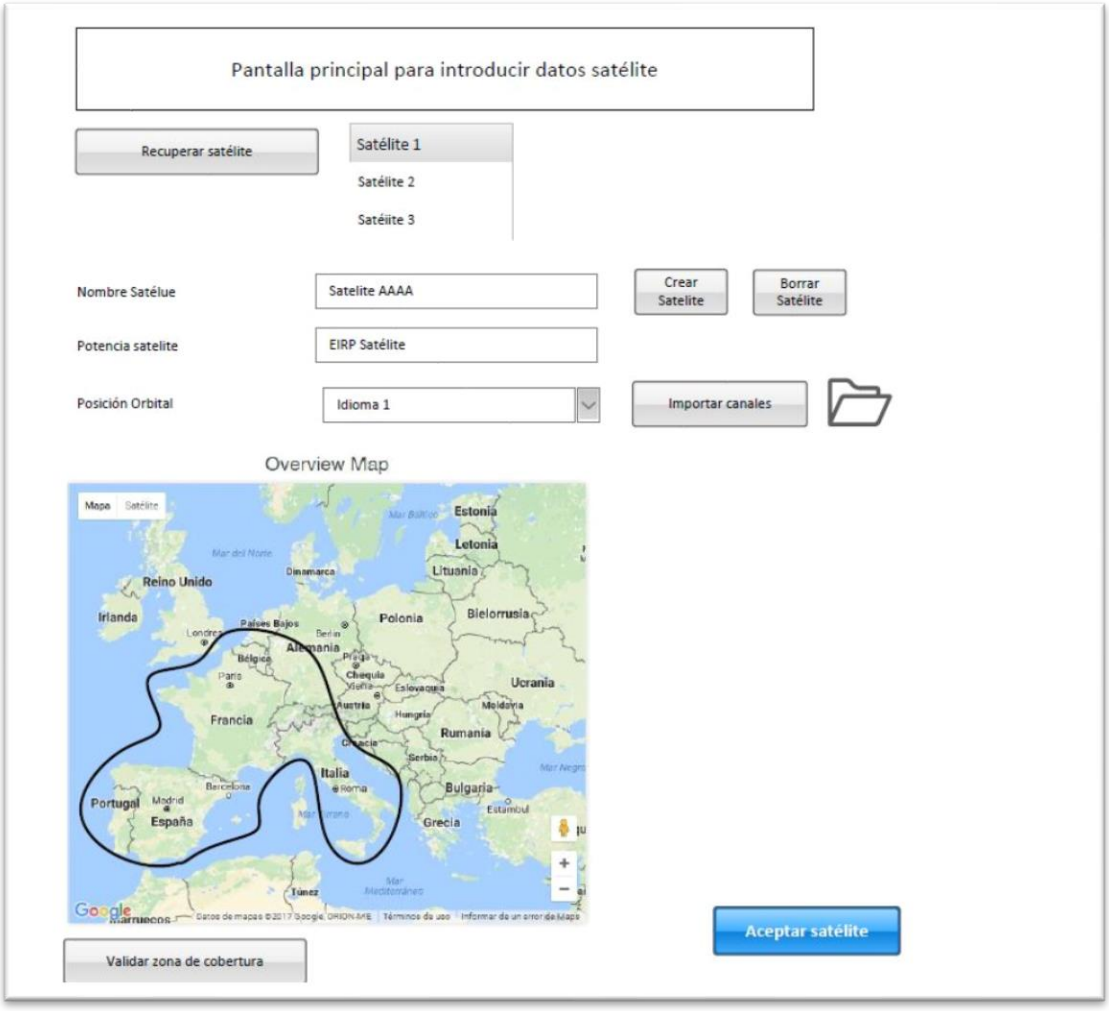

Figura 2

Por su parte, este boceto fue modificado para que la información del transpondedor que acompaña a las coordenadas del área se pueda seleccionar de un desplegable, en lugar de ser mecanografiada por el usuario. Evitamos así posibles errores de escritura y al mismo tiempo listamos los transpondedores existentes.

Reseñaremos también, que la unidad de emisión con la que trabajamos finalmente no es el satélite sino cada uno de los transpondedores que éste alberga.

Por último, la atribución de "Importar canales" no debía recaer sobre el usuario sino sobre el administrador que actualiza la base de datos, por eso desaparece de la interfaz esa opción.

2.3. Provisionamiento de la base de datos

El proveedor de información externo de donde obtenemos todos los datos sobre satélites DVB (posición, canales, idiomas, etc.) nos ofrece una utilidad mediante la que descargaremos mensualmente un fichero CSV actualizado. Nuestro sistema precisa importar esa información, de modo que se requiere un puente que reciba el fichero CSV e introduzca su información en la base de datos SQL. Esta labor será llevada a cabo por el administrador, pudiéndose también programar la ejecución del *script* que ejecuta la tarea periódicamente.

3. Herramientas

En el presente capítulo se enumeran las herramientas informáticas empleadas durante el desarrollo del proyecto.

El entorno en el que se han desarrollado las distintas piezas de código ha sido Netbeans 8.2 sobre Ubuntu 14.04 LTS.

La base de datos es SQL, en una distribución *open source* denominada MariaDB. La versión 15.1 Distrib 5.5.57-MariaDB para Debian.

El procesamiento está sostenido por Node.js (versión 0.10.25), un intérprete de JavaScript para el lado del servidor. Además, hemos utilizado un par de librerías para Node.js:

- express v4.13.4 para mantener el servidor a la escucha e identificar los distintos tipos de peticiones y URLs.
- MySql v2.13.0 para conectar la base de datos mariaDB con el servidor JavaScript.

También hemos empleado librerías del lado del cliente, a saber:

- XMLHttpRequest para lanzar peticiones REST desde el cliente.
- MapBox y MapBox-gl para incluir mapas interactivos en la interfaz de usuario, y para poder dibujar polígonos y recoger las coordenadas de sus vértices, respectivamente.
- Tablefilter para poder filtrar la tabla expuesta con la información de los canales.

Por su parte, la interfaz de usuario es HTML plano ya que sólo se pretendía dotarla de funcionalidad.

4. Diseño

En este capítulo se describirán las piezas que conforman la aplicación y la relación entre ellas que permite el funcionamiento como conjunto. Esta parte del desarrollo sienta las bases de la estructura de la aplicación, por lo que este capítulo resulta fundamental para entender cómo funciona ésta.

4.2. Esquema general

La figura 3 ilustra el esquema general de la aplicación. Como podemos ver está compuesta por siete módulos propios.

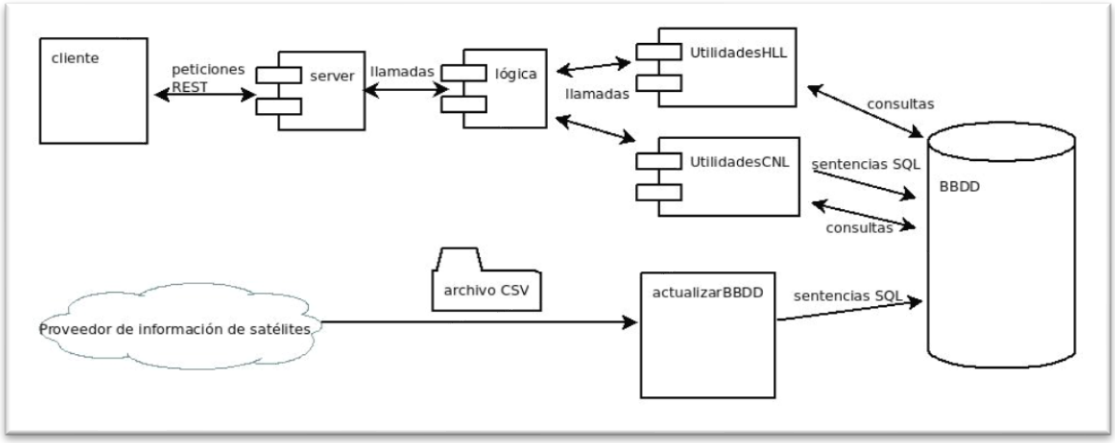

Figura 3

El funcionamiento es simple. La base de datos es alimentada fundamentalmente por el módulo actualizarBBDD, que es quien se encarga de tomar la información del archivo CSV proporcionado por el proveedor y convertirla en sentencias SQL.

La base de datos se encuentra en el corazón de la aplicación, ya que es donde se almacena la información que da utilidad a ésta.

UtilidadesCNL y UtilidadesHLL están a la misma altura en cuanto a la estructura se refiere, su diferenciación es meramente organizativa y responde a la naturaleza bicéfala del cliente. Como se puede intuir por los nombres, un archivo se encarga de aquellas consultas que provienen de la parte del cliente referidas a los canales disponibles mientras que la otra se encarga de aquellas que conciernen a las áreas de cobertura.

El puente que determina a qué funciones de utilidad hay que llamar y en qué orden es lógica.

La función de server es escuchar en espera de peticiones por parte del cliente y llamar a una función de lógica según la naturaleza de la petición recibida.

El cliente es la interfaz con el usuario. Proporciona un mecanismo gráfico de interacción con la aplicación. Está compuesto de dos páginas web, una para pedir canales disponibles en una ruta y otra para introducir áreas de cobertura.

4.3. Protocolo

Para las comunicaciones entre la parte del cliente y la del servidor hemos utilizado un protocolo tipo REST (véase *A Brief Introduction to REST*). Las peticiones siempre nacen del cliente y siempre obtienen una respuesta del servidor, al menos un mensaje de OK. Aquellas flechas que sostienen texto corresponden a mensajes que transmiten la información indicada en formato JSON en el cuerpo del mensaje; aquéllas sin etiquetar son mensajes de control.

Tal como ilustra la figura 4, el cliente está dividido en dos páginas, una de ellas para buscar los canales disponibles en una ruta determinada mientras que la otra tiene la finalidad de manipular la base de datos de huellas de transpondedores.

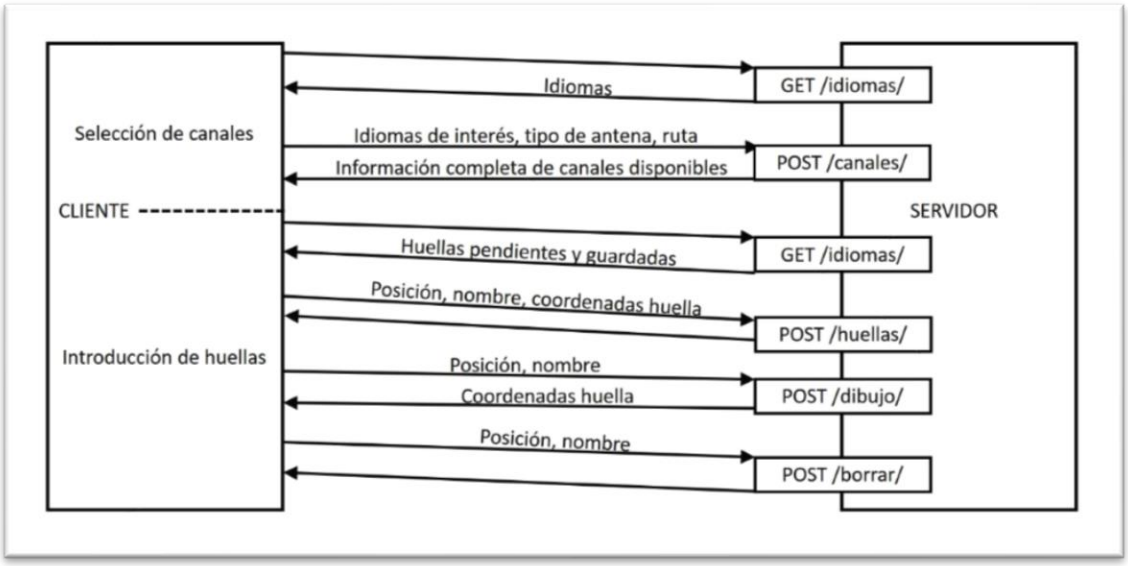

Figura 4

"GET Idiomas" devuelve al cliente todos los códigos de idioma que hay almacenados en la tabla CanalIdioma (véase 4.4.), para mostrárselos al usuario y que pueda elegir sólo entre los idiomas disponibles.

Mediante "POST Canales", el cliente envía al servidor los idiomas que interesan al usuario y el tipo de antena receptora que equipa, así como las coordenadas que describen la ruta geográfica que va a seguir. La respuesta del servidor incluye la lista de canales con su información completa (véase 4.1.).

"GET Huellas" devuelve al cliente de forma diferenciada las parejas posicion-nombre_huella (véase 4.4.) de aquellas huellas de las que conocemos el área de cobertura y de aquellas que tienen el área de cobertura por determinar.

"POST Huellas" traslada al servidor un objeto con las coordenadas que describen el área de cobertura. Este dibujo descriptivo tendrá tres niveles que diferencian la potencia de recepción requerida en cada zona. Junto con las coordenadas, incluiremos la posición

orbital y el nombre (posicion-nombre_huella) que identifican a la huella que estamos tratando.

El mensaje "POST Dibujo" envía desde el cliente la posición y nombre de la huella que se desea visualizar, siendo respondido por el servidor con el objeto que contiene las coordenadas que definen dicha huella.

Por último, "POST Borrar" envía la identificación de la huella, como siempre: posicion-nombre_huella. El servidor elimina el registro correspondiente en la tabla Huella (véase 4.4.), de forma que esa huella pasará a estar pendiente de que se introduzca su área de cobertura.

4.4. Base de datos

La base de datos está compuesta por cuatro tablas como muestra la figura 5. En ellas podemos ver los distintos campos junto al tipo de dato que guardan. Los puntos negros junto al campo implican que no puede ser nulo, y los campos subrayados se corresponden con las claves primarias (varios campos subrayados en una misma tabla conforman una clave primaria todos ellos concatenados). Identificamos unívocamente un canal determinado por su (TID, SID, VID) en la tabla Canal, y sus idiomas en la tabla CanalIdioma.

Relacionamos un canal con el transpondedor que lo emite mediante el campo TID, en las tablas Canal y Transpondedor. Relacionamos un transpondedor con su huella a través de los campos (posicion, nombre_huella) en las tablas Transpondedor y Huella.

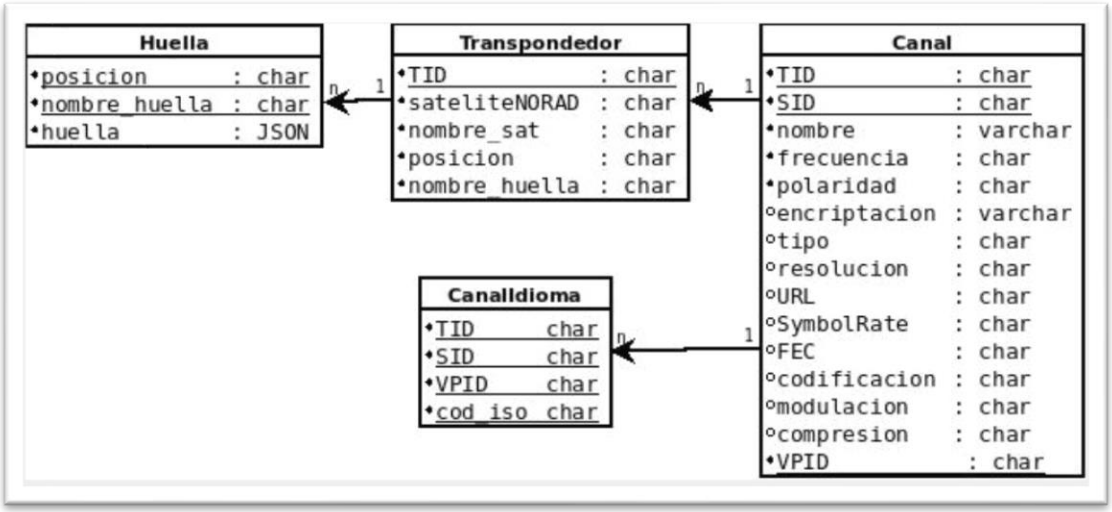

Figura 5

4.5. Algoritmos

Cada tipo de mensaje descrito en el protocolo requiere un proceso para darle respuesta o efectuar en la base de datos los cambios indicados. Con el fin de mantener un código limpio y ordenado, las tareas se organizarán en diversos archivos, como describe el punto 7. Aquí vamos a detallar en pseudocódigo qué trato hay que dar a cada petición.

4.5.1. GET Idiomas

La figura 6 representa el flujo seguido por una petición “GET /idiomas/”. En este caso el cuerpo de la petición está vacío ya que no se necesita ninguna información por parte del cliente. Así pues, la ejecución llevada a cabo es siempre exactamente igual, variando tan sólo su respuesta en función del estado de la base de datos.

```
1  identificar petición como "GET /idiomas/"
2  consultar a la BBDD por los idiomas disponibles
3  responder el resultado de la consulta
```

Figura 6

4.5.2. POST Canales

La figura 7 representa el algoritmo seguido por una petición "POST /canales/". El cuerpo de la petición debe indicar los idiomas que interesan al usuario (hasta 4 idiomas), así como las coordenadas que describen la ruta que seguirá.

Inicialmente, se consulta la base de datos para obtener una lista con los identificadores de los transpondedores que ofrecen los idiomas precisados, para después cribarla según si éstos ofrecen cobertura para toda la ruta indicada por el usuario. El orden no es casual, sino que responde a eficiencia, ya que son necesarias más operaciones para saber si la ruta está dentro de un área determinada que para saber si un transpondedor ofrece ciertos idiomas. Así, conseguimos que el proceso más pesado tenga lugar sobre una cantidad de datos menor.

Una vez acotada por las condiciones de idiomas y cobertura la lista de identificadores, obtenemos la información completa de los canales que ofrecen los transpondedores incluidos en ésta. Devolvemos a la parte de cliente esta información completa.

```
identificar petición como "POST /canales/"
consultar BBDD para obtener los TID que ofrecen los idiomas requeridos
for cada elemento de la lista TID devuelta por la consulta previa
    consultar BBDD para obtener la huella del TID
    comprobar que la ruta esté en la huella
    si la ruta está en la huella
        acumular TID en la lista de respuesta
    fin de si
fin de for
obtener la información completa de los canales de la lista de respuesta
responder la información completa de los canales
```

Figura 7

4.5.3. GET Huellas

La figura 8 muestra el algoritmo seguido por una petición "GET /huellas/". El cuerpo de esta petición también está vacío por el mismo motivo que en GET Idiomas. Asimismo, siempre realiza las mismas consultas a la base de datos: identificadores de las huellas guardadas en la tabla Transpondedores e identificadores de las huellas guardadas en la tabla Huellas (identificadores son parejas posicion-nombre_huella). Dependiendo del estado de la base de datos, las respuestas a estas consultas variarán. Estas respuestas sirven de *input* para la función que las compara y construye la respuesta, integrada por dos partes: la diferencia entre las respuestas de la base de datos serán los identificadores de las huellas pendientes de dibujar; y la intersección entre las respuestas de la base de datos serán los identificadores de las huellas cuyo dibujo ya está guardado. Entendemos dibujo como la descripción de las áreas de cobertura.

```
identificar petición como "GET /huellas/"
obtener todos los pares (posicion, huella) de la tabla Transpondedores
obtener todos los pares (posicion, huella) de la tabla Huellas
comparar las respuestas de las consultas de 2 y 3
si un elemento está en ambas
    acumular elemento en huellasGuardadas
si no
    acumular elemento en huellasPendientes
fin de si
responder (huellasPendientes, huellasGuardadas)
```

Figura 8

4.5.4. POST Huellas

La figura 9 indica qué hacer con una petición "POST /huellas/". El cuerpo de la petición incluye la identificación de la huella (posicion-nombre_huella) y las coordenadas que describen el perímetro de recepción para los tres niveles de potencia que estamos considerando. Tan sólo hay que guardar el registro en la tabla Huellas.

```
1  identificar petición como "POST /huellas/"
2  guardar en la BBDD las huellas recibidas en el cuerpo de la petición
```

Figura 9

4.5.5. POST Dibujo

La figura 10 muestra los pasos seguidos por una petición "POST /dibujo/". En el cuerpo de la petición encontramos el identificador (posicion-nombre_huella) de la huella de la cual se quiere conocer el área de cobertura (con sus tres niveles). A partir del identificador se consulta a la base de datos para obtener las coordenadas que describen el área de cobertura indicada y se responde al cliente con el identificador y las coordenadas que describen la huella.

```
1  identificar petición como "POST /dibujo/"
2  consultar BBDD con el binomio(posicion, nombre_huella) indicado
3  responder el resultado de la consulta
```

Figura 10

4.5.6. POST Borrar

La figura 11 representa el flujo seguido por una petición "POST /borrar/". El cuerpo de la petición contiene el identificador (posicion-nombre_huella) de la huella cuyo dibujo se quiere eliminar. Entendemos dibujo como las coordenadas que describen el perímetro del área de cobertura, con sus tres niveles de potencia ofrecida. Sólo hay que eliminar de la base de datos el registro indicado en la petición.

```
1  identificar petición como "POST /borrar/"
2  eliminar de la BBDD el registro indicado en la petición
```

Figura 11

5. Implementación

En este capítulo explicamos cómo se ha llevado a cabo la implementación de la aplicación. En concreto, describimos los recursos que hemos desarrollado y cómo la ejecución salta entre ellos a través de las llamadas a métodos. Asimismo, dedicamos una sección a recoger problemas destacables que hemos enfrentado y la solución encontrada.

5.1. Recursos

Los recursos desarrollados y almacenados en el servidor que sostiene la aplicación del lado del servidor son los que recoge la figura 12, donde también se puede observar la relación entre los mismos.

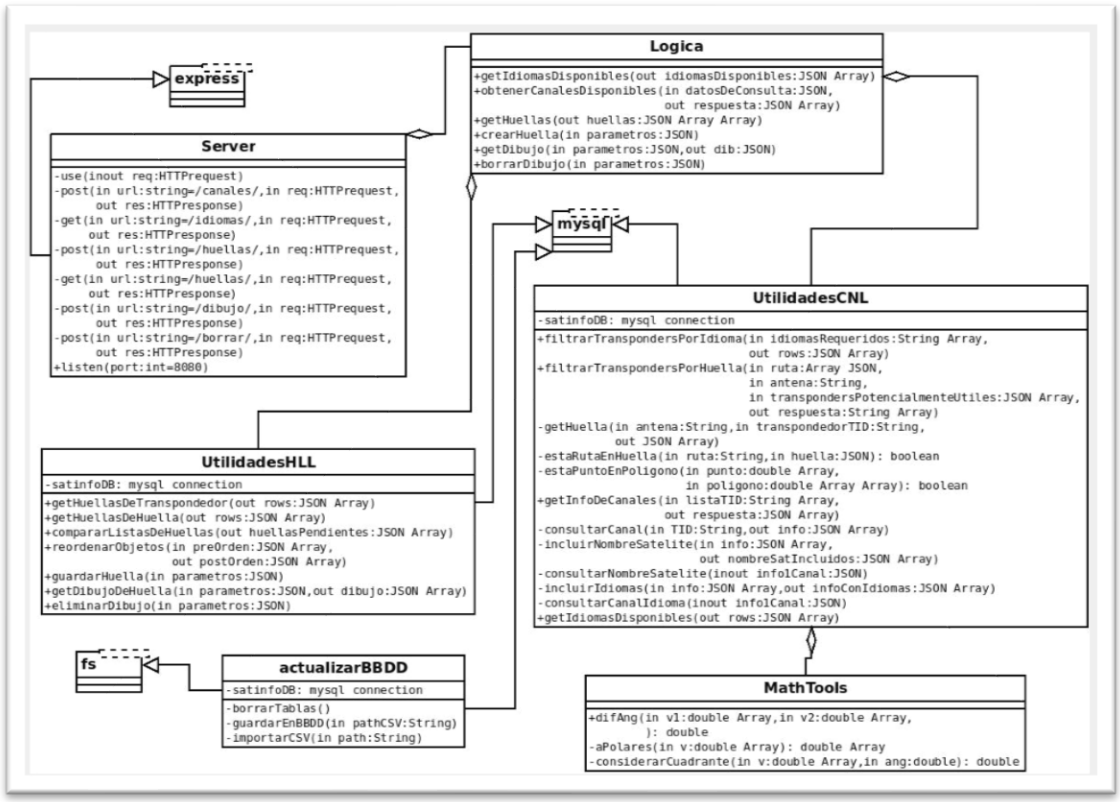

Figura 12

La clase Server hereda de la clase de biblioteca express las funciones básicas de servidor para escuchar un puerto y reconocer las peticiones. Tiene a la clase logica [sic] ya que llama a sus funciones para manejar las peticiones dependiendo de su naturaleza.

De la misma forma, la clase logica tiene UtilidadesHLL y UtilidadesCNL, las cuales heredan de la clase de biblioteca mysql para disponer con facilidad de funciones con conexión a la base de datos. UtilidadesCNL, además, tiene MathTools donde hemos implementado las funciones estrictamente matemáticas necesarias.

Por su parte, actualizarBBDD hereda de la clase de biblioteca fs que permite interacción con el sistema de archivos del sistema para poder leer el archivo CSV con la información de los satélites.

Por otro lado, la interfaz de usuario que implementa las funciones de cliente en nuestra arquitectura consiste en los archivos expuestos en la figura 13.

Los ficheros *html* son los que contienen las etiquetas que dan forma a la estructura de la web, mientras que los ficheros de tipo *js* la dotan de funcionalidad. Podemos ver la relación entre las funciones y cuál es la finalidad de cada una de ellas en detalle en el apartado 5.3.

No se ha desarrollado código *css* más allá de algunas descripciones en el atributo "style" de alguna etiqueta ya que esta interfaz pretende ser meramente funcional, sin sentido estético, el cual queda abierto a un diseño profesional en este sentido.

Figura 13

5.2. Llamadas en el servidor

Vamos a ilustrar visualmente la estructura del código del proyecto y, en particular, el recorrido que se lleva a cabo a través de las funciones implementadas en el servidor cuando éste recibe una petición (la información intercambiada y la finalidad de cada tipo de petición se encuentran recogidas en el apartado 6.2. Protocolo).

En la siguiente figura (Figura 14) elegimos el color identificativo de cada archivo, puesto que en adelante el texto contenido en los cuadrados corresponderá con las funciones que toman lugar.

Figura 14

5.2.1. GET Idiomas

El cliente envía la petición de forma automática al cargarse la página web que es la interfaz de usuario, y provoca en el servidor las llamadas ilustradas por la figura 15. Con la información que recibe como respuesta rellena los desplegables con las opciones de idioma disponibles.

Figura 15

5.2.2. POST Canales

Cuando el usuario ha descrito la ruta, elegido los idiomas que le interesan y el tipo de antena receptora de que dispone, pulsará en consultar y el cliente realiza la petición aquí tratada. El servidor ejecuta las funciones que vemos en la figura 16 para devolverle un objeto JSON con la información a desplegar en la tabla de la interfaz. filtrarTranspondersPorHuella ejecuta en bucle el bloque mostrado bajo la etiqueta "N veces" para comprobar huella a huella si cubre el área necesaria o no; estaRutaEnHuella, por su parte, ejecuta "i veces" el bucle a partir de estaPuntoEnPoligono, comprobando para cada punto de la ruta si está dentro del área de cobertura (sólo si todos los puntos lo están, estaRutaEnHuella devolverá "true").

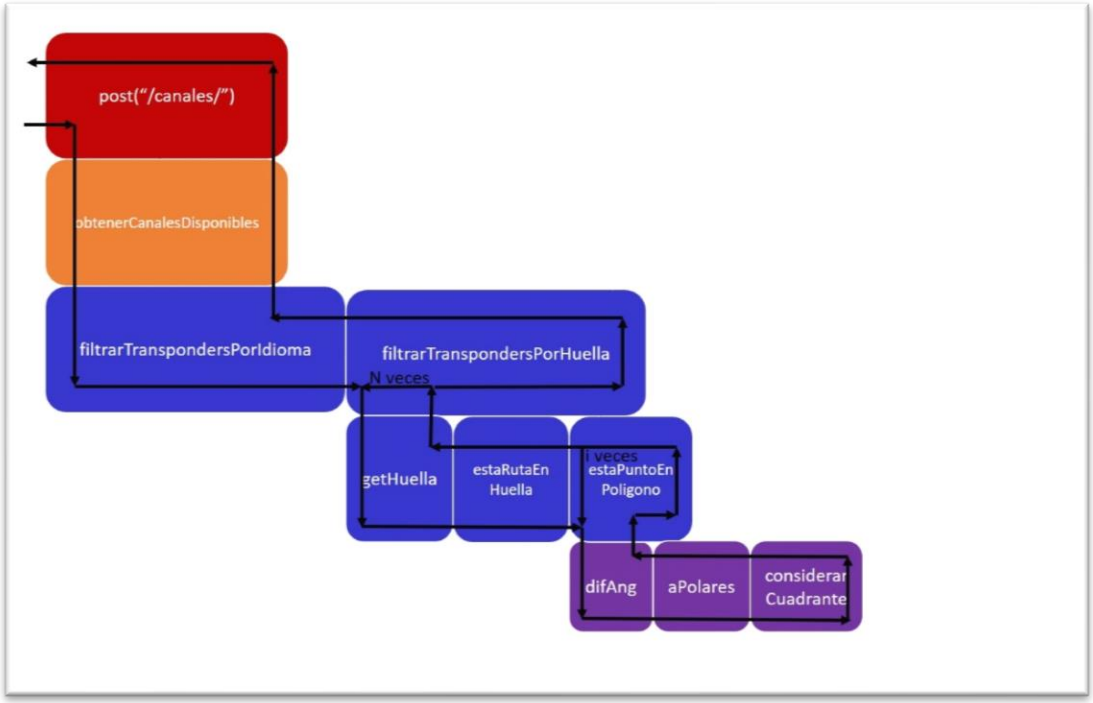

Figura 16

5.2.3. GET Huellas

Esta petición se envía automáticamente al cargar la web para introducir los dibujos correspondientes a las áreas de cobertura de los transpondedores. Las respuestas de getHuellasDeTranspondedor y de getHuellasDeHuella pasan por la función reordenarObjetos, como muestra la figura 17, que las acomoda de mejor forma para juntarlas manteniendo la diferenciación. Así, el cliente recibe en una sola respuesta los nombres de las huellas que deben ser dibujadas y las que ya lo están, pudiendo distinguir entre ellas para ser dispuestas en sendos desplegables de la interfaz.

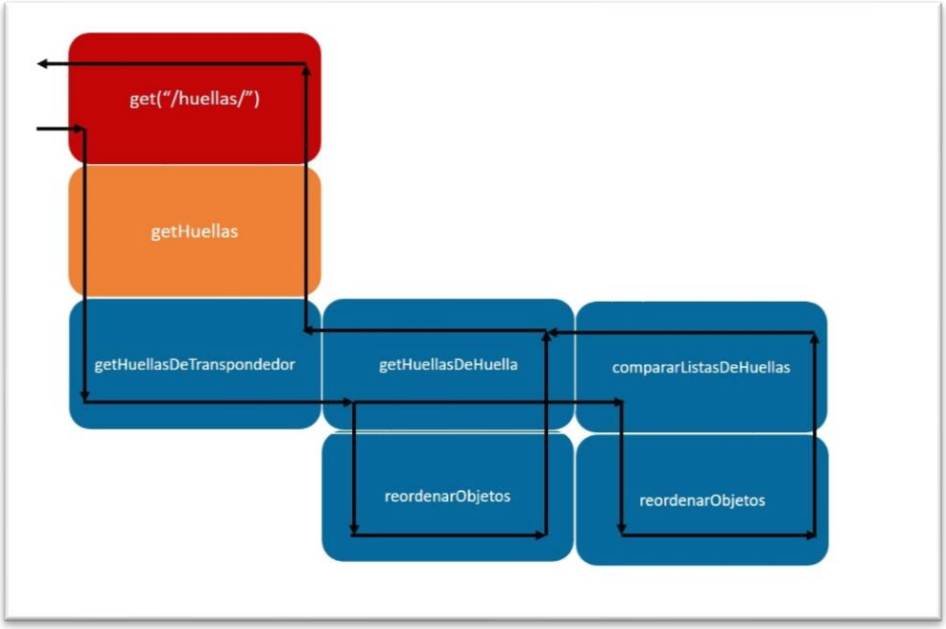

Figura 17

5.2.4. POST Huellas

Cuando el usuario introduce los puntos que describen el área de cobertura de una huella, pulsa en el botón "Guardar" de la interfaz y se envía esta petición al servidor. Como ilustra la figura 18, la información del cuerpo de la petición se traslada hasta la función guardarHuella, que creará un nuevo registro en la base de datos con ella.

Figura 18

5.2.5. POST Dibujo

Esta cadena de ejecuciones mostrada en la figura 19, la provoca la petición que envía el cliente al servidor cuando el usuario pulsa el botón "Visualizar". getDibujoDeHuella consulta la base de datos con los parámetros recibidos y devuelve un objeto con las coordenadas de los vértices del área de cobertura.

Figura 19

5.2.6. POST Borrar

Cuando el usuario pulsa el botón "Borrar", envía al servidor la petición "POST /borrar/", en el cuerpo de la cual se incluye el identificador de la huella que se desea borrar. Esta información se traslada hasta eliminarDibujo, como vemos en la figura 20, que construye y ejecuta la correspondiente sentencia SQL.

Figura 20

5.3. JavaScript en la interfaz de usuario

La interfaz de usuario consiste en dos páginas web. La funcionalidad de cada elemento y la apariencia de éstas se explica en el apartado 6.3., de modo que aquí veremos cómo funcionan desde el "segundo plano": sus funciones JavaScript.

5.3.1. Consulta de canales disponibles

- "`getIdiomas()`". Ejecución por "onload" de la etiqueta "body". Lanza al servidor la petición "GET /idiomas/", a través de la cual recibe los idiomas disponibles entre todos los canales. Recibida la respuesta, la pasa a la función "`cargarDesplegables(idiomasDisponibles)`" en la variable "idiomasDisponibles".
- "`cargarDesplegables(idiomasDisponibles)`". Ejecución por llamada desde "`getIdiomas()`". Genera el código *html* con el que llena los elementos de tipo *select* de idiomas disponibles, con las opciones que indicó el servidor.
- "`enviarConsulta()`". Ejecución por "onclick" del botón "Consultar". Comprueba que sólo haya una ruta dibujada en el mapa para evitar ambigüedades y, en tal caso, obtiene los valores de los elementos que toman información de parte del usuario: los puntos marcados sobre el mapa para dibujar la ruta, el valor de los desplegables de idioma y el valor del desplegable correspondiente al tipo de antena. Con esta información, envía una petición "POST /canales/" al servidor, cuya respuesta pasa como parámetros a la función "`crearTabla()`".
- "`crearTabla()`". Ejecución por llamada desde "`enviarConsulta()`". Genera el código *html* necesario para construir una tabla en la que mostrar la respuesta del servidor (incluyendo el botón para excluir canales incompatibles).
- "`excluirCanalesIncompatibles(id)`". Ejecución por "onclick" del botón de selección en la primera columna de la tabla de canales. Recibe en "id" una identificación de qué botón lanzó su ejecución. A través de esta variable, toma el valor de las celdas clave para la compatibilidad de los canales y las convierte en patrón para el filtro de la tabla (llamando a la función de biblioteca "`SetFilterValue()`"), quedando así visibles sólo los canales que sean compatibles con la selección del usuario.

5.3.2. Gestión de huellas de cobertura

- "`getHuellas()`". Ejecución por "onload" de la etiqueta "body". Envía al servidor la petición "GET /huellas/", de la que recibe como respuesta las parejas posición-nombre_huella de las huellas que hay guardadas y de las que no tenemos su área de cobertura aún. Llama a "`actualizarDesplegableHllPendientes()`" y a "`actualizarDesplegableHllGuardadas()`", trasladando en la llamada como parámetro la parte respectiva de la respuesta del servidor.
- "`actualizarDesplegableHllPendientes()`". Ejecución por llamada desde "`getHuellas()`". Llena el desplegable para la posición de los transpondedores cuyas huellas no han sido dibujadas con la información que le llega como parámetro. Llama a "`actualizarCheckboxes()`".
- "`actualizarCheckboxes()`". Ejecución por llamada desde "`actualizarDesplegableHllPendientes()`" o por "onchange" del desplegable de posiciones de huellas pendientes. Cambia las opciones disponibles para marcar en la lista de nombres de huellas pendientes de ser dibujadas.
- "`actualizarDesplegableHllGuardadas()`". Ejecución por llamada desde "`getHuellas()`". Llena el desplegable para la posición de los transpondedores cuyas huellas ya han sido dibujadas con la información que le llega como parámetro. Llama a "`actualizarNombresGuardadas()`".
- "`actualizarNombresGuardadas()`". Ejecución por llamada desde "`actualizarDesplegableHllGuardadas()`" o por "onchange" del desplegable de posiciones de huellas guardadas. Llena el desplegable para los nombres de las huellas que ya han sido dibujadas y están disponibles para visualizar o borrar.
- "`anadirLinea()`". Ejecución por "onclick" del botón "OK". Relaciona la última línea dibujada por el usuario con el nivel de ganancia seleccionado.
- "`visualizar()`". Ejecución por "onclick" del botón "Visualizar". Toma los valores de los desplegables que identifican la huella guardada de la que el usuario quiere ver el área de cobertura y los envía al servidor a través de una petición "POST /dibujo/". Cuando recibe la respuesta la introduce en los parámetros de la función de biblioteca del mapa que permite dibujar polígonos sobre éste.
- "`borrar()`". Ejecución por "onclick" del botón "Borrar". Toma los valores de los desplegables que identifican la huella guardada que el usuario quiere borrar y los envía al servidor a través de una petición "POST /borrar/".
- "`mostrarMapa()`" se ejecuta al tiempo que se carga la página. Únicamente llama a la función de biblioteca que permite mostrar un mapa en la página web.
- "`guardarHuella()`". Ejecución por "onclick" del botón "Guardar". Recoge la información introducida por el usuario: huella dibujada (con sus tres niveles distintos), posición del transpondedor y nombre de la huella. Una vez recogida, comprueba que no haya errores de utilización por parte del usuario a través de "`comprobaciones()`", que devuelve un boleano "true" si está todo correcto. En caso afirmativo, traslada esta información al servidor por medio de una petición "POST /huellas/".

5.4. Problemas surgidos y soluciones implementadas

El principal problema al que nos hemos enfrentado ha sido controlar el flujo de programa a través de las llamadas asíncronas dentro de bucles. Para ello, hemos implementado una forma de iteración que llamaremos pseudo-bucle, pero cuyo funcionamiento es exactamente el mismo que en un bucle *for*, establecida una condición inicial: comprobación, ejecución, actualización. En la figura 21, podemos ver la estructura ejemplificada de este código.

```
function funcionDeEjemplo( param0, callback){
        var respuesta = [];

        var max = param0.length - 1;
        var cnt = 0; // variable que controla las iteraciones

        funcionQueContieneElCodigoRepetitivo( param0[cnt], callbackDePseudobucle = function(param1, param2){

                /* La siguiente línea es un ejemplo, sería cualquier código que afecte a las variables
                 * de funcionDeEjemplo a partir del resultado de funcionQueContieneElCodigoRepetitivo
                 */
                respuesta.push(param1 + param2);

                if( cnt === max){
                        callback( respuesta );
                }else{
                        cnt++;
                        funcionQueContieneElCodigoRepetitivo( param0[cnt], callbackDePseudobucle );
                }
        });
}
```

Figura 21

Por otra parte, cabe resaltar la necesidad de una configuración apropiada sobre las cabeceras CORS (Cross-Origin Resource Sharing) si se ofrece acceso a la aplicación a través de Internet. Se entrega con una configuración en la que están permitidas las peticiones a cualquier servidor diferente del que provee la página, ya que al correr de forma local no existen riesgos de seguridad a este respecto.

6. Guía de uso

Este capítulo contiene la guía de uso, tanto para instalación como para el empleo de la herramienta implementada. En la guía de instalación propondremos una configuración como estándar, paso a paso, de modo que pueda estar sujeta a cambios introducidos por un técnico con conocimiento avanzado, pero también pueda ser seguida por personal menos experimentado. Por su parte, el apartado “Uso”, pretende ser un manual básico.

6.2. Instalación

Antes de instalar la aplicación, es necesario instalar (y configurar) en el sistema algunas dependencias. Vamos a explicar los pasos a seguir para poner en funcionamiento la aplicación sobre un sistema Ubuntu, desde cero.

6.2.1. Directorio de instalación y dependencias

En primer lugar, deberá decidirse el directorio de instalación. Para una instalación de uso local recomendamos `/home/<user>/`, pero esta ubicación puede cambiarse por cualquier otra que convenga mejor al administrador.

1. Crear la carpeta de instalación en el directorio seleccionado [`mkdir /home/<user>/satselector/`].
2. Copiar los archivos de la aplicación a la nueva ubicación [`cp /path/donde/tengamos/los/archivos/de/la/aplicacion/* /home/<user>/satselector/`].
3. Instalar mariaDB [`sudo apt-get install mariadb-server`]. Nos pedirá confirmación antes de proceder a la instalación. Después, introduciremos una contraseña con la que protegeremos los permisos de *root* dentro de mariaDB.
4. Instalar node.js [`sudo apt-get install nodejs-legacy`]. Nos pedirá confirmación antes de proceder a la instalación.

6.2.2. Configuración de la base de datos

1. Para dotar de unos mínimos de seguridad a nuestra base de datos realizamos una configuración rápida escribiendo en el terminal [`mysql_secure_install`]. Este comando eliminará usuarios anónimos de la BBDD, evitará la conexión remota, y eliminará la base de datos de prueba predeterminada. En cada paso nos permite elegir si queremos proceder con esa línea o no.
2. Abrimos la aplicación mariaDB con autoridad de root [`mysql -u root -p`]. Con la opción "-u" indicamos el usuario y "-p" provoca que nos pida la contraseña en la línea siguiente, aunque también puede indicarse a continuación de la opción (no recomendado).
3. Ahora nos deberíamos encontrar en la consola de mariaDB viendo algo como lo que ilustra la figura 22. Creamos la base de datos sobre la que trabajará nuestra aplicación [`CREATE DATABASE SatInfo`].

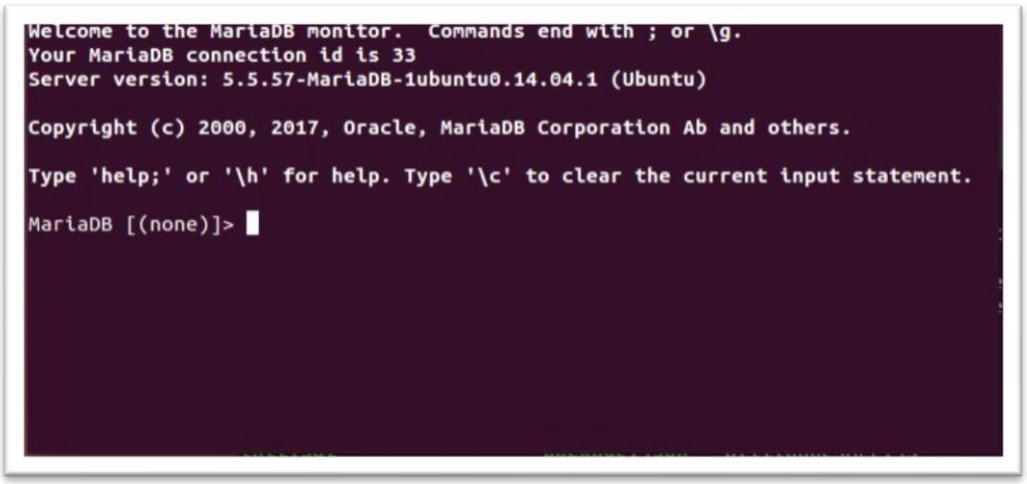

Figura 22

4. Elegimos la base de datos sobre la que se ejecuten las siguientes sentencias [`USE SatInfo`]. Ahora debería aparecer "SatInfo" entre los corchetes de la consola.
5. Entre los archivos que hemos copiado en el apartado 8.1.1. se encuentra uno llamado "init.sql", que construye la estructura de la BBDD. Téngase en cuenta si se eligió un directorio de instalación diferente al propuesto, en nuestro ejemplo debemos introducir en la consola de mariaDB [`\. /home/<user>/satselector/configurator_satselector/SatInfo/init.sql`].
6. Vamos a crear los usuarios de la BBDD, que serán en realidad las clases que tendrán acceso a la misma. En los archivos "actualizarBBDD.js", "UtilidadesCNL.js" y "UtilidadesHLL.js", al comienzo, hay una variable JSON con los parámetros de acceso a la BBDD. Es importante que coincidan con los que vamos a introducir en este punto. Para el ejemplo, utilizaremos los valores predeterminados y con acceso local.
   [`CREATE USER 'actualizador'@'localhost' IDENTIFIED BY 'contrasena';`]
   [`CREATE USER 'consultor'@'localhost' IDENTIFIED BY 'contrasena';`]
7. Ya solo falta dotar de permisos a los usuarios recién creados. Los necesarios para el correcto funcionamiento de la aplicación son los que otorgan los siguientes comandos.
   [`GRANT INSERT, DELETE ON SatInfo.CanalIdioma TO 'actualizador'@'localhost';`]
   [`GRANT INSERT, DELETE ON SatInfo.Canal TO 'actualizador'@'localhost';`]
   [`GRANT INSERT, DELETE ON SatInfo.Transpondedor TO 'actualizador'@'localhost';`]
   [`GRANT SELECT ON SatInfo.CanalIdioma TO 'consultor'@'localhost';`]
   [`GRANT SELECT, INSERT, DELETE ON SatInfo.Huella TO 'consultor'@'localhost';`]
   [`GRANT SELECT, INSERT, DELETE ON SatInfo.Transpondedor TO 'consultor'@'localhost';`]
   [`GRANT SELECT ON SatInfo.Canal TO 'consultor'@'localhost';`]

6.2.3. Instalar la aplicación

Para empezar el funcionamiento de la aplicación, lo primero que hay que hacer es llenar la base de datos con la información de los satélites. Para eso, debemos ubicar en `/home/<user>/satselector/configurator_satselector/SatInfo/archivo` el fichero CSV que nos envía nuestro proveedor de información, y a continuación ejecutar el *script* "actualizarBBDD.js" con node para trasladar la información a la base de datos: [`node /home/<user>/satselector/configurator_satselector/SatInfo/actualizarBBDD.js`]. Este script está implementado para el formato de datos que nos proporciona SatBeams, si cambia el proveedor habrá que realizar los cambios pertinentes para el correcto volcado de información.

Una vez tenemos la base de datos cargada, sólo tenemos que ejecutar el *script* "server.js" para que el lado de servidor esté operativo:
[`node /home/<user>/satselector/configurator_satselector/SatInfo/server.js`]. En ese momento la consola deberá tener un aspecto como el que muestra la figura 23.

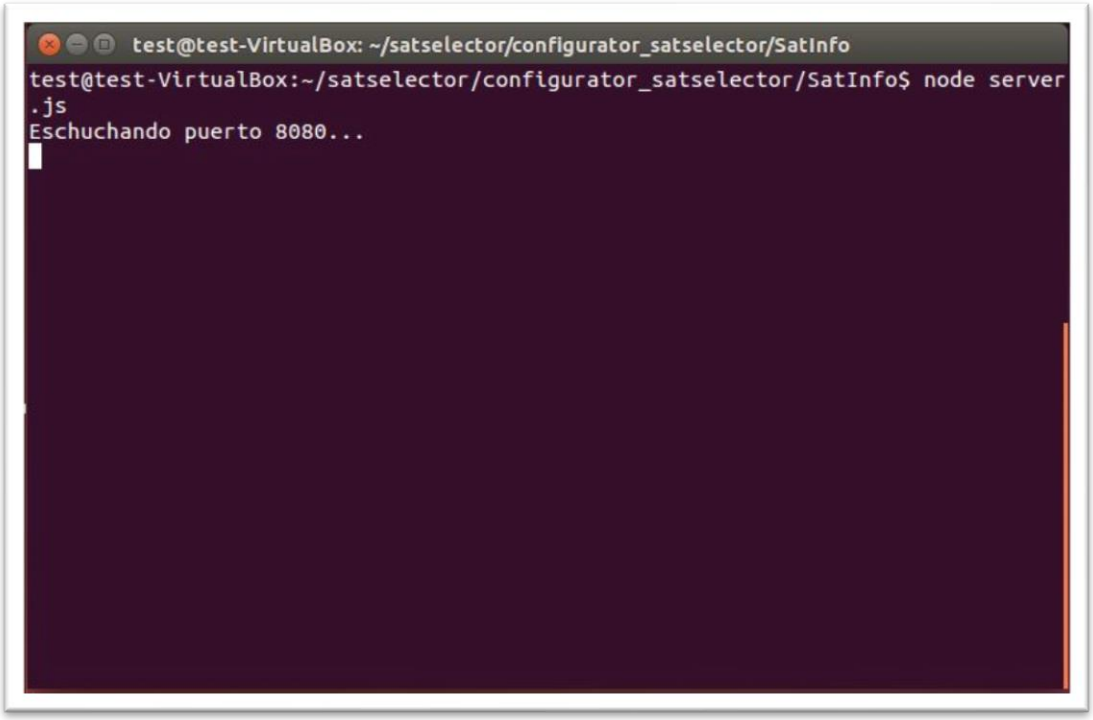

Figura 23

6.3. *Uso*

El usuario final sólo se va a enfrentar a dos interfaces: “consulta de canales disponibles” y “gestión de huellas de cobertura”. Siguiendo la ruta a través del sistema de archivos `/home/<user>/satselector/configurator_satselector/WebClient/public_html/` podemos acceder a sendos documentos *html* que podremos utilizar abriendo con un navegador.

6.3.1. Consulta de canales disponibles

La figura 24 nos muestra la apariencia de esta interfaz, con algunos números superpuestos para facilitar la explicación.

Figura 24

Los cuatro desplegables identificados con el número 1 corresponden a los cuatro idiomas que se pueden seleccionar para filtrar los canales interesantes al usuario. Las opciones que se ofrecen son las que se encuentran disponibles en los transpondedores de los que se tiene información.

El desplegable identificado con el número 2 nos permite seleccionar, entre tres modelos, la antena que se empleará para la recepción, de forma que el servidor sepa la huella de qué nivel de potencia tiene que considerar.

Justo debajo de la etiqueta con el número 3 tenemos un botón con el que el cursor se convierte en herramienta de dibujo sobre el mapa. Tal como indican las instrucciones en la parte inferior, con un clic se marcan los cambios de dirección para dibujar trazos que se ajusten a la ruta del autobús; y un segundo clic sobre el último punto libera el cursor.

El botón “Consultar” identificado con un 4 envía la petición al servidor, cuya respuesta se mostrará en una tabla debajo. El botón “Refresh” recarga la página de modo que limpia los dibujos que haya en el mapa.

Haciendo clic sobre el botón identificado con el número 6 pasamos a “gestión de huellas de cobertura”.

La tabla que muestra la respuesta del servidor tiene el aspecto que podemos ver en la figura 25. Sobre esta imagen hemos superpuesto etiquetas numeradas que explicamos a continuación.

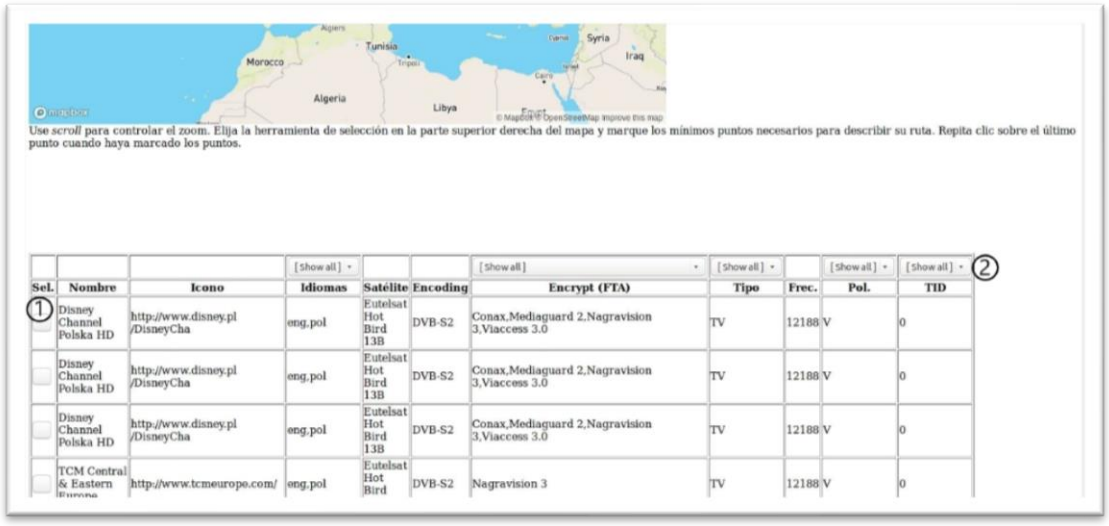

Figura 25

Con el número 1 etiquetamos el botón de selección, que permite al usuario elegir el canal más interesante para él, modificando automáticamente los filtros (etiquetados por el número 2) para que se ajusten a la configuración de ese canal. Así, ese canal se convierte en patrón y la lista se ve reducida a aquellos que sean compatibles con su configuración. Por supuesto, el manejo de los filtros también puede hacerse manualmente, clicando sobre los desplegables que encabezan las columnas se ofrecen los distintos valores que existen en la tabla para proceder al filtrado conforme a alguno de ellos.

6.3.2. Gestión de huellas de cobertura

A continuación, encontramos una imagen de esta interfaz con etiquetas numeradas para explicar la función de cada elemento (figura 26).

Figura 26

Con los desplegables identificados con los números 1 y 2, seleccionamos, entre aquellas cuyo dibujo ya tenemos guardado, la huella que queremos visualizar en el mapa por su posición y nombre; al seleccionar la posición con el desplegable 1, las opciones del desplegable 2 cambian para mostrar únicamente los nombres disponibles en dicha posición. Clicando en 3 se hace efectiva la selección y se mostrará el dibujo con los perímetros de las áreas con distinto nivel de potencia recibida. Clicando en 4 eliminaremos de la base de datos la huella que haya seleccionada.

La zona identificada con un 5 ofrece tres opciones de selección excluyente: sólo puede estar marcada una de ellas, e indica el nivel de potencia al que corresponde el dibujo que se haga. Clicando en 6 el cursor se convierte en la herramienta de dibujo en la zona del mapa, con la cual marcamos sobre el mapa los vértices que definen el área de cobertura; clicando de nuevo sobre el último punto marcado liberaremos el cursor. Cada vez que se dibuje un polígono en el mapa, se debe clicar en 7 para hacer efectiva la relación de éste con el nivel de potencia determinado por 5.

Las etiquetas 8 y 9 son análogas a 1 y 2 pero, en este caso, tratando las huellas de las cuales el dibujo todavía no ha sido introducido a la base de datos.

Una vez dibujadas las tres zonas de cobertura e identificada la huella a la que pertenecen, un clic en 10 (“Guardar”), trasladará la información al servidor para que la incluya en la base de datos. Por su parte, el botón “Refresh” identificado con un 11 sirve para limpiar el mapa recargando la página.

7. Conclusiones y trabajo futuro

Este proyecto ha sido desarrollado a petición de Azimut Electronics, para facilitar la selección de paquetes comerciales de canales de televisión por satélite por parte de sus clientes de 'Azimut On Road': la filial de sistemas de entretenimiento para medios de transporte.

Como respuesta a la necesidad presentada, hemos implementado una aplicación cliente-servidor, a través de la cual podemos listar los canales de televisión satelital disponibles para una determinada ruta de viaje, uno de los equipos de recepción que instala Azimut y hasta cuatro idiomas de interés; todo ello seleccionable por el usuario. Se convierte así una tarea tediosa y mecánica en un paso rápido a través de su informatización.

A lo largo de este documento, hemos repasado el proceso de desarrollo a través de la explicación de las fases del mismo. Vimos las necesidades indicadas por Azimut Electronics desde un punto de vista técnico, el diseño de nuestra aplicación en abstracto, las herramientas informáticas elegidas para la implementación y la implementación en sí misma. También se ha incluido un breve manual de consulta (la Guía de uso) como documentación para facilitar la tarea del instalador y el usuario de la aplicación.

Trabajo futuro

Tal como indicábamos en el análisis de requerimientos, el proyecto queda abierto a una ampliación que incluya plantillas para los clientes habituales de Azimut Electronics, que guarden sus rutas e idiomas cotidianos.

Asimismo, queremos dejar constancia de otra propuesta de ampliación, consistente en la recogida de la información de emisión por parte de los satélites. Esta labor actualmente es llevada a cabo por un intermediario (el proveedor del archivo CSV que alimenta la base de datos) que cobra una cuota mensual por concentrar la información que las empresas que mantienen los repetidores en órbita difunden.

Bibliografía

STEFAN TILKOV. (2007).
A Brief Introduction to REST.
<https://www.infoq.com/articles/rest-introduction>

MARIADB CORPORATION AB. (2017).
MariaDB: Documentation.
<https://mariadb.com/kb/en/mariadb/documentation/>

STRONGLOOP (2017).
Express documentation.
<http://expressjs.com/4x/api.html>

THESYS MANAGEMENT CONSULTING, S.L. (2017).
Controlar la ejecución asíncrona
<https://www.todojs.com/controlar-la-ejecucion-asincrona/>

MOZILLA AND INDIVIDUAL CONTRIBUTORS. (2005-2018).
XMLHttpRequest documentation
<https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>

MOZILLA AND INDIVIDUAL CONTRIBUTORS. (2005-2018).
Control de acceso HTTP (CORS)
<https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS>

MAPBOX, INC. (2017).
Mapbox GL JS documentation
<https://www.mapbox.com/mapbox-gl-js/api/>

MAX GUGLIELMI. (2017).
TableFilter documentation
<http://tablefilter.free.fr/doc.php>