Grado en Ingeniería Informática

Trabajo Final de Grado

Back-end para Visor Web DICOM de imagen radiológica

Autor:
Imanol Fraile de la Cruz

Supervisor:
Rafael Forcada Martínez
Tutor académico:
Rafael Berlanga Llavori

Fecha de lectura: 10 de Julio de 2014
Curso académico 2013/2014

Resumen

El análisis de imágenes radiológicas presenta un papel fundamental para el diagnóstico y prevención de enfermedades graves. DICOM (*Digital Imaging and COmmunication in Medicine*) [1] es en estos momentos el estándar de imagen médica digital más utilizado en todo el mundo.

Actualmente existen diversos visores de imágenes DICOM para entornos de escritorio. Sin embargo, no ocurre lo mismo en el entorno web.

Por lo tanto, es este el motivo de peso para desarrollar un sistema que permita a los médicos radiólogos observar las radiografías de sus pacientes de una manera rápida, precisa e intuitiva en sus ordenadores y dispositivos multimedia, tales como tabletas o teléfonos móviles. Esto será posible gracias al desarrollo completo de un visor web que permita visualizar imágenes en formato DICOM. De esta forma, se reduce el tiempo de respuesta de los médicos a la hora de otorgar a sus pacientes un diagnóstico cuando no se encuentren en su puesto habitual de trabajo y no dispongan del tradicional visor de escritorio.

Palabras clave

DICOM: estándar de imagen médica digital

WADO: servidor web de consulta y acceso a imágenes DICOM

PHP: lenguaje de desarrollo en el lado del servidor

MYSQL: sistema de gestión de bases de datos

MVC: patrón de diseño típico en aplicaciones web

Keywords

DICOM: digital medical image standard

WADO: web server for query and access to DICOM images

PHP: server side development language

MYSQL: database management system

MVC: typical design pattern in web applications

Índice general

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

Hoy en día, el imparable aumento de potencia y funcionalidades de los dispositivos multimedia como los teléfonos, tabletas o televisores inteligentes está provocando que no podamos concebir la existencia de una aplicación de escritorio sin su correspondiente versión web. Si a esto le añadimos la movilidad y escalabilidad que nos ofrece la computación en la nube, estamos haciendo referencia a aplicaciones web que se pueden ejecutar en cualquier dispositivo con una conexión a Internet.

Existen ya varios visores de imágenes radiológicas que se descargan, se instalan y se ejecutan con unos resultados asombrosos. Sin embargo, la motivación de este proyecto radica en poder ofrecer las funcionalidades más esenciales de los visores de escritorio en un entorno web. Este entorno será, por lo tanto, accesible desde cualquier navegador web, aportando así una movilidad muy elevada para usuarios que requieran desplazamientos y necesiten realizar diagnósticos en el menor tiempo posible. Estamos hablando efectivamente de técnicos radiólogos, profesionales médicos encargados de manipular imágenes que utilizan para diagnosticar enfermedades.

También es objeto de motivación que el proyecto sea desarrollado completamente sin la colaboración de terceras empresas. De este modo, se gana independencia frente a la inclusión de futuras funcionalidades, aunque también se requiere un trabajo más duro y meticuloso, debiendo controlar cualquier aspecto y dificultad que se presente en el camino sin más ayuda que la del propio equipo de trabajo e Internet.

El proyecto ha sido desarrollado en la empresa ActualTec Innovación Tecnológica, S.L. (Actualmed), ubicada en el parque científico, tecnológico y empresarial (Espaitec) de la Universidad Jaime I de Castellón. El objetivo de esta empresa es el de proporcionar sistemas radiológicos de última generación. Su

equipo humano ofrece soluciones óptimas utilizando tecnología basada en software libre y hardware Apple para obtener resultados asombrosos.

1.2. Objetivos del proyecto

El desarrollo de un visor web para el tratamiento de imágenes radiológicas requiere, como todo proyecto informático con interfaz de usuario, de la implementación de código estructurado en dos apartados diferentes pero perfectamente sincronizados.

El primero de ellos se centra en la parte visual, es decir, en el desarrollo del código que compone el visor web encargado de mostrar las imágenes y ofrecer la interacción con el usuario final. A este apartado se le denomina parte frontal o *front-end*.

El segundo se trata de la parte oculta, la zona trasera, el código del servidor, conocido como *back-end* de la aplicación. Su propósito es el de escuchar y recibir los datos de las peticiones que le envía el *front-end* para devolverle los resultados que desee.

Al inicio de este proyecto, el desarrollo del *front-end* se encontraba en un estado bastante avanzado. De su elaboración se ha encargado por completo un programador especializado con el que se ha mantenido comunicación durante prácticamente la totalidad del tiempo dedicado a este proyecto.

Por lo tanto, el objetivo de este proyecto consta del desarrollo completo del servidor o *back-end* capaz de hacer frente a las peticiones del *front-end*.

Además de la implementación del propio código, se requiere emplear gran parte del espacio temporal del proyecto en la realización de pruebas. Las fases de prueba deben resultar muy productivas, ya que el objetivo principal de las mismas es la localización de errores importantes tanto en el código del servidor como en el del visor web.

Capítulo 2

Descripción del proyecto

2.1. DICOM, PACS y el desarrollo del proyecto

DICOM (*Digital Imaging and COmmunications in Medicine*) representa años de esfuerzo de creación del estándar fundamental sobre imagen médica digital más universal del momento. Al contrario de lo que popularmente se dice, DICOM es mucho más que un simple formato de archivo con extensión ".dcm" que contiene una imagen médica y sus propios metadatos. DICOM abarca protocolos de transferencia de datos, almacenamiento y visualización de imágenes especialmente diseñados para cubrir todos los aspectos funcionales de la imagen médica digital.

PACS (*Picture Archiving and Communication Systems*) es otro concepto directamente relacionado con DICOM que se debe explicar. Los PACS son sistemas médicos (formados por hardware y software) diseñados y utilizados para procesar imágenes médicas digitales. Están formados por tres componentes principales:

- Dispositivos para la adquisición de imágenes digitales (por ejemplo, una máquina de rayos X).
- Discos duros para el almacenamiento de las imágenes digitales adquiridas (por ejemplo, servidores alojados en la nube).
- Estaciones de trabajo donde los radiólogos visualizan las imágenes almacenadas (por ejemplo, un ordenador o una tableta).

La estrecha relación entre PACS y DICOM garantiza una alta interoperabilidad. Por esta razón, cualquier hardware o software PACS viene con su propia *Declaración de Conformidad DICOM*. Esta *Declaración* es un documento muy importante que explica hasta qué punto es soportado el estándar DICOM por el dispositivo en cuestión.

El proyecto se centra en el desarrollo de un servidor capaz de responder a las peticiones de un visor web encargado de presentar imágenes DICOM en pantalla para los médicos radiólogos. Las tareas principales del servidor son el acceso a una base de datos de estudios de pacientes, la descarga de imágenes DICOM con sus

respectivos metadatos y la manipulación de estas imágenes para conseguir una comunicación con el visor web lo más rápida posible.

Para alcanzar una buena rapidez es primordial no repetir acciones innecesarias. Parte de la complejidad reside en descubrir qué acciones son las que se pueden omitir para conseguir mayor velocidad.

Además, es completamente necesaria una caché de resultados si no queremos que el visor web vuelva a descargar imágenes que se visitan con mucha frecuencia. Sin embargo, mantener muchas imágenes en caché puede terminar por consumir el espacio del propio servidor, así que se debe buscar un equilibrio para optimizar la caché. No obstante, si el software desarrollado se encuentra replicado en múltiples servidores, el tamaño de la caché no será un problema grave, pudiendo así mantener en ella imágenes que ya no se visitan con tanta frecuencia.

2.2. Metodología y entorno de trabajo

La metodología seguida para la implementación del código del servidor está compuesta por tres fases bien diferenciadas.

La primera consiste en la contextualización del problema, pues sin saber qué resultado final se quiere obtener, difícilmente se podrán alcanzar los objetivos propuestos.

La segunda fase contiene cualquier actividad relacionada con la búsqueda de información. Internet [2] ha sido la fuente principal (*Figura 1*).

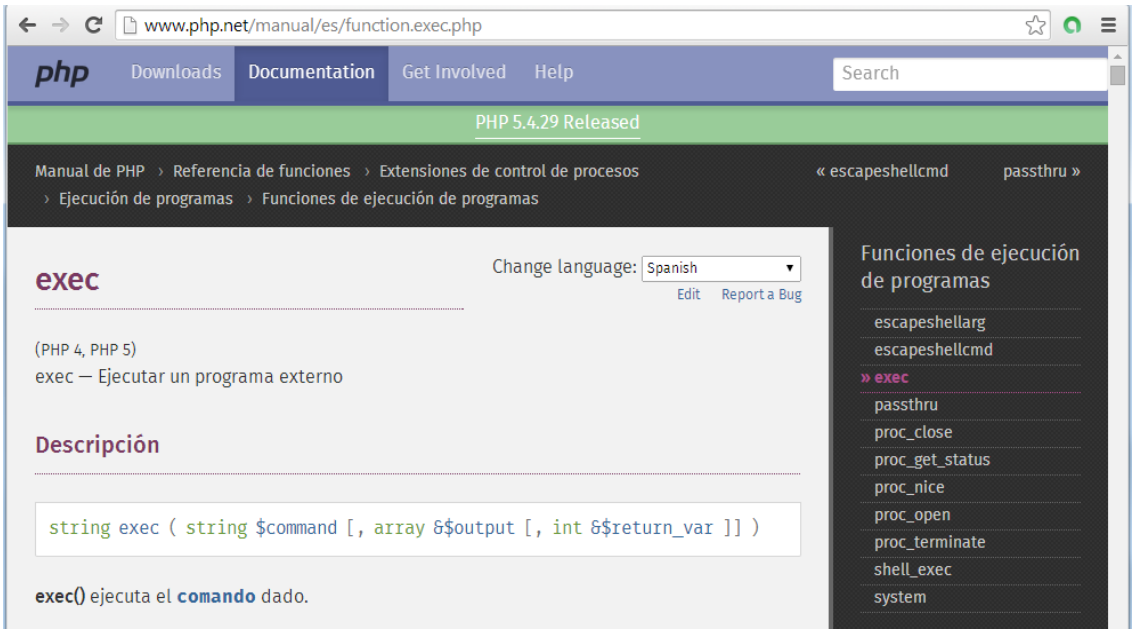

Figura 1.- Fragmento del manual web sobre la función *exec* de PHP [3].

Sin embargo, la ayuda recibida por los otros desarrolladores de este proyecto ha resultado muy importante para avanzar en los momentos críticos. Uno de los desarrolladores ha residido en el mismo espacio de trabajo que yo y el otro en Barcelona, con el que he intercambiado innumerables correos electrónicos durante las pruebas finales. El primero supervisaba mis avances con el *back-end* y el segundo ha tomado el desarrollo completo de la parte del *front-end*.

Una vez recopilada la información necesaria, la tercera fase consiste en plasmarla en el código. Generalmente se comprueba que los ejemplos proporcionados por los manuales funcionen correctamente y a continuación se adaptan a las necesidades del proyecto (*Figura 2*).

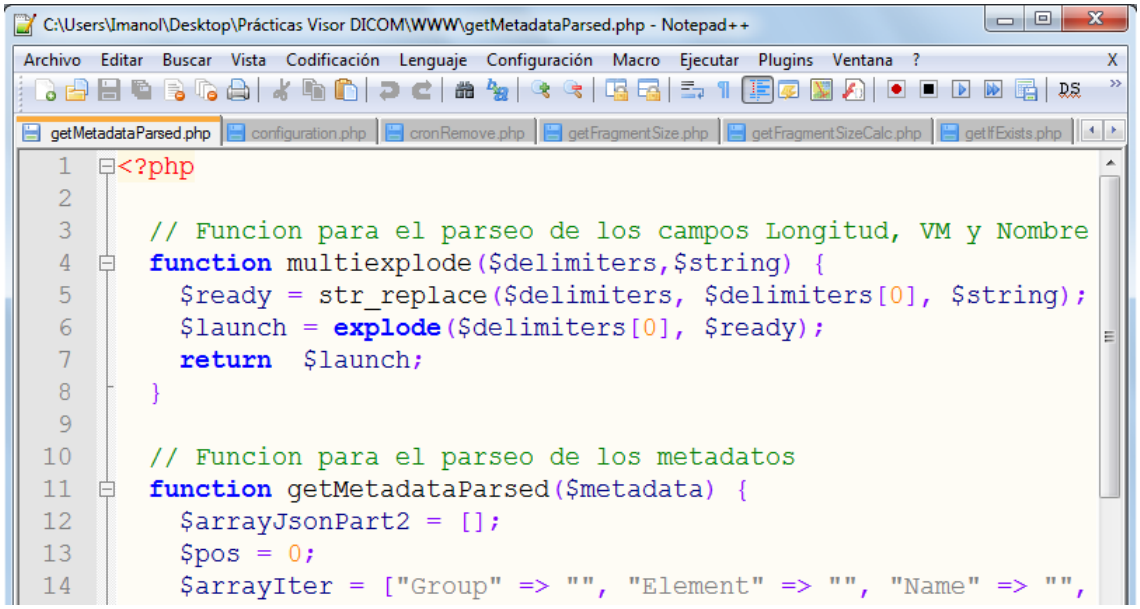

Figura 2.- Ejemplo de código PHP preparado para la fase de pruebas.

El patrón de diseño por excelencia en el desarrollo web es el conocido como MVC (Modelo, Vista, Controlador). De la Vista se encarga el visor web, mientras que del Modelo y del Controlador se encarga el *back-end*. Cuando el usuario final realiza acciones como abrir el estudio de un paciente o realizar zoom sobre una imagen, el Controlador se encarga de escuchar estas peticiones y avisar al Modelo para que acceda a la base de datos y descargue, fragmente o muestre las imágenes en función de la acción solicitada, devolviendo finalmente los datos útiles al Controlador y éste a la Vista.

Finalmente, la cuarta fase está compuesta por las pruebas definitivas, ya sean en modo local en los inicios del proyecto o con el servidor dedicado de demostración en las fases intermedias y avanzadas.

El entorno de trabajo personal es Windows, por lo que se ha utilizado el programa Filezilla para el intercambio de ficheros entre mi ordenador personal y

el servidor dedicado. El protocolo utilizado es SFTP (*Secure File Transfer Protocol*) para una transferencia de ficheros segura entre ambas partes.

En la *Figura 3* podemos observar a gran escala el aspecto completo de la interfaz del programa Filezilla. La *Figura 3a* detalla los datos de conexión junto a la ventana inferior donde se muestra el estado de las peticiones al servidor interno, sean de éxito o de fracaso. En este caso en particular la conexión se realizó sin problemas y se listaron correctamente todos los directorios.

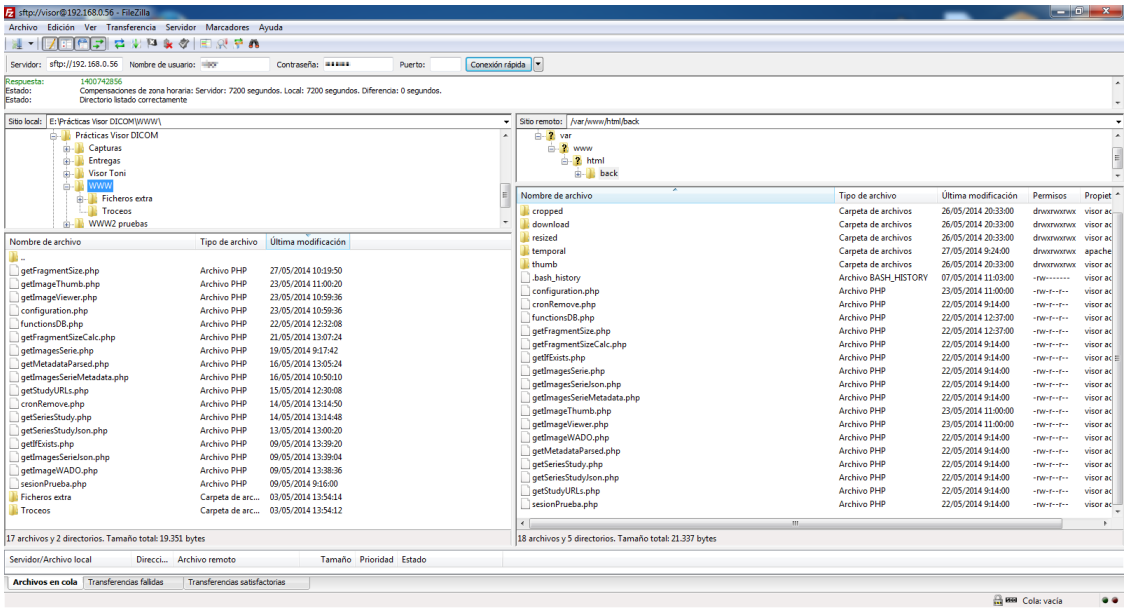

Figura 3.- Aspecto general del entorno de trabajo del proyecto con Filezilla.

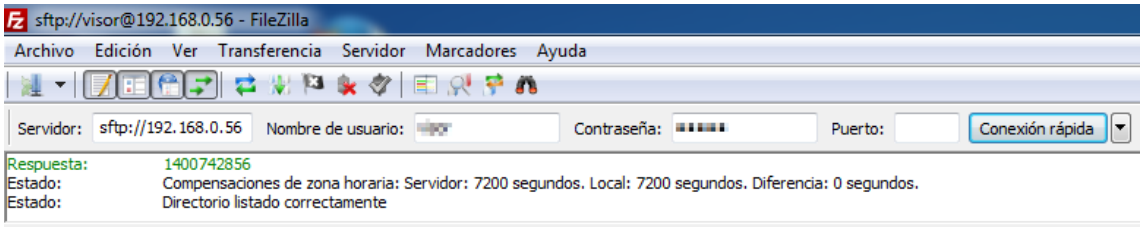

Figura 3a.- Detalle de los datos de conexión y la última respuesta del servidor interno 192.168.0.56.

La *Figura 3b* muestra la estructura de directorios de la parte del cliente (parte izquierda de la *Figura 3*), es decir, el código que desarrollo en mi ordenador para después copiarlo en el servidor y poder realizar las respectivas pruebas.

De forma análoga, la *Figura 3c* muestra la estructura de directorios de la parte del servidor (parte derecha de la *Figura 3*). Estos son los ficheros a los que llama el visor web para pedirles los datos que necesita manipular en la parte del *front-end*.

Las *figuras 3, 3a, 3b y 3c* pretenden mostrar únicamente las diferentes partes de la interfaz del programa Filezilla y no proporcionar detalles de la

implementación de ninguno de los ficheros que aparecen en ellas. Los detalles de implementación se desarrollan en el Capítulo 5.

El código desarrollado es prácticamente el elemento más importante de este proyecto, por lo que debe replicarse en varios lugares en caso de una posible catástrofe. Por ese motivo tenemos el código almacenado en varios ordenadores que además están en distintos lugares físicos.

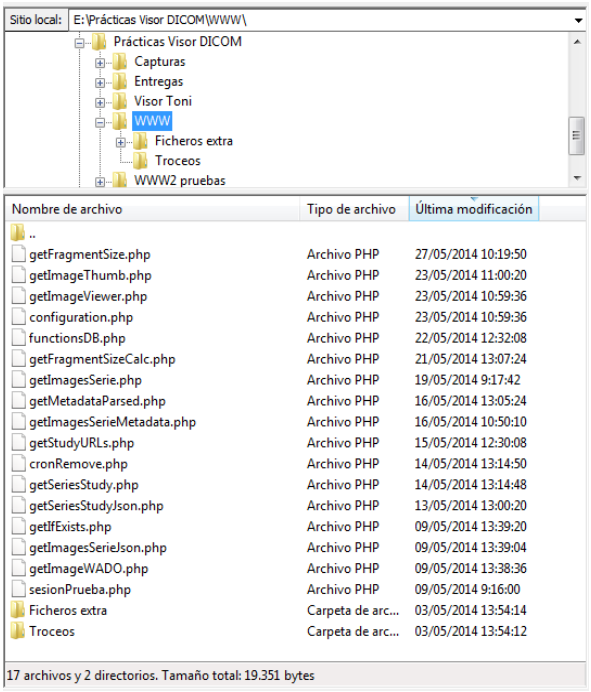

Figura 3b.- Estructura de directorios del *back-end* del sitio local.

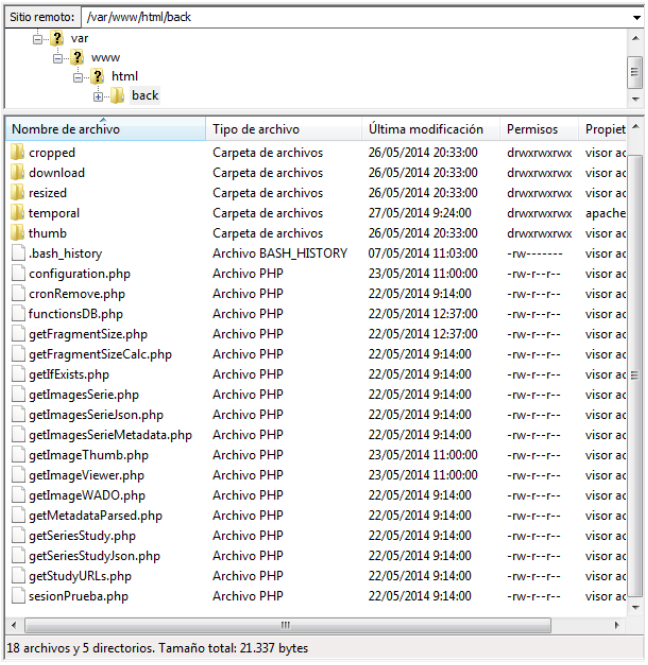

Figura 3c.- Estructura de directorios del *back-end* del servidor 192.168.0.56.

Capítulo 3

Planificación del proyecto

3.1. Introducción a la jerarquía DICOM

La base de datos de consultas sigue la jerarquía Paciente-Estudios-Series-Imágenes (detalles en la *Figura 12* del apartado 4.2). Un paciente puede haberse hecho varios estudios. A su vez, cada estudio puede estar formado por varias series y cada una de estas series por varias imágenes médicas.

Es importante tener esto en cuenta para comprender la definición de las tareas. En el apartado 4.2 se detalla el diseño de la base de datos y en el apartado 5.1 los pasos de los programas desarrollados.

3.2. Definición de tareas

Tras comprender la jerarquía que sigue la base de datos de consultas se procede a explicar la definición de las tareas a llevar a cabo para el cumplimiento de los objetivos propuestos.

Al comienzo se proporcionó un documento que recogía la interacción principal que debía existir entre el servidor y el visor web. Del mismo se han extraído las tareas principales, además de tareas adicionales que no se contemplaban desde el principio y que han ido surgiendo sobre la marcha (Tareas 4.1 y 5). A continuación se detallan todas y cada una de las tareas que el servidor debe realizar.

Tarea 1. Obtener la información de las series de un estudio

El visor web se abre en la ruta:

http://www.actualpacs.com/viewer?studyId=<%study_iuid%>

donde <%study_iuid%> se corresponde con el identificador único del estudio que se quiere abrir. Una vez abierto, el visor realiza una petición al servidor pasándole

como parámetro el *study_iuid*. Esta petición se realiza con el objetivo de que el servidor devuelva en formato JSON (*JavaScript Object Notation*) la información relativa a las series que componen el estudio, devolviendo la siguiente información por cada serie:

- Identificador único de la serie (su *series_iuid*).
- Tipo de objeto de la serie, pudiendo ser una imagen o un fichero PDF (*Portable Document Format*).
- Número total de imágenes que componen la serie.
- *Thumb*. Se trata del identificador único de la imagen (*sop_iuid*) que se desea mostrar en la previsualización de la serie al desplegar la lista de series de un estudio en pantalla.

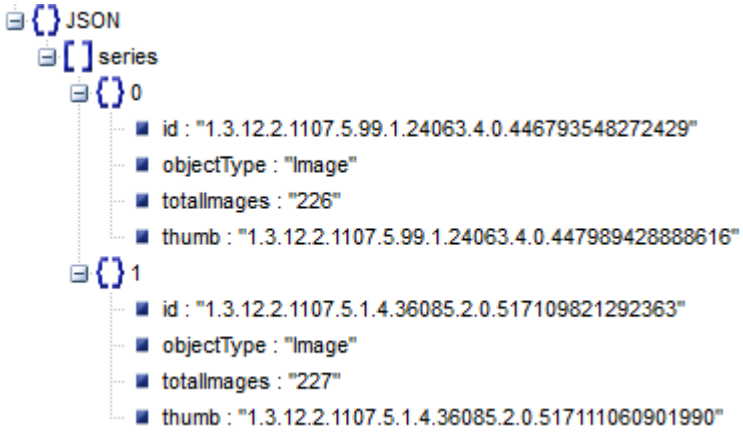

Figura 4.- Objeto JSON devuelto por el servidor como resultado de la Tarea 1.

La *Figura 4* muestra un ejemplo de cómo sería la respuesta recibida por el servidor para un estudio que contiene 2 series. La primera está formada por 226 imágenes y la segunda por 227. El visualizador web JsonViewer [4] ha resultado de gran ayuda a la hora de comprobar la correcta construcción de los objetos JSON, además de poder visualizarlos de una manera agradable y no en una sola línea. Las imágenes *thumb* o miniaturas aparecen en la columna izquierda del visor, que se detalla en la *Figura 14* del apartado 4.3.

Tarea 2. Obtener los identificadores y metadatos de una serie

Una vez cargada la interfaz de las diferentes series que componen el estudio, cuando se hace clic para abrir la serie en el visor, se realiza otra petición al servidor para que éste le diga tanto los identificadores únicos de las imágenes que componen la serie (los *sop_iuid*), así como los metadatos correspondientes a la serie. Los datos que devuelve el servidor son:

- Vector con los identificadores únicos que componen la serie. Deben aparecer ordenados en base a la columna *inst_no* tal y como se aprecia en el apartado 4.2.3.

- Objeto *metadata* asociado a la primera imagen de la serie. Este objeto se trata de un vector compuesto por los campos propios de los metadatos DICOM. Cabe aclarar que únicamente se recoge un objeto metadata porque es el mismo para todas las imágenes de una serie.

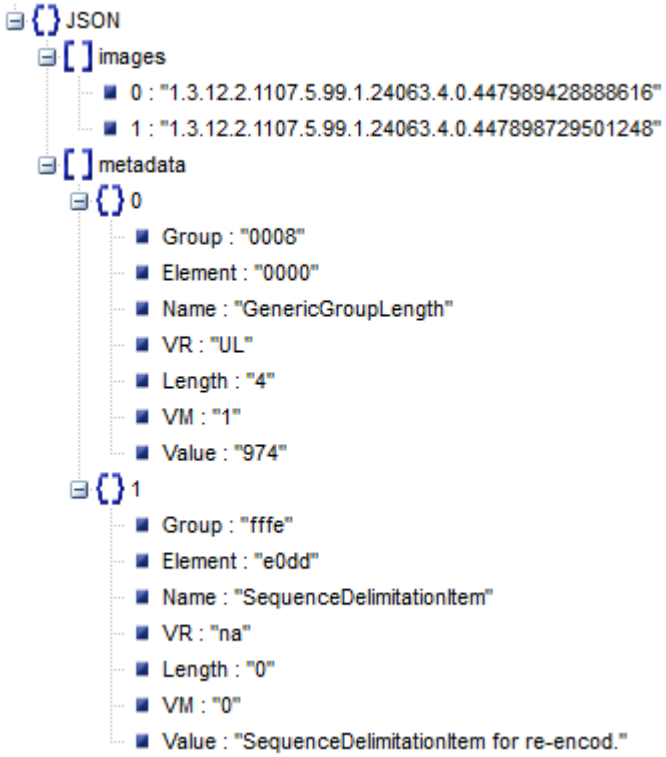

Figura 5.- Objeto JSON devuelto por el servidor como resultado de la Tarea 2.

La *Figura 5* muestra un ejemplo de cómo sería la respuesta recibida por el servidor para una serie que contiene dos imágenes y dos metadatos. El número de metadatos es siempre mucho mayor (generalmente unos 180 metadatos de media en cada serie), pero esta figura pretende mostrar únicamente la estructura de los mismos. En el apartado 5.1.2 se analiza cómo se extraen los metadatos DICOM.

Tarea 3. Calcular el ancho y alto de los fragmentos de una imagen

La estrategia elegida para la visualización de imágenes en el visor consiste en trocear la imagen que se debe mostrar en tantos fragmentos como sean necesarios para que el visor únicamente pida los que el usuario está visualizando en la ventana. De esta forma se evita descargar continuamente imágenes completas en cada petición del cliente que pueden llegar a ocupar decenas de megabytes. Con esta estrategia se aumenta la velocidad de visualización de imágenes tal y como realiza por ejemplo la aplicación Google Maps.

Cuando se dispone de la información de los datos que componen la serie, se realiza una petición al servidor pasándole como parámetro el ancho (*width*) y alto (*height*) de la imagen que se va a visualizar, de modo que desde el servidor se

devuelve el tamaño (ancho y alto) de cada fragmento en los que se dividirá la imagen. El visor necesita conocer el tamaño de cada fragmento para construirse una especie de matriz donde irá colocando los fragmentos que pedirá acto seguido al servidor.

La *Figura 6* muestra un ejemplo de cómo sería la respuesta recibida por el servidor para una imagen de 139 píxeles de ancho por 159 píxeles de alto. Partir cualquier imagen en un número exacto de fragmentos cuadrados es imposible, pero se intenta siempre que exista el mayor número de fragmentos cuadrados para mantener una homogeneidad en la conexión.

Figura 6.- Objeto JSON devuelto por el servidor como resultado de la Tarea 3.

Tarea 4.1. Descargar la imagen de previsualización de una serie

El visor necesita rellenar su columna izquierda con la primera imagen de cada una de las series de un estudio cualquiera. De esta forma, el radiólogo sabe lo que se va a encontrar dentro de una serie sin tener que abrirla antes. Por lo tanto, el visor proporciona al servidor los identificadores únicos de la imagen en concreto (*study_iuid*, *series_iuid* y *sop_iuid*) para que se la descargue, la escale y se la devuelva como imagen de miniatura o *thumb*.

Por poner un ejemplo, si la imagen de la *Figura 7a* es la que el visor ha pedido al servidor que descargue (imagen original), la respuesta que devuelve el servidor en base a esta imagen es la mostrada en la *Figura 7b* (imagen escalada).

Figura 7a.- Imagen original que el servidor ha descargado.

Figura 7b.- Imagen devuelta por el servidor como resultado de la Tarea 4.1.

Las imágenes que hay que descargar se encuentran alojadas en un servidor WADO (*Web Access to DICOM Objects*) [5]. Para descargarlas hay que enviar peticiones específicas a este servidor que pueden contener varios parámetros. Los más importantes son los identificadores únicos (UIDs), el *WindowWidth* (WW) y el *WindowLevel* (WL). El impacto del WW y WL se aprecia en la *Figura 17*.

Tarea 4.2. Descargar, escalar y trocear una imagen de una serie

Para que el visor pueda mostrar las imágenes adaptadas al ancho y alto de la ventana que las contiene, lo primero que hace es lanzar una petición al servidor con los siguientes parámetros:

- Ancho (*width*) y alto (*height*) de la imagen que se quiere mostrar en el visor web.
- Número de fragmento que se debe devolver (*tileNumber*).
- Modificadores del rango de píxeles de la imagen a descargar (*windowWidth* y *windowLevel*).
- Identificadores únicos del estudio, serie e imagen a descargar (*study_iuid*, *series_iuid* y *sop_iuid*).

Los pasos básicos que realiza el servidor al recibir los parámetros se resumen a continuación:

- Crear distintos directorios para las imágenes descargadas, escaladas y troceadas.
- Descargar la imagen pedida aplicando los parámetros *windowWidth* y *windowLevel* en la petición al servidor WADO.
- Escalar la imagen descargada aplicando los parámetros ancho (*width*) y alto (*height*).
- Trocear la imagen escalada en los fragmentos que corresponda.
- Devolver la imagen correspondiente al número de fragmento pedido (*tileNumber*).

La *Figura 8a* muestra un ejemplo de imagen descargada y troceada en cuatro fragmentos.

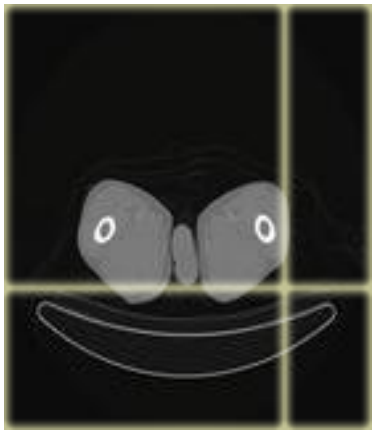

Figura 8a.- Imagen original que el servidor ha descargado y fragmentado.

Si los numeramos de izquierda a derecha y de arriba abajo comenzando desde cero, cuando el visor nos pide el fragmento número dos, la respuesta del servidor queda reflejada en la *Figura 8b*.

Figura 8b.- Imagen devuelta por el servidor como resultado de la Tarea 4.2.

Tarea 5. Implementar una memoria caché

El propósito de esta tarea es desarrollar un *script* o *cron* cuya funcionalidad es la de eliminar los ficheros del servidor relacionados con las imágenes que no se hayan visitado en las últimas horas. El visor va a recibir muchísimas peticiones por minuto en un futuro, por lo que será necesario liberar el espacio en disco que ocupan los ficheros que no se han consultado durante un determinado periodo de tiempo.

Si un radiólogo no visita el estudio de un paciente durante más de dos o tres horas, el *script* eliminará las imágenes del servidor que hacen de caché para dejar espacio a estudios con una fecha de último acceso más reciente. Estos estudios tienen una mayor probabilidad de ser nuevamente visitados.

3.3. Planificación temporal de las tareas

La *Tabla 1* muestra en color negro las tareas y costes temporales de la planificación inicial. Los cambios sobre la planificación inicial se destacan en otro color (cambios de costes temporales y nuevas tareas no contempladas al inicio).

Número de tarea	Nombre de tarea	Tiempo (horas)
1	Desarrollar la propuesta técnica	**103 → 57**
1.1	Inicio	13
1.1.1	Definir proyecto con tutor y supervisor	1
1.1.2	Definir método de trabajo y documentación	6
1.1.3	Definir formato y estándares de trabajo	6
1.2	Documentar y planificar el proyecto	50 → 18
1.2.1	Revisar contexto y buscar información	40 → 12
1.2.2	Identificar alcance y objetivos	10 → 6
1.3	Planificar el proyecto	40 → 26
1.3.1	Definir tareas	10 → 12
1.3.2	Crear diagrama de Gantt	4
1.3.3	Documentar la propuesta del proyecto	30 → 10
1.3.4	Entregar propuesta técnica	0
2	Desarrollo técnico del proyecto	**197 → 243**
2.1	Definir requisitos del proyecto	28 → 20
2.1.1	Definir y documentar requisitos de datos	14 → 10
2.1.2	Definir requisitos tecnológicos y de plataforma	14 → 10
2.2	Desarrollar el servidor	160 → 220
2.2.1	Programación	150 → 190
2.2.1.1	Tarea 1. Obtener la información de las series de un estudio	30 → 25
2.2.1.2	Tarea 2. Obtener los identificadores y metadatos de una serie	40 → 50
2.2.1.3	Tarea 3. Calcular el ancho y alto de los fragmentos de una imagen	40 → 15
2.2.1.4	Tarea 4 → Tarea 4.1. Descargar la imagen de previsualización de una serie	40 → 30
2.2.1.5	→ Tarea 4.2. Descargar, escalar y trocear una imagen de una serie	50
2.2.1.6	Tarea 5. Implementar una memoria caché	20
2.2.2	Pruebas	10 → 30
2.3	Puesta en marcha	9 → 3
2.3.1	Implantación	9 → 3
2.3.2	Entrega final	0
	Total	**300**

3	Documentación y presentación del TFG	**76**
3.1	Redacción de informes quincenales	5
3.2	Redacción de la memoria técnica	50
3.3	Entrega de la memoria técnica	0
3.4	Preparación de la presentación oral	20
3.5	Presentación oral	1

Tabla 1.- Planificación inicial y final del Trabajo Final de Grado. En un color distinto al negro se muestran las tareas que no se planificaron al inicio y las horas totales estimadas en caso de haber variado respecto a las originales.

Las horas estimadas para el desarrollo de la propuesta técnica fueron casi el doble de las que realmente han terminado siendo (103 horas iniciales frente a las 57 finales).

De este modo, las horas dedicadas para el desarrollo técnico del proyecto han aumentado de 197 a 243. La Tarea 4 inicial se ha dividido en dos tareas de semejante complejidad y se ha añadido como nueva tarea la Tarea 5.

Por último, cabe destacar que las horas finales dedicadas a la realización de pruebas y detección de errores han sido muchas más (30 horas) que las planteadas al inicio (10 horas).

La *Figura 9* presenta el diagrama de Gantt que da apoyo visual a la planificación temporal final del proyecto.

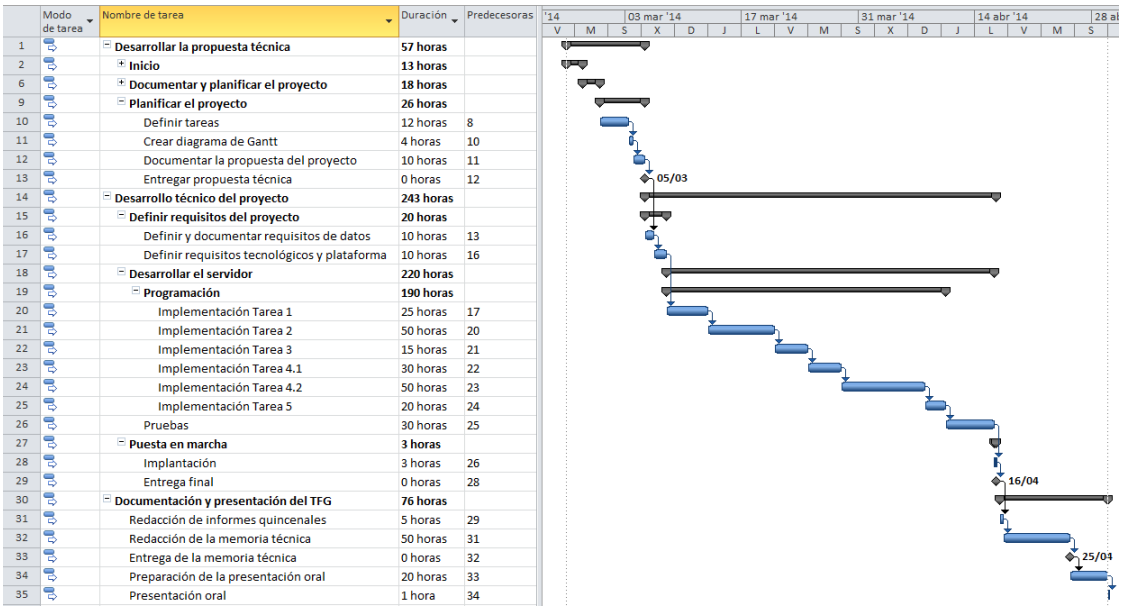

Figura 9.- Diagrama de Gantt sobre la planificación temporal definitiva (se han ignorado los días festivos).

3.4. Estimación de recursos del proyecto

Para poder desarrollar con comodidad el código del *back-end* se ha estimado la disponibilidad de un ordenador personal en la propia empresa con una buena conexión tanto a la intranet como a Internet.

El periodo de pruebas es muy importante y lo mejor es disponer de permisos de lectura y escritura sobre los directorios y ficheros que componen el entorno de prueba del visor de la empresa para no molestar continuamente a los compañeros de trabajo.

Además, es necesaria la colaboración y supervisión con el otro encargado del *back-end* y con el responsable del *front-end* para no quedarme nunca en un punto muerto durante el desarrollo del proyecto.

3.5. Presupuesto

Por lo que respecta al coste total que ha supuesto la elaboración del servidor *back-end*, podemos dividirlo en coste de hardware y software y en coste de recursos humanos.

El coste de hardware y software es nulo. El ordenador empleado para el desarrollo se encuentra completamente amortizado, ya que se continuará utilizando en este y futuros proyectos sin ningún tipo de inconveniente. El conjunto de software perteneciente al desarrollo del *back-end* es software libre. No se ha utilizado ningún programa o componente por el que se haya tenido que abonar una cantidad de dinero.

En cuanto al coste de recursos humanos, se ha cumplido la estimación de horas que había reflejadas en el contrato con la empresa, es decir, 300 horas presenciales. Se ha dedicado alguna que otra hora más fuera del lugar de trabajo pero no ha sido relevante.

En resumen, el presupuesto de este proyecto se mide en las horas-persona de trabajo que he realizado, que al tratarse de una estancia en prácticas sin beca, es gratuito.

Capítulo 4

Análisis y diseño del software

4.1. Análisis de requisitos

El análisis de requisitos se divide en dos partes: el software que va a llevar a cabo los objetivos de las tareas y el hardware donde se va a ejecutar el software.

Por lo que respecta al software, a la hora de programar es conveniente realizar un estudio sobre qué lenguaje de programación nos va a presentar más ventajas a la hora de desarrollar nuestro código. Actualmente, los lenguajes de servidor de los que más se habla son:

- ASP.NET (*Active Server Pages*).
- JSP (*Java Server Pages*).
- Node.js.
- PHP (*PHP Hypertext Pre-processor*).

La *Tabla 2* recoge la comparativa de las características más relevantes para este proyecto de estos cuatro lenguajes.

Lenguaje	Aprendizaje	Multiplataforma	Precio
ASP.NET	Intermedio	No (Windows)	Propietario
JSP	Complejo	Sí	Gratuito
Node.js	Intermedio	Sí	Gratuito
PHP	Sencillo	Sí	Gratuito

Tabla 2.- Comparativa de los lenguajes de programación en el lado del servidor analizados para este proyecto.

Los aspectos de mayor interés para decidir el lenguaje han sido la facilidad de aprendizaje, el precio y la calidad y accesibilidad de la documentación. Tras un estudio de las ventajas e inconvenientes, se ha elegido el lenguaje PHP por las siguientes razones:

- Es muy fácil de aprender.
- Se caracteriza por ser un lenguaje muy rápido.
- Es un lenguaje multiplataforma.
- Ofrece capacidad de conexión con la mayoría de los sistemas de gestión de bases de datos tradicionales (MySQL, PostgreSQL, Oracle, Microsoft SQL Server, etc.).
- Presenta la capacidad de expandir su potencial utilizando módulos.
- Posee documentación en su página oficial, la cual incluye descripción y ejemplos de cada una de sus funciones.
- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Incluye gran cantidad de funciones.

Como Sistema de Gestión de Bases de Datos (SGBD) se ha elegido MySQL porque podemos administrar sus bases de datos de una manera sencilla gracias al entorno web que suministra la herramienta phpMyAdmin. La *Figura 10* muestra el contenido de la tabla de estudios de la base de datos de consultas en la interfaz web de phpMyAdmin.

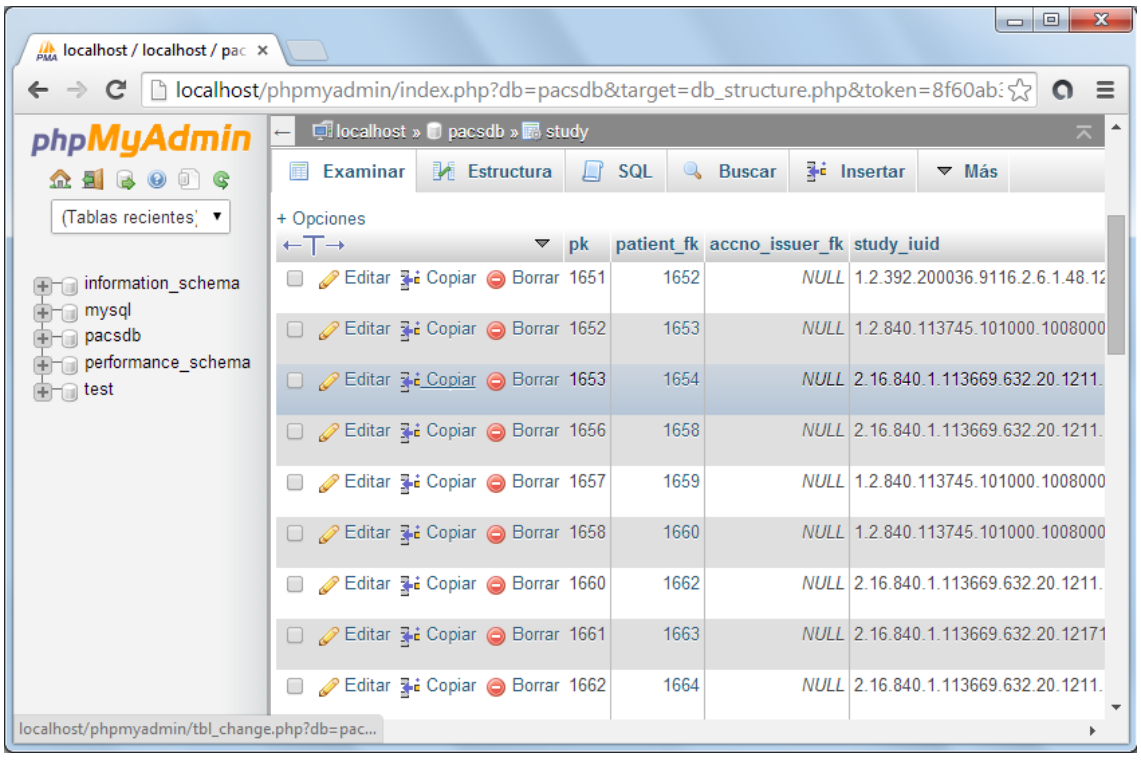

Figura 10.- Interfaz web de phpMyAdmin.

Para poder hacer uso del lenguaje PHP, las bases de datos MySQL y la herramienta phpMyAdmin, podemos instalar cada uno de estos componentes por separado o bien instalar una infraestructura de Internet que los englobe a todos. WAMP (Windows, Apache, MySQL y PHP) [6] es la infraestructura elegida para

trabajar con el código desarrollado de manera local en Windows. Existe también una versión para Linux (LAMP) y otra para Macintosh (MAMP). WAMP se descarga, se instala y ya tenemos configurado nuestro servidor web Apache, nuestro SGBD MySQL y nuestro entorno PHP, todos ellos en sus últimas versiones, ya que WAMP recibe soporte y actualizaciones de sus desarrolladores constantemente.

El software necesario para realizar las acciones que tienen que ver con la manipulación de ficheros e imágenes DICOM está formado por la herramienta DCMTK y el conjunto de utilidades ImageMagick [7]. La herramienta DCMTK se emplea para descargar y extraer los metadatos DICOM de un servidor PACS. ImageMagick brinda un amplio abanico de comandos y herramientas para manipular imágenes desde el terminal sin tener que editarlas con programas de edición de imagen. Cabe destacar que todo este software es completamente libre.

Para escribir el código necesitamos algún editor de textos avanzado que facilite visualmente esta tarea. El editor utilizado durante la elaboración del código del *back-end* ha sido Notepad++. Es rápido, sencillo, eficaz, y estructura el código como si de un entorno de desarrollo integrado se tratase.

En cuanto a los requisitos hardware, prácticamente cualquier arquitectura y sistema operativo es capaz de soportar la instalación de Apache, MySQL y PHP. Sin embargo, el servidor que ejecute el código del *back-end* necesita una gran cantidad de memoria y espacio de almacenamiento para hacer frente a todas las peticiones e imágenes que debe almacenar. Como el número de peticiones puede ir creciendo exponencialmente, la mejor solución es contratar capacidad de cómputo y espacio de almacenamiento en la nube. De esta forma se gana elasticidad, disponibilidad y un ahorro de dinero y recursos muy considerable.

4.2. Diseño de la base de datos de consultas pacsdb

La *Figura 11* muestra la totalidad de tablas y relaciones que componen la base de datos de consultas bautizada como pacsdb. No obstante, las tablas que nos interesan son únicamente tres: *Study*, *Series* e *Instance*. Este apartado tiene como objetivo describir la relación entre estas tres tablas, junto con sus columnas relevantes. La base de datos pacsdb ya estaba diseñada al inicio del proyecto.

El diccionario de datos DICOM presenta bastantes atributos, por lo que es completamente necesario que se ordenen siguiendo algún tipo de estructura lógica.

El orden del que hablamos se consigue con la jerarquía de Paciente-Estudios-Series-Imágenes, donde:

- Un paciente puede tener múltiples estudios.

- Cada estudio (tabla *Study*) puede incluir una o más series de imágenes.
- Cada serie (tabla *Series*) puede estar formada por una o más imágenes o instancias (tabla *Instance*).

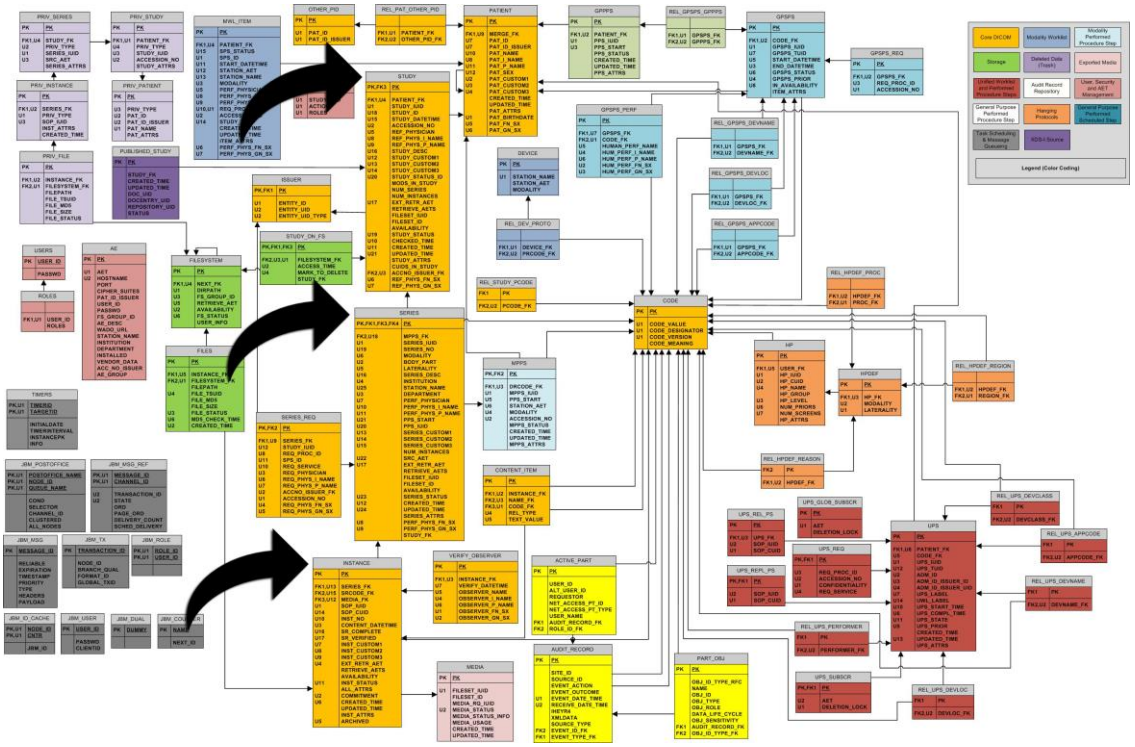

Figura 11.- Estructura de la base de datos pacsdb. Las flechas negras señalizan las tablas principales de consulta (*Study*, *Series* e *Instance*).

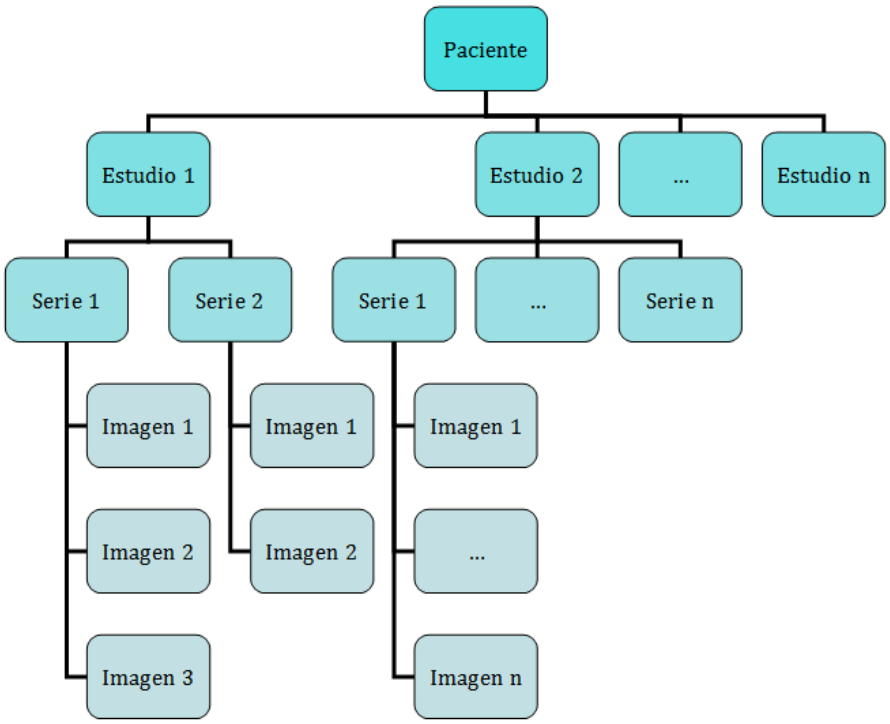

Figura 12.- Organigrama de la jerarquía Paciente-Estudios-Series-Imágenes.

Esta jerarquía refleja lo que ocurre en el mundo real cuando un paciente necesita disponer de imágenes médicas. Por ejemplo, la paciente Susana Cabedo llega a un hospital donde se le han realizado algunos estudios (tabla *Study*), como podrían ser una mamografía, un examen de ultrasonidos y una radiografía del brazo izquierdo. Cada uno de estos tres estudios estará formado por una o múltiples series de imágenes (tabla *Series*), como por ejemplo una serie con el contraste de las imágenes muy alto, otra serie coronal y otra más axial. Finalmente, cada una de estas series estará formada por las propias imágenes o instancias (tabla *Instance*), desde una hasta muchas.

En la *Figura 12* observamos de forma más directa la distribución de los estudios de un paciente cualquiera tal y como se ha descrito.

Así pues, se dispone de cuatro niveles, cada uno de ellos con su propio identificador único. Cabe destacar que estos identificadores no se corresponden con las claves primarias ni ajenas de las tablas. Las claves primarias se denominan como "pk", del inglés *primary key*. Las claves ajenas se llaman por el nombre de la tabla seguido de "_fk", del inglés *foreign key* (por ejemplo *study_fk* o *series_fk*).

Así pues, ha llegado el momento de observar las columnas de interés de las tablas *Study*, *Series* e *Instance* que se habían señalizado en la *Figura 11*.

4.2.1. La tabla de Estudios (*Study*)

El identificador único o UID (*Unique IDentifier*) de cada estudio de la base de datos se localiza en la columna *study_iuid*. En la *Tabla 3* se puede ver que estos identificadores no son de tipo numérico sino de tipo cadena. Estas cadenas las genera el sistema automáticamente, ya que su complejidad y extensión haría una odisea introducirlas a mano.

pk ▲	patient_fk	study_iuid	num_series	num_instances
1651	1652	1.2.392.200036.9116.2.6.1.48.1214227973.1234955411.426484	4	25
1652	1653	1.2.840.113745.101000.1008000.38179.6792.6324567	3	680
1653	1654	2.16.840.1.113669.632.20.1211.10000330985	2	638
1656	1658	2.16.840.1.113669.632.20.1211.10000502993	7	589
1657	1659	1.2.840.113745.101000.1008000.38306.6432.6715426	17	816
1658	1660	1.2.840.113745.101000.1008000.38048.4626.5933732	2	166
1660	1662	2.16.840.1.113669.632.20.1211.10000231621	1	166
1661	1663	2.16.840.1.113669.632.20.121711.10000160881	6	368
1662	1664	2.16.840.1.113669.632.20.1211.10000307912	3	541

Tabla 3.- Columnas de interés de la tabla *Study*.

La columna *num_series* indica de cuántas series está compuesto el estudio y la columna *num_instances* especifica la suma total de imágenes que tiene el estudio entre todas sus series.

La clave primaria "pk" corresponde con la clave ajena *study_fk* de la tabla *Series*.

4.2.2. La tabla de Series (*Series*)

El tercer nivel de la jerarquía lo ocupa la tabla Series (*Tabla 4*). Los identificadores únicos de las series se recogen en la columna *series_iuid*. La columna *num_instances* indica de cuántas imágenes o instancias está formada cada serie de imágenes. Para evitar cualquier posible ambigüedad, el concepto "imagen" es equivalente a "instancia".

pk	study_fk	series_iuid	num_instances
5697	1651	1.2.392.200036.9116.2.6.1.48.1214227973.1234958039.831513	15
5698	1651	1.2.392.200036.9116.2.6.1.48.1214227973.1234958664.140428	8
5699	1651	1.2.392.200036.9116.2.6.1.48.1214227973.1234957889.836576	1
5700	1651	1.2.392.200036.9116.2.6.1.48.1214227973.1234958610.628440	1
5701	1652	1.3.12.2.1107.5.99.1.24063.4.0.446793548272429	226
5702	1652	1.3.12.2.1107.5.1.4.36085.2.0.517109821292363	227
5703	1652	1.3.12.2.1107.5.1.4.36085.2.0.517714246252254	227
5704	1653	1.2.840.113704.1.111.672.1161866320.1	244
5705	1653	1.2.840.113704.1.111.4848.1161868385.1	394
5708	1656	1.3.12.2.1107.5.2.31.30222.30000007072005313478100000444	28
5709	1656	1.3.12.2.1107.5.2.31.30222.30000007072005313478100000474	24
5710	1656	1.3.12.2.1107.5.2.31.30222.30000007072005313478100000570	28
5711	1656	1.3.12.2.1107.5.2.31.30222.30000007072005313478100000600	29

Tabla 4.- Columnas de interés de la tabla *Series*.

La clave primaria "pk" corresponde con la clave ajena *series_fk* de la tabla *Instance*.

4.2.3. La tabla de Imágenes o Instancias (*Instance*)

En el nivel más bajo de la jerarquía se encuentra la tabla *Instance*, donde reside la información de las propias imágenes (*Tabla 5*). El identificador único de cada imagen se encuentra en la columna *sop_iuid*.

pk	series_fk	sop_iuid	inst_no
162336	5697	1.2.392.200036.9116.2.6.1.48.1214227973.1234958039.838055	1
162337	5697	1.2.392.200036.9116.2.6.1.48.1214227973.1234958041.873583	2
162338	5697	1.2.392.200036.9116.2.6.1.48.1214227973.1234958043.896459	3
162339	5697	1.2.392.200036.9116.2.6.1.48.1214227973.1234958045.927006	4
162340	5697	1.2.392.200036.9116.2.6.1.48.1214227973.1234958047.958078	5

Tabla 5.- Columnas de interés de la tabla *Instance*.

La columna *inst_no* indica el orden que ocupa cada imagen dentro de la serie. Se trata de una columna muy importante porque si aparecieran las imágenes desordenadas no se podrían visualizar de forma lógica en el visor web.

4.3. Diseño de la interfaz

Tal y como se ha comentado en el apartado 1.2, el diseño de la interfaz del visor web se ha llevado a cabo por una tercera persona de manera independiente. Sin embargo, antes de pasar con la implementación de las tareas del siguiente capítulo es conveniente observar la interacción que se produce entre el *front-end* o la Vista y el *back-end* o el Modelo-Controlador.

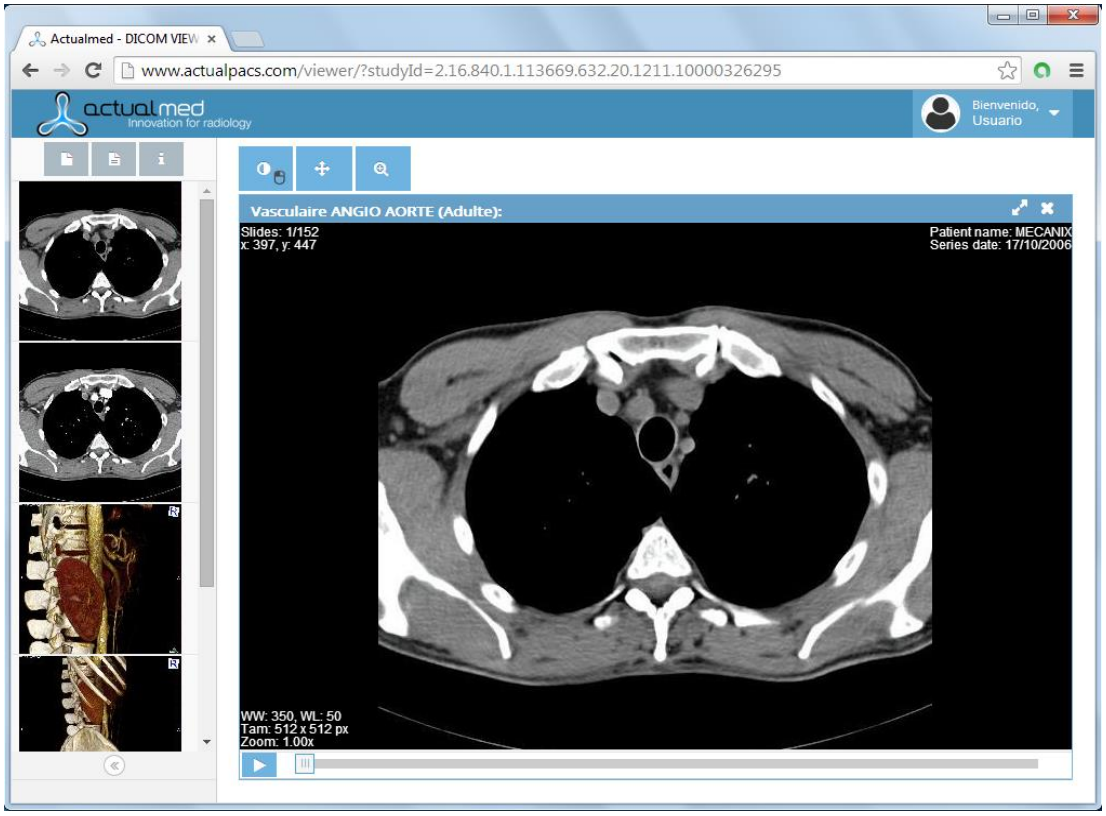

Figura 13.- Aspecto general de la interfaz del visor web DICOM.

La *Figura 13* muestra el aspecto general del visor tras haber abierto la información relativa de un estudio determinado. A continuación se resumen las interacciones más importantes en función de las acciones que realiza el usuario.

El usuario abre un estudio. La Vista pide al Controlador que el Modelo:

→ Le proporcione los identificadores únicos (*series_iuid*) de las series que componen el estudio.

→ Le comunique el identificador único (*sop_iuid*) de la primera imagen de cada una de las series.

→ Descargue la primera imagen de cada una de las series para mostrarlas en la columna izquierda del visor (*Figura 14*) a modo de previsualización de lo que el usuario va a encontrar en cada serie.

Figura 14.- Columna izquierda del visor. Cada imagen representa una serie distinta.

El usuario abre una serie. La Vista pide al Controlador que el Modelo:

→ Le proporcione los identificadores únicos (*sop_iuid*) de las imágenes que componen la serie.

→ Descargue y le comunique los metadatos de la serie (*Figura 15*).

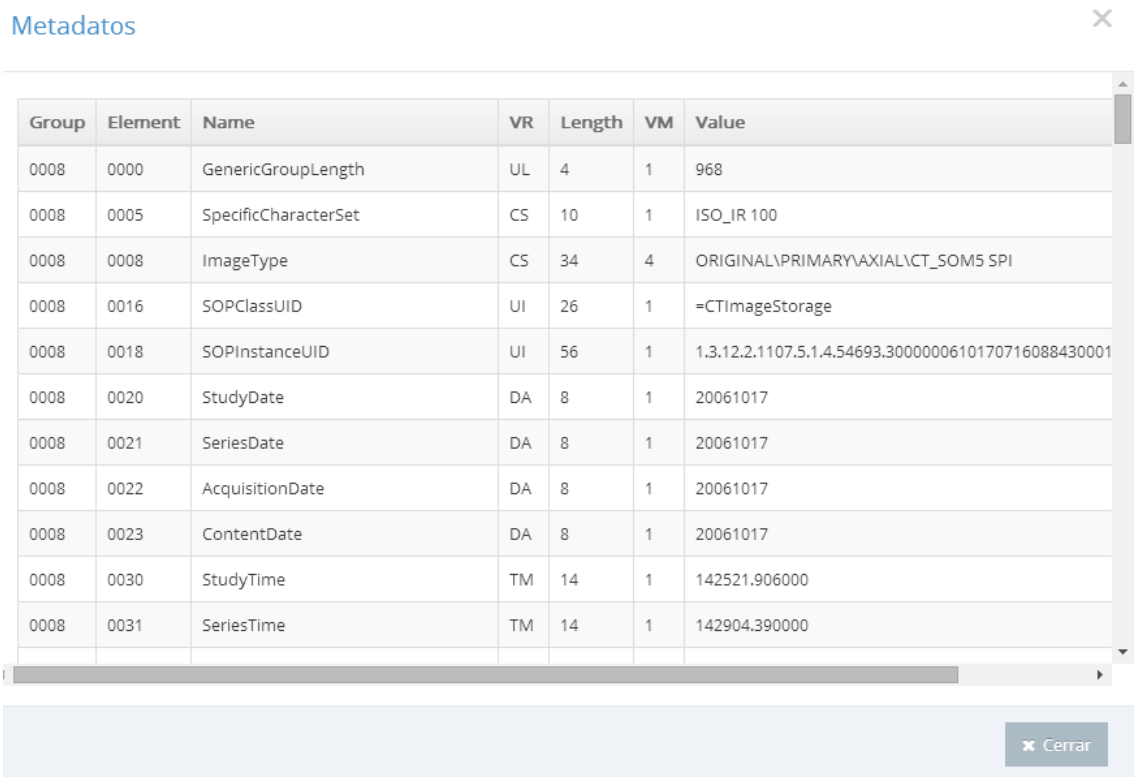

Figura 15.- Ventana donde se muestran los metadatos de una serie en forma de tabla.

→ Descargue, trocee y le muestre cada una de las imágenes que componen la serie en la ventana del visor (*Figura 16*).

El usuario utiliza las herramientas disponibles para manipular las imágenes:

Rango de píxeles mostrados (WindowWidth/WindowLevel):

Se utiliza para resaltar zonas de la radiografía como por ejemplo los huesos. El usuario selecciona la herramienta, hace clic sobre la imagen y antes de soltar el botón, arrastra el cursor para ver cómo quedarán los cambios en tiempo real.

Posición de la imagen:

Sirve para desplazar la imagen actual a través de la ventana de visualización. El usuario selecciona la herramienta, hace clic sobre la imagen y antes de soltar el botón, arrastra el cursor para desplazarla donde desee.

Tamaño de la imagen:

Su función es ampliar o reducir la imagen actual. El usuario selecciona la herramienta, hace clic sobre la imagen y antes de soltar el botón, arrastra el cursor hacia arriba para ampliar y hacia abajo para reducir el tamaño de la imagen.

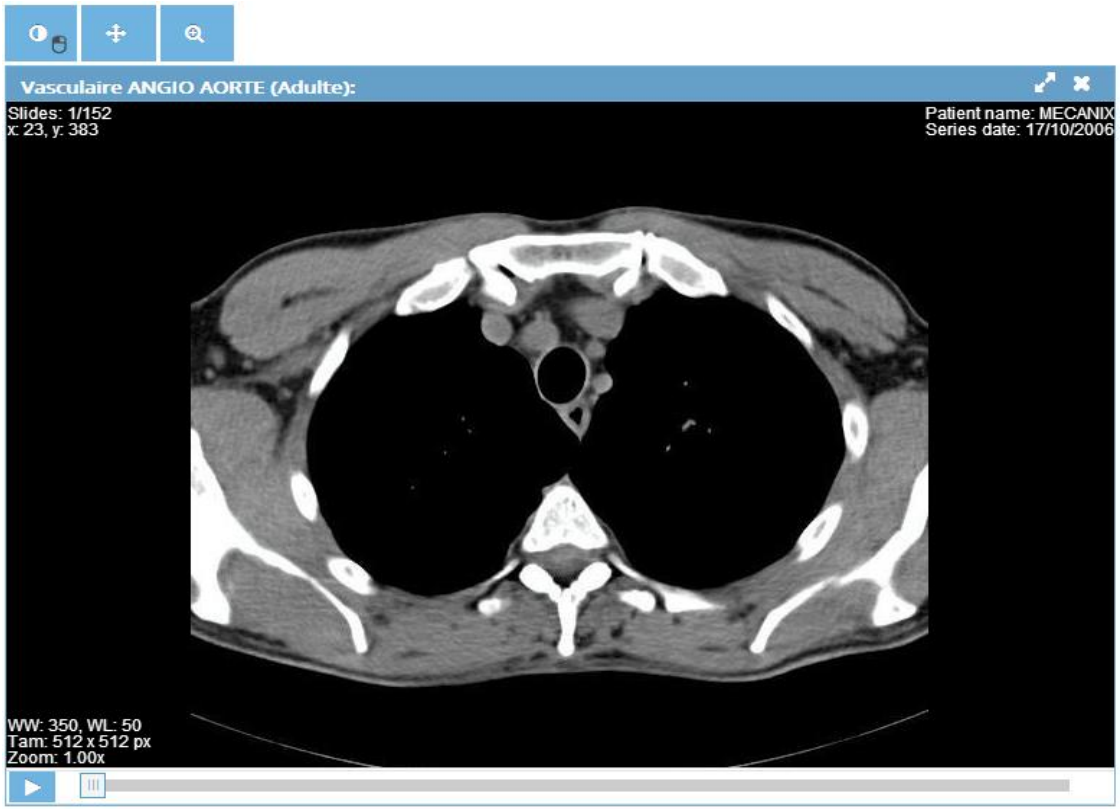

Figura 16.- Zona del visor donde se muestran y manipulan las imágenes de cada una de las series del estudio.

Cada vez que se suelta el botón del ratón utilizando la herramienta que modifica el rango de píxeles mostrados (WW/WL), la Vista pide al Controlador que el Modelo:

→ Descargue, trocee y le muestre las nuevas imágenes con los parámetros WW y WL que el usuario desea.

La *Figura 17* muestra un ejemplo de cómo sería el resultado para dos movimientos del ratón distintos aplicados a una misma imagen tras utilizar esta herramienta.

Los cambios realizados con la herramienta de desplazamiento no generan nuevas peticiones, ya que no se necesita descargar ninguna imagen nueva puesto que no se modifican sus píxeles, sino su posición en el visor.

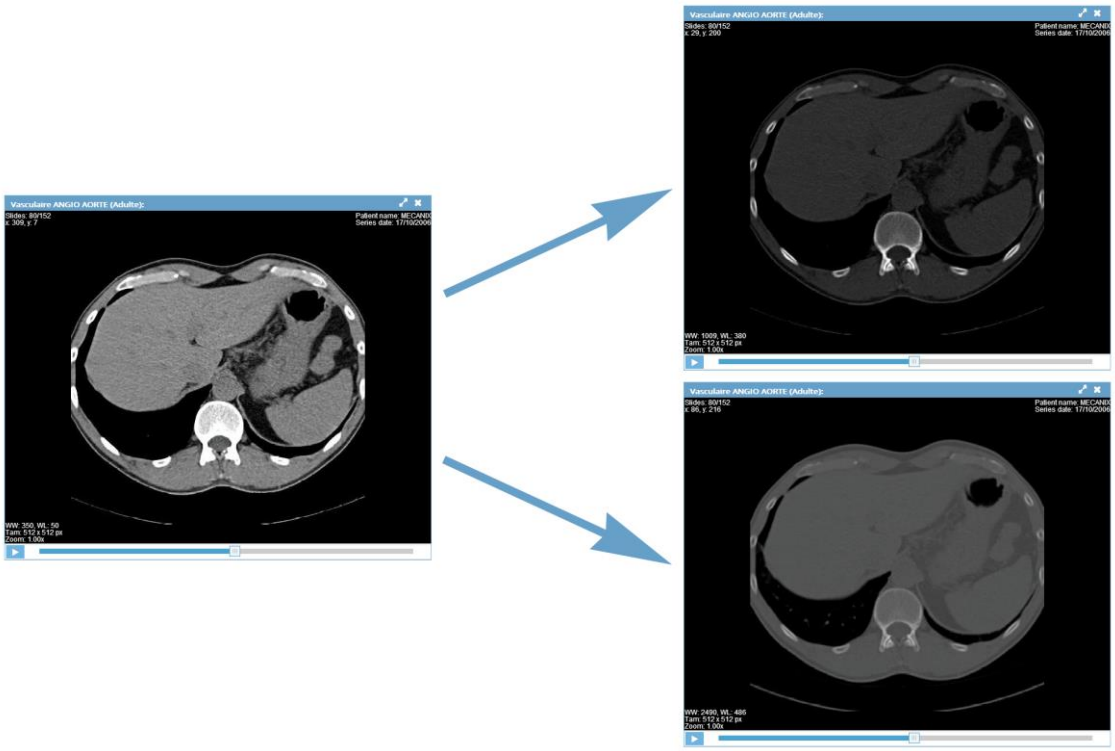

Figura 17.- Ejemplo de dos resultados diferentes tras utilizar la herramienta para modificar el rango de píxeles mostrados (WW/WL). En la parte izquierda se muestra la imagen original y en la parte derecha la misma imagen tras aplicarle distintos valores de *WindowWidth* y *WindowLevel*.

Por su parte, los cambios realizados con la herramienta de variación del tamaño de la imagen generarán nuevas peticiones únicamente si la imagen original es más grande que el tamaño de la ventana donde se visualiza. En ese caso, la Vista pide al Controlador que el Modelo:

→ Descargue, trocee y le muestre las nuevas imágenes que pertenezcan únicamente a la zona de la imagen original que el usuario ha decidido ampliar y a sus zonas vecinas. Así se evita realizar un número incontable de peticiones que saturarían rápidamente al servidor.

La *Figura 18* muestra un ejemplo de cómo sería el resultado para una imagen de gran resolución tras utilizar esta herramienta y ampliar una porción de la misma.

El visor web funciona muy bien cuando ya dispone de los resultados de las peticiones del usuario de anteriores ejecuciones. Su velocidad en estos casos es prácticamente comparable a la de un visor de escritorio que tiene desde el inicio todas las imágenes en el disco duro.

Figura 18.- Ejemplo del resultado tras realizar una ampliación de una zona concreta de una imagen. En la izquierda, la imagen original. En el centro y en la derecha el fragmento de la imagen ampliado antes (baja resolución) y después (alta resolución) del procesamiento de las peticiones.

Para finalizar este capítulo, se muestra el aspecto del visor en el navegador Google Chrome de un teléfono móvil con sistema operativo Android 4.2.2 (*Figura 19*). Como se puede observar, el visor está diseñado para adaptarse a cualquier tamaño de ventana, ocultando las diversas funcionalidades en pestañas para que la visualización de las imágenes tome protagonismo en pantalla.

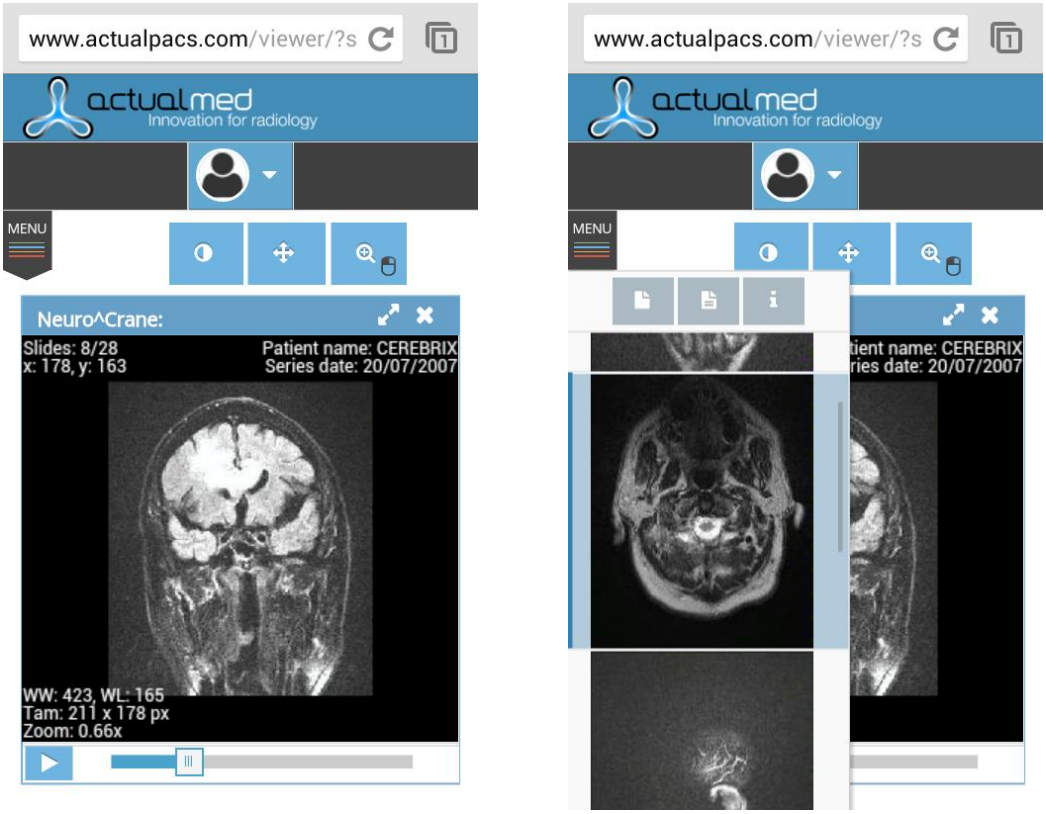

Figura 19.- Aspecto del visor web DICOM en un smartphone. La columna de las series se oculta dentro del botón “Menu”.

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

Durante los siguientes subapartados se desarrollan los detalles de implementación más relevantes llevados a cabo para la consecución de los objetivos de las tareas propuestas de este proyecto. Cada descripción de las tareas comienza presentando la estructura de la petición realizada por el visor web, seguida del orden de ejecución de los programas involucrados en la resolución de la petición.

Es importante mencionar que los programas PHP elaborados cuentan con un nivel de modularidad que ha ido evolucionando semana tras semana. La ventaja principal que aporta la modularidad es la subdivisión de una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Gracias a esto, la depuración de errores es mucho más rápida y sencilla, incluso para una persona ajena al proyecto.

Además, la modularidad brinda al código un alto nivel de dinamismo. Si tenemos los parámetros de acceso a la base de datos en un único punto, será más sencillo adaptarse a otra base de datos que si estuvieran replicados por todos los programas PHP que deben acceder a ella. Lo mismo ocurre con las conexiones a otros servidores, como el servidor WADO, o con las rutas absolutas de los directorios donde se descargan y trocean las imágenes médicas. Si tenemos todos los datos que pueden variar en un único programa de configuración, la adaptabilidad del código aumenta considerablemente.

5.1.1. Obtener la información de las series de un estudio

Estructura de la petición:

→ .../getSeriesStudy.php?studyId=<study_iuid>

Programa getSeriesStudy.php

→ Llama a las funciones de functionsDB.php y getSeriesStudyJson.php y muestra el objeto JSON.

Programa functionsDB.php

→ Comprueba qué parámetros relacionados con la base de datos (*study_iuid*, *series_iuid*, *sop_iuid*) debe recoger de la petición.

→ Recoge el *study_iuid* y deja vacíos al resto.

→ Conecta con la base de datos de consultas pacsdb.

→ Ejecuta la sentencia SQL correspondiente y devuelve el resultado a getSeriesStudyJson.php.

Programa getSeriesStudyJson.php

→ Recoge el resultado de la sentencia SQL.

→ Crea una matriz.

→ Recorre el resultado de la sentencia SQL y guarda cada fila en la matriz.

→ Convierte la matriz en un objeto JSON (*Figura 4*) y lo devuelve a getSeriesStudy.php.

5.1.2. Obtener los identificadores y metadatos de una serie

Estructura de la petición:

→ .../getImagesSerie.php?studyId=<study_iuid>& serieId=<series_iuid>

Programa getImagesSerie.php

→ Llama a las funciones y variables de functionsDB.php, getImagesSerieJson.php, configuration.php, getImagesSerieMetadata.php y getMetadataParsed.php y muestra el objeto JSON (*Figura 5*).

Programa functionsDB.php

→ Comprueba qué parámetros relacionados con la base de datos (*study_iuid*, *series_iuid*, *sop_iuid*) debe recoger de la petición.

→ Recoge el *study_iuid* y el *series_iuid*, dejando vacío el *sop_iuid*.

→ Conecta con la base de datos de consultas pacsdb.

→ Ejecuta la sentencia SQL correspondiente y devuelve el resultado a getImagesSerieJson.php.

Programa getImagesSerieJson.php

→ Recoge el resultado de la sentencia SQL.

→ Crea una matriz.

→ Recorre el resultado de la sentencia SQL y guarda cada fila en la matriz.

→ Si la matriz no está vacía, se guarda en una variable el *sop_iuid* de la primera imagen de la matriz.

→ En caso contrario, se guarda un valor vacío para que PHP no devuelva un mensaje de advertencia o *warning* que arruine la comunicación.

→ Convierte la matriz en un objeto JSON y la devuelve a getImagesSerie.php junto con la variable donde reside el *sop_iuid*.

Programa configuration.php

Para obtener los metadatos deseados es necesario que el servidor donde se ejecuta el *back-end* tenga instalada la herramienta DCMTK (*DICOM ToolKit*). El software DCMTK es una colección de librerías y aplicaciones que implementan gran parte del estándar DICOM. A continuación se describen los dos comandos de DCMTK relevantes en este proyecto:

- Comando **getscu** [8]: envía una petición a un servidor PACS preguntando por una imagen con un *sop_iuid* concreto. El fichero DICOM que se recibe como respuesta se almacena en un directorio llamado "temporal".
- Comando **dcmdump** [9]: vuelca el contenido del fichero DICOM anterior en la salida estándar o *stdout* para poder ser manipulado y extraer así sus metadatos.

El programa configuration.php contiene la ruta absoluta de los comandos getscu *(/usr/local/bin/getscu*), dcmdump *(/usr/local/bin/dcmdump*) y el directorio "temporal" *(/var/www/html/back/temporal*) en sus respectivas variables.

Programa getImagesSerieMetadata.php

→ Recibe las variables de configuration.php y getImagesSerie.php (el *sop_iuid*).

→ Ejecuta el comando getscu.

→ Ejecuta el comando dcmdump.

→ Devuelve el resultado (los metadatos) a getImagesSerie.php, que a su vez se lo envía a getMetadataParsed.php para que extraiga los metadatos en el formato JSON deseado.

Programa getMetadataParsed.php

→ Recibe los metadatos en su salida estándar (*stdout*).

→ Crea una matriz.

→ Trocea cada línea de metadatos hasta poder extraer la información relevante y guardarla en la matriz.

→ Convierte la matriz en un objeto JSON y lo devuelve a getImagesSerie.php para que lo concatene con el JSON devuelto por getImagesSerieJson.php y poder construir así el objeto JSON deseado por el visor.

Para que sea más sencillo comprender lo que hace el programa getMetadataParsed.php, imaginemos que los metadatos que recibe son los mostrados en la *Figura 20*.

```
[14] => # Dicom-Data-Set
[15] => # Used TransferSyntax: Little Endian Implicit
[16] => (0008,0000) UL 974                                          #   4, 1 GenericGroupLength
[17] => (0008,0005) CS [ISO_IR 100]                                 #  10, 1 SpecificCharacterSet
[18] => (0008,0008) CS [ORIGINAL\PRIMARY\AXIAL\CT_SOM5 SPI]         #  34, 4 ImageType
[19] => (0008,0016) UI =CTImageStorage                              #  26, 1 SOPClassUID
[20] => (0008,0018) UI [1.3.12.2.1107.5.99.1.24063.4.0.8888616]     #  46, 1 SOPInstanceUID
[21] => (0008,0020) DA [20040721]                                   #   8, 1 StudyDate
[22] => (0008,0021) DA [20040701]                                   #   8, 1 SeriesDate
[23] => (0008,0022) DA [20040721]                                   #   8, 1 AcquisitionDate
[24] => (0008,0023) DA [20040721]                                   #   8, 1 ContentDate
[25] => (0008,0030) TM [100851.980000]                              #  14, 1 StudyTime
[26] => (0008,0031) TM [120000.000000]                              #  14, 1 SeriesTime
[27] => (0008,0032) TM [101233.956026]                              #  14, 1 AcquisitionTime
[28] => (0008,0033) TM [101233.956026]                              #  14, 1 ContentTime
[29] => (0008,0050) SH [1909715]                                    #   8, 1 AccessionNumber
[30] => (0008,0060) CS [CT]                                         #   2, 1 Modality
[31] => (0008,0070) LO [SIEMENS]                                    #   8, 1 Manufacturer
[32] => (0008,0080) LO [UCLA PET/CT]                                #  12, 1 InstitutionName
```

Figura 20.- Ejemplo de metadatos devueltos tras la ejecución del comando dcmdump. Se han marcado en rojo los campos de interés de cada línea.

Los campos que debe extraer de cada una de las líneas para incluirlos en cada posición de la matriz son:

- ***Group***: primer número de cuatro dígitos (*Ej.: 0008*).
- ***Element***: segundo número de cuatro dígitos (*Ej.: 0030*).
- ***Name***: última cadena que comienza por letra mayúscula (*Ej.: StudyTime*).
- ***VR***: cadena formada por dos letras mayúsculas (*Ej.: TM*).
- ***Length***: número comprendido entre "#" y "," (*Ej.: 14*).
- ***VM***: número comprendido entre los campos "Length" y "Name" (*Ej.: 1*).
- ***Value***: cadena de contenido variable, comprendida entre el campo "VR" y el carácter "#" (*Ej.: 100851.980000*).

5.1.3. Calcular el ancho y alto de los fragmentos de una imagen

Estructura de la petición:

→ .../getFragmentSize.php?width=<int>&height=<int>

Programa getFragmentSize.php

→ Llama a la función de getFragmentSizeCalc.php.

→ Recoge el resultado de getFragmentSizeCalc.php.

→ Construye el objeto JSON (*Figura 6*) y lo muestra.

Programa getFragmentSizeCalc.php

→ Recibe el ancho y alto (enteros mayores que cero).

→ Divide ambos parámetros entre un porcentaje predefinido.

→ Redondea los dos resultados.

→ Devuelve el resultado más grande de los dos, que simboliza el ancho y alto (mismo valor, pues los fragmentos serán cuadrados) de cada fragmento en los que se dividirá la imagen original.

5.1.4. Descargar la imagen de previsualización de una serie

Estructura de la petición:

→ .../getImageThumb.php?
studyId=<study_iuid>&serieId=<series_iuid>&objectId=<sop_iuid>

Programa getImageThumb.php

→ Llama a las funciones y variables de functionsDB.php, getIfExists.php, configuration.php y getImageWADO.php.

→ Si la consulta recibida contiene resultados, comprueba si se creó con anterioridad el directorio donde se debe almacenar la imagen descargada del servidor WADO (directorio "thumb").

→ En caso afirmativo, actualiza la fecha de último acceso de la imagen y la muestra al visor.

→ En caso negativo, crea el directorio, descarga la imagen y la muestra al visor.

Programa functionsDB.php

→ Comprueba qué parámetros relacionados con la base de datos (*study_iuid*, *series_iuid*, *sop_iuid*) debe recoger de la petición.

→ Recoge el *study_iuid*, el *series_iuid* y el *sop_iuid*.

→ Conecta con la base de datos de consultas pacsdb.

→ Ejecuta la sentencia SQL correspondiente y devuelve el resultado a getImageThumb.php.

Programa getIfExists.php

→ Recibe el resultado de una consulta SQL y comprueba si está vacío o no.

Se utiliza para abortar la ejecución del programa getImageThumb.php en caso de recibir un resultado vacío. De esta forma se evita la creación de directorios inservibles.

Programa configuration.php

Este programa también contiene las rutas absolutas en sus respectivas variables de los diferentes directorios donde se deben almacenar las imágenes descargadas (directorios "download" y "thumb"), escaladas (directorio "resized") y troceadas (directorio "cropped"). Son cuatro directorios en total, pero para esta tarea será necesario únicamente el directorio "thumb".

Programa getImageWADO.php

→ Recibe una petición WADO y la ruta del directorio "thumb".

→ Conecta con la URL (*Uniform Resource Locator*) de la petición WADO.

→ Descarga la imagen con un tamaño predefinido.

→ Abre el directorio "thumb", escribe la imagen y lo cierra.

5.1.5. Descargar, escalar y trocear una imagen de una serie

Estructura de la petición:

→ .../getImageViewer.php?
studyId=<study_iuid>&serieId=<series_iuid>&objectId=<sop_iuid>&
width=<int>&height=<int>&tileNumber=<int>&
windowWidth=<int>&windowLevel=±<int>

Programa getImageViewer.php

→ Llama a las funciones y variables de functionsDB.php, getIfExists.php, configuration.php, getImageWADO.php y getFragmentSizeCalc.php.

→ Si la consulta recibida contiene resultados, comprueba si se creó con anterioridad el directorio donde se debe almacenar el fragmento (*tileNumber*) de la imagen descargada del servidor WADO (directorio "cropped").

→ En caso afirmativo, actualiza la fecha de último acceso del fragmento y lo muestra al visor.

→ En caso negativo, crea el directorio "download" y descarga la imagen completa.

→ Crea el directorio “resized” y guarda en él una copia de la imagen original, escalada con los parámetros alto (*height*) y ancho (*width*).

→ Recupera el alto y ancho de los fragmentos en los que se debe trocear la imagen escalada proporcionados por getFragmentSizeCalc.php.

→ Crea el directorio “cropped”, trocea la imagen escalada y guarda en él los fragmentos.

→ Muestra el fragmento al visor en base al *tileNumber* pedido.

Programa functionsDB.php

→ Comprueba qué parámetros relacionados con la base de datos (*study_iuid*, *series_iuid*, *sop_iuid*) debe recoger de la petición.

→ Recoge el *study_iuid*, el *series_iuid* y el *sop_iuid*.

→ Conecta con la base de datos de consultas pacsdb.

→ Ejecuta la sentencia SQL correspondiente y devuelve el resultado a getImageViewer.php.

Programa getIfExists.php

→ Recibe el resultado de una consulta SQL y comprueba si está vacío o no.

Se utiliza para abortar la ejecución del programa getImageViewer.php en caso de recibir un resultado vacío. De esta forma se evita la creación de directorios inservibles.

Programa configuration.php

Contiene las rutas absolutas en sus respectivas variables de los diferentes directorios donde se deben almacenar las imágenes descargadas (directorio “download”), escaladas (directorio “resized”) y troceadas (directorio “cropped”).

Programa getImageWADO.php

→ Recibe una petición WADO y la ruta del directorio “download”.

→ Conecta con la URL de la petición WADO.

→ Descarga la imagen aplicando los parámetros *windowWidth* y *windowLevel*.

→ Abre el directorio “download”, escribe la imagen y lo cierra.

Programa getFragmentSizeCalc.php

→ Recibe el ancho y alto (enteros mayores que cero).

→ Divide ambos parámetros entre un porcentaje predefinido.

→ Redondea los dos resultados.

→ Devuelve el resultado más grande de los dos, que simboliza el ancho y alto (mismo valor, pues los fragmentos serán cuadrados) de cada fragmento en los que se dividirá la imagen original.

5.1.6. Implementar una memoria caché

Estructura de la petición:

→ .../cronRemove.php

Programa cronRemove.php

→ Abre el directorio que se le pasa como parámetro.

→ Mientras queden elementos en su interior, analiza cada elemento.

→ Si es un directorio (distinto de los directorios "." y ".."), se llama a sí mismo de forma recursiva.

→ Si es un fichero (una imagen médica), compara su fecha de último acceso con la fecha actual, y si la diferencia es superior a un tiempo predefinido, lo elimina. De lo contrario, no hace nada.

Programa configuration.php

Contiene las rutas absolutas en sus respectivas variables de los diferentes directorios que debe analizar el programa cronRemove.php en base a la fecha de último acceso (directorios "download", "thumb", "resized", "cropped" y sus respectivos subdirectorios).

5.2. Validación y pruebas

Una vez desarrollada la primera versión de cada una de las tareas, el encargado de la implementación del visor comienza a probar los programas del *back-end* haciéndoles peticiones.

Para la validación de los programas desarrollados se ha mantenido una comunicación constante con el encargado del visor. Gracias a este intercambio de información se ha conseguido definir con total exactitud el resultado a devolver por cada tarea.

Tras la fase de validación comienza la fase de pruebas con los estudios anonimizados de la base de datos pacsdb. Tanto el responsable del visor como yo, hemos estado abriendo en el visor web diversos estudios para comprobar la rapidez, limitaciones y fallos que se producen.

Tuvimos problemas con ciertos estudios que no se cargaban en el visor por el mero hecho de contener metadatos con caracteres en una codificación distinta a UTF-8 (*Figura 21*). Al construir el objeto JSON con los metadatos, éste se devolvía vacío. La solución fue codificar estos metadatos en UTF-8 y estos estudios no volvieron a fallar.

```
[35] => (0008,1010) SH [CT54693]                                  #   8, 1 StationName
[36] => (0008,1030) LO [Extr�mit�s inf�rieures Pied cheville]     #  50, 1 StudyDescription
[37] => (0008,1032) SQ (Sequence with explicit length #=1)        #  72, 1 ProcedureCodeSequence
[38] =>   (fffe,e000) na (Item with explicit length #=4)          #  64, 1 Item
[39] =>     (0008,0000) UL 52                                     #   4, 1 GenericGroupLength
[40] =>     (0008,0100) SH [CTPIED]                               #   6, 1 CodeValue
[41] =>     (0008,0102) SH [XPLORE]                               #   6, 1 CodingSchemeDesignator
[42] =>     (0008,0104) LO [CT pied-cheville]                     #  16, 1 CodeMeaning
```

Figura 21.- Ejemplo de metadatos cuyos caracteres se encuentran en una codificación distinta a UTF-8. Se han localizado gracias a que el navegador los muestra con un símbolo de interrogación.

Otro problema del que nos dimos cuenta fue que a la hora de descargar una imagen con el comando getscu para extraer sus metadatos, si la imagen pesaba más de 25 megabytes, el visor tardaba muchísimo en obtener los metadatos o incluso no llegaba a obtenerlos nunca (*Figura 22*).

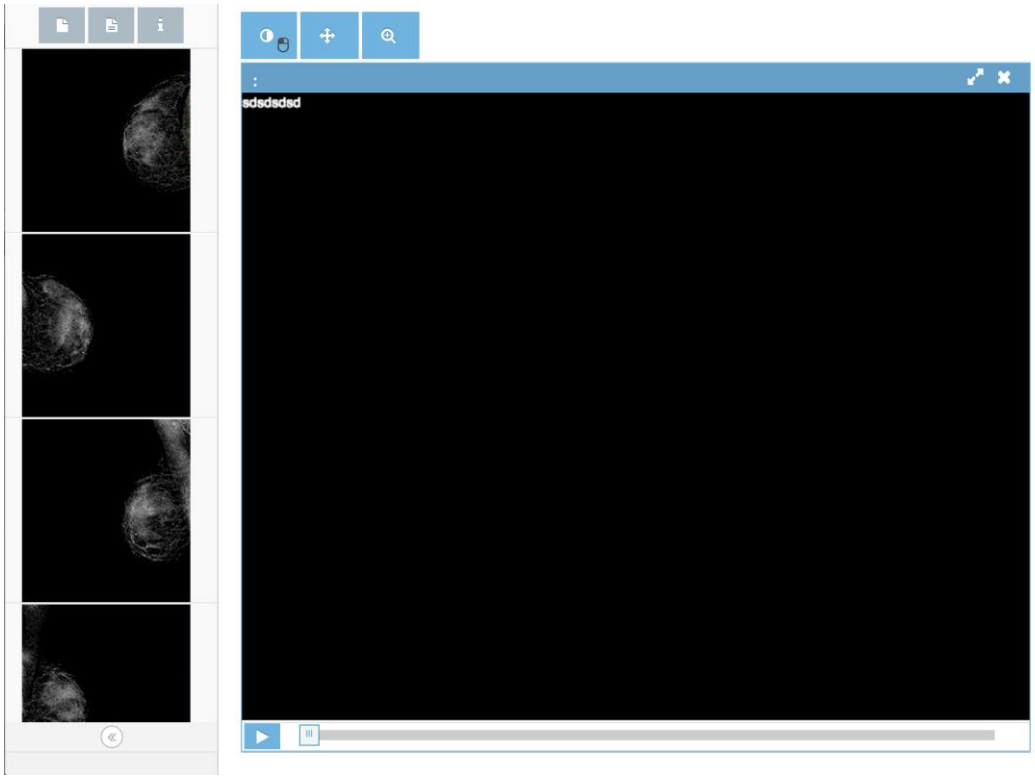

Figura 22.- Ejemplo de estudio de una mamografía que el visor no conseguía cargar.

Nos dimos cuenta que estábamos descargando las imágenes sin comprimir, por lo que modificamos las opciones del comando getscu para descargarlas

comprimidas y ahorrar tiempo, ya que el objetivo de getscu en este proyecto no es la imagen, sino sus metadatos asociados.

Para alcanzar un rendimiento razonable cuando el usuario pide un estudio por primera vez, se ha tenido que ir probando el visor con distintos criterios de troceado para las imágenes descargadas. Tras realizar estas pruebas se ha establecido que es preferible trocear las imágenes en fragmentos grandes para que el visor no realice muchas peticiones pidiendo un gran número de fragmentos pequeños.

Por último, se implementó un programa de ayuda (getStudyURLs.php) que recoge todas las posibles URL con todos los estudios de la base de datos pacsdb para que sea más cómodo realizar pruebas durante el desarrollo del visor (*Figura 23*).

```
1  http://www.actualpacs.com/viewer/?studyId=1.2.840.113745.101000.1008000.38179.6792.6324567
2  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000330985
3  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000502993
4  http://www.actualpacs.com/viewer/?studyId=1.2.840.113745.101000.1008000.38306.6432.6715426
5  http://www.actualpacs.com/viewer/?studyId=1.2.840.113745.101000.1008000.38048.4626.5933732
6  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000231621
7  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.121711.10000160881
8  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000307912
9  http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000329900
10 http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000322126
11 http://www.actualpacs.com/viewer/?studyId=2.16.840.1.113669.632.20.1211.10000326295
12 http://www.actualpacs.com/viewer/?studyId=1.3.12.2.1107.5.2.12.21264.4.0.8969333128827685
```

Figura 23.- Ejemplos de URL generadas por el programa getStudyURLs.php para copiar y pegar directamente en un navegador web.

Capítulo 6

Conclusiones

Gracias al esfuerzo depositado durante la implementación y el periodo de evaluación y pruebas, ha sido posible crear un Controlador y un Modelo para hacer frente a las acciones que va a realizar un médico radiólogo con el visor web DICOM para imagen radiológica.

Se han estudiado al detalle los programas que componen el *back-end* para que ninguna de las acciones que realiza se convierta en un considerable cuello de botella.

El visor web ofrece un rendimiento muy similar a los visores de escritorio cuando el usuario accede por segunda vez a imágenes ya descargadas, escaladas y troceadas por el Modelo previamente.

Sin embargo, cuando se accede por primera vez a un estudio que presenta un gran número de series y una gran cantidad de imágenes por serie, el visor no responde todo lo rápido que cabría esperar. La Vista necesita el resultado de muchísimas peticiones al mismo tiempo y el lenguaje PHP no está pensado para el paralelismo de forma nativa.

Como trabajo futuro de este proyecto se espera la incorporación de nuevas herramientas de manipulación de imágenes en el visor. De momento, el visor web es útil cuando el médico radiólogo no tiene acceso a un visor de escritorio. Con las herramientas básicas que ofrece el visor web, el radiólogo puede ejecutar un primer diagnóstico de manera rápida y eficaz en casos de extrema urgencia.

Además, las pruebas se han realizado en un entorno muy pequeño. A este visor aún le queda un largo recorrido para convertirse en un producto final. Lo ideal sería que el servidor de prueba donde se ha depositado el código del *back-end* fueran en realidad muchos servidores, y si éstos pertenecen a la nube, mucho mejor, pues ganamos elasticidad y ahorro ya que pagamos únicamente por lo que usamos.

Por último, me gustaría destacar un entorno de programación en la capa del servidor diferente a PHP que está tomando fuerza en estos tiempos y podría agilizar mucho las operaciones del Modelo. Se trata de Node.js, basado en el lenguaje JavaScript y creado con el enfoque de aportar utilidad en la creación de programas de red altamente escalables.

Si migrásemos el código PHP desarrollado a esta tecnología, conseguiríamos que el troceado de imágenes fuera mucho más rápido gracias a la concurrencia que ofrece. Claro está que tendríamos que cambiar la filosofía de programación, además de la base de datos utilizada. No obstante, cualquier cambio que ayude a mejorar la velocidad de un visor web para imagen radiológica será crucial para el éxito y generalización del mismo.

Bibliografía

- Libro de consulta:

  [1] OLEG, S.P. 2008. *Digital Imaging and Communications in Medicine (DICOM): a practical introduction and survival guide*. Boston: Springer. ISBN 978-3-540-74570-9.

- Páginas web:

  [2] Wikipedia, the free encyclopedia (http://en.wikipedia.org/)

  [3] PHP: Hypertext Preprocessor (http://www.php.net/manual/es/)

  [4] JSON Viewer (http://jsonviewer.stack.hu/)

  [5] WADO Server (http://www.research.ibm.com/software/wado/)

  [6] WAMP Server (http://www.wampserver.com/en/)

  [7] ImageMagick, Wikipedia (http://es.wikipedia.org/wiki/ImageMagick)

  [8] DCMTK (getscu) (http://support.dcmtk.org/docs/getscu.html)

  [9] DCMTK (dcmdump) (http://support.dcmtk.org/docs/dcmdump.html)