INSTITUTO TECNOLÓGICO DE TELEFÓNICA

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

CURSO ACADÉMICO 2017/2019

TRABAJO DE FIN DE GRADO

APLICACIÓN PARA LA GESTIÓN DE UNA CLÍNICA MÉDICA

MBSalud

APLICACIÓN PARA LA GESTIÓN DE UNA CLÍNICA MÉDICA

MBSalud

Autor: Ana Belén Fernández Díaz

Tutor: Raquel Cerdá Losa

INDICE

1. MODULOS FORMATIVOS APLICADOS EN EL PROYECTO.

Programación multimedia y dispositivos móviles. App en android.

Programación java.

Entornos de desarrollo. Sistema de control de versiones, Git, uso de GitHub.

Bases de datos y acceso a datos.

Desarrollo de interfaces. Puesta en marcha de las medidas de usabilidad de las aplicaciones, realización de pruebas y testeo de las aplicaciones.

Empresa en iniciativa emprendedora. Innovación de proceso y marketing.

Programación de servicios y procesos.

Sistemas informáticos.

2. OBJETIVOS DEL PROYECTO.

El objetivo principal de este proyecto consiste en crear una aplicación móvil multiplataforma utilizando tecnologías novedosas para ello.

La aplicación está enfocado a la sanidad, en concreto a la gestión de un centro médico de salud u hospital donde tanto pacientes como especialistas médicos pueden hacer uso de ella.

La elección de esta idea de proyecto surge con visión de la necesidad de un cambio hacía lo digital por parte de los consultorios para agilizar su gestión y por parte de los usuarios de hacer un uso eficaz, responsable y de compromiso con los servicios médicos.

Se trata de proveer la integración de aplicaciones móviles en este tipo de negocio con un panel de gestión intuitivo y fácil de manejar, configurar y personalizar que facilitará el sistema de organización de la empresa y proporcionará gran fidelidad a los usuarios y especialista.

Desde esta aplicación un usuario podría solicitar, editar, eliminar y consultar sus citas; el profesional médico podrá comprobar los pacientes por fecha y visualizar su historial, anular, añadir y consultar citas por paciente, todo ello de forma rápida y sencilla, ya que una de los principales objetivos de este producto es la forma de llegar al usuario, por lo que una interfaz amigable y la facilidad de uso se hacen indispensables en este proyecto.

El principal objetivo a destacar de este proyecto, es el reto que supone llevar a cabo un software desde cero, conocer todo el proceso de desarrollo de una app desde su concepción hasta su distribución, planificando todas sus etapas, desde el estudio y análisis, hasta la codificación y las pruebas necesarias; lo que conlleva una planificación detallada del tiempo y los recursos, así como la puesta en práctica de las capacidades y conocimientos adquiridos a lo largo del ciclo y otros conocimientos necesarios que se han requerido durante todas las fases.

3. HERRAMIENTAS, LENGUAJES Y TECNOLOGIAS UTILIZADAS

Las principales tecnologías necesarias para el desarrollo del proyecto son las siguientes:

JAVA: lenguaje de programación utilizado en el desarrollo de aplicaciones Android.

HTTP : protocolo de transferencia de hipertexto. Necesario para realizar la comunicación entre la aplicación Android y el servidor que contiene los datos.

Android Studio : Android Studio es el entorno de desarrollo integrado oficial para la plataforma Android

JSON: (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.
JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un *objeto*, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arreglos, vectores, listas o secuencias.

XML: es otro lenguaje a emplear en este proyecto cuya principal función es describir datos cuyo formato posibilita la lectura de datos a través de diferentes aplicaciones y en este proyecto servirá de interfaz grafica para la configuración de las activities.

Sus siglas significan eXtensible Markup Language, que en español es Lenguaje de Marcado Extendible**.** Es un lenguaje basado en etiquetas descriptivas, cuyo fin es representar información de texto en forma jerárquica y legible, y a su vez darle formato. XML cobra importancia al surgir la necesidad de interoperabilidad entre dos sistemas de información, ya que propone un puente de intercambio entre dos fuentes de datos distintas.

Realmente no es exactamente un lenguaje tal cual, sino un sistema que permite definir lenguajes. XML nació para manejar y estructurar opciones de configuración para sistemas operativos o aplicaciones, pero posteriormente se eligió para intercambiar información entre distintas tecnologías, debido a que establece un solo formato para obtener datos. Esto facilita importar y exportar información.

XML propone una sintaxis cómoda para las maquinas y los humanos, siendo un lenguaje afable para la construcción de estructuras complejas.

Entre las cualidades de esta tecnología podemos destacar:

- Independencia de datos y forma: Con XML podemos establecer la estructura de la información y al mismo tiempo separar la forma en que se interpretará dependiendo el contexto. Significa que si tenemos un entorno A y un entorno B que usaran los datos de un archivo XML, solo definimos los datos y cambiamos su forma dependiendo de cada contexto.
- Extensibilidad: En cualquier momento podemos agregar nuevos componentes y atributos a un archivo XML.
- Representación jerárquica: XML permite expresar anidaciones entre componentes. Lo que permite establecer un comportamiento de nodos padre e hijo.
- Interoperabilidad: Esta característica es mortal en XML, ya que permite intercambiar información entre aplicaciones, módulos o sistemas sin importar la arquitectura o tecnología usada.

PHP: Es un lenguaje totalmente libre y abierto normalmente dedicado al desarrollo web y que puede ser incrustado en lenguaje HTML. Utilizado también en servidores para gestionar llamadas http que lo hace perfecto para alcanzar los objetivos de este proyecto.

Tiene una curva de aprendizaje muy baja, su sintaxis es simple y cumple estándares básicos de la programación orientada a objetos. No son necesarios complejos entornos de desarrollo, que incluso necesitan su propio periodo de aprendizaje. Los IDEs disponibles son gratuitos y los entornos de desarrollo son de rápida y fácil configuración.

SQL: es un lenguaje universal de acceso a datos, que se puede aplicar en diferentes tipos de bases de datos relacionales. Con SQL se pueden manejar bases de datos Access, MySql y tb otros modelos relaciones como Oracle, SqlServer.

MySql: es el sistema de gestión de base de datos elegido ya que es un gestor de base de datos *relacional*(bd en la que la información está relacionada. Las tablas están relacionadas entre sí), *multihilo*(la bd soporta varios procesos a la vez, podemos realizar varias peticiones a la bd de forma simultánea) y *multiusuario* (bd en la que hay varios hay usuarios conectados pidiendo o modificando la información).

Una característica muy importante de Mysql es q es libre, es decir, es gratuita. Otra cosa muy importante de Mysql es que se puede descargar el código fuente de mysql y se puede modificar al gusto. Es un sistema muy potente que tiene gran comunidad de soporte.

PHPMyAdmin: que es como una consola de administración desde la que es más sencillo manejar las bases de datos. Permite administrar de manera visual mis bases de datos muy fácilmente desde el navegador

Apache Tomcat: Se opta por Apache como servidor web http por su filosofía de código abierto y por su esencia de multiplataforma; además de la existencia de multitud de documentación, su gran calidad como software y por su gran integración con las dos tecnologías elegidas(PHP y MySql).

Es necesario disponer de un servidor web para hacer las pruebas de mi aplicación.

Para ello existen dos opciones: por un lado, se puede contratar un espacio web de una empresa de hosting y cada vez que se quiere hacer una prueba hay que subir el código y luego la pruebas a través del navegador. Esto hace que sea visible para todo el mundo al estar en el hosting.

Otra opción es instalar un servidor web en el ordenador para convertirlo en servidor para hacer las pruebas de forma sencilla sin que sean visibles. Para esto hay que instalar tres programas:

Servidor web: Apache. Windows.

Extension PHP, es para que el ordenador sea capaz de interpretar el lenguaje.

Gestor de base de datos: MySQL que es gratuito y tiene gran comunidad de soporte.

Existen paquetes que traen las tres cosas, son los llamados Bundles,que son paquetes de servidores web para Windows totalmente gratuitos y de código abierto que permiten instalar desde un único paquete un servidor Apache con PHP y un servidor MySQL para poder ejecutar en local cualquier aplicación que desarrollemos por nuestra cuenta sin tener que subir los archivos a un servidor web dedicado conectado a la nube.

Los más utilizados son XAMPP, WAMP, EASYPHP. En mi caso he usado la plataforma WAMP. Todos estos paquetes son muy parecidos, lo único que cambia es la interfaz.

WAMP es un acrónimo que significa Windows, Apache, MySQL y PHP. Es un stack o conjunto de soluciones de software que significa que cuando instalas WAMP, estás instalando Apache, MySQL y PHP en tu sistema operativo (Windows en el caso de WAMP).

Lo que representa cada letra es lo siguiente:

- **"**W" significa Windows; también hay LAMP (para Linux) y MAMP (para Mac).
- "A" significa Apache. Apache es el software de servidor que se encarga de servir las páginas web. Cuando se solicita ver una página, Apache cumple la solicitud a través de HTTP y muestra el sitio.
- "M" significa MySQL. El trabajo de MySQL es ser el sistema de gestión de base de datos para el servidor. Almacena toda la información relevante, como el contenido de tu sitio, los perfiles de usuario, etc.
- "P" significa PHP. Es el lenguaje de programación en el cual está escrito y actúa como aglutinante para toda este stack de soluciones. PHP se ejecuta junto con Apache y se comunica con MySQL.

API: Siglas de Interfaz de Programación de Aplicaciones. Es un conjunto de funciones, métodos o procedimientos que ofrece una biblioteca para ser utilizado por otro software como una capa de abstracción.

Servicio Web: Tecnología que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

Recurso: Datos (arrays, imágenes, cadenas, números, estilos) que se almacenan independientemente del código de aplicación para mejorar la organización del proyecto y permitir su adaptación a cambios.

Cliente: Aplicación informática que consume un servicio remoto de otro ordenador conocido como servidor.

Servidor: Nodo que forma parte de una red y provee de servicios a nodos denominados clientes.

Multiplataforma: Válido para varios sistemas.

Balsamiq Mockups: Herramienta para crear prototipos, bocetos o wireframes. Programa de escritorio que tiene una interfaz fácil de usar y tienen varios objetos prediseñados que ofrece para su uso. Permite exportar el diseño en png y pdf. Me ha resultado muy útil.

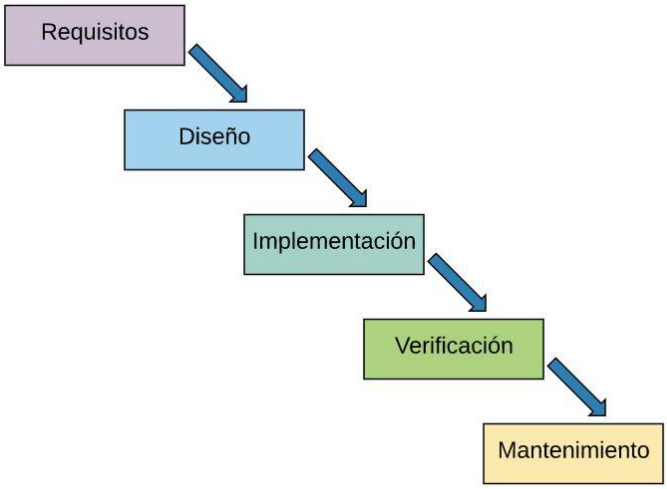
ARC: Advanced Rest Client es una extensión de Chrome que nos permite lanzar peticiones a servicios o APIs RestFul. Puedes hacer cualquier tipo de petición además de las habituales GET, POST, PUT y DELETE. Permite pasar parámetros a las peticiones y muestra el resultado devuelto por el servicio que queremos probar. Su funcionamiento es muy simple y no requiere ningún tipo de configuración especial.

4. FASES DEL PROYECTO

En este apartado se describe la secuencia de actividades y las diferentes tareas a seguir para generar el desarrollo organizado de software requerido, es decir el ciclo de vida del desarrollo software.

He considerado que el modelo en cascada (waterfall) es el más acertado para el desarrollo de este proyecto, siguiendo una serie de etapas de forma sucesiva, de modo que una etapa comienza cuando termina la etapa anterior.

En un primer momento ha sido laborioso el diseño de todos los requisitos y funcionalidades correctamente, donde era necesario que fuesen claros y no iban a cambiar a lo largo del proceso de desarrollo, pero esto ha permitido seguir una estructura ordenada en la que he seguido una seria de etapas de forma sucesiva donde la etapa siguiente comienza cuando termina la etapa anterior y esto ha sido muy útil para el desarrollo del proyecto.



Etapas del modelo en cascada.-

En esta fase ha sido necesario determinar qué elementos intervienen en el sistema a desarrollar y cuál sería el comportamiento esperado del proyecto. Para ello, he estudiado otras aplicaciones de gestión sanitaria existentes en el mercado de similares características, y una vez comprobado su funcionamiento y en base a los comentarios que los usuarios reflejaban en las reseñas de la aplicación he extraído una serie de requisitos que he considerado necesarios incluir en este proyecto para desarrollar una aplicación pudiese alcanzar los objetivos esperados por los usuarios.

4.1. REQUISITOS:

4.1.1. Gestión de los usuarios paciente y los usuarios profesionales médicos.

4.1.1.1. Usuario paciente:

La aplicación debe permitir al usuario introducir sus datos, en este caso su documento nacional de identidad y una contraseña, para registrarse o loguearse cumplimentando un formulario y será el sistema el encargado de validar esos datos que si son correctos le redirigirá a la pantalla principal con la sesión iniciada; o por el contrarió informará con un mensaje de error de que la validación no se ha realizado correctamente.

Cuando el usuario se loguea por primera vez puede activar la casilla de "Recordar contraseña" para mantener los datos de acceso cumplimentados para el siguiente inicio de sesión.

Por otra lado, una vez iniciada la sesión puede salir de la aplicación a través del botón de "Cerrar sesión" que le redirigirá a la pantalla principal pero sin estar logueado.

El usuario tiene una pantalla que le permite, una vez que crea su perfil, acceder a modificar los datos de la cuenta por si en algún momento han cambiado. Puede solicitar cambiar su contraseña de acceso o solicitar eliminar su cuenta y en ese momento se borrarán todos sus datos de la base de datos y se da de baja de la aplicación. Cualquiera de estas modificaciones será validada por el sistema.

4.1.1.2. Usuario profesional médico:

El profesional médico únicamente puede acceder a la aplicación a través de un login con su número de identificación personal y su contraseña. Nunca podrá registrarse o darse de alta desde la aplicación puesto que debe estar dado de alta en la base de datos para poder acceder.

Este usuario formará parte de la lista del cuadro médico que oferta el centro sanitario propietario de la aplicación siendo este último el encargado de dar de alta cada profesional médico para permitirle el acceso a través de la app.

Cuando el usuario profesional médico se loguea por primera vez puede activar la casilla de "Recordar contraseña" para mantener los datos de acceso cumplimentados para el siguiente inicio de sesión.

Una vez iniciada la sesión puede salir de la aplicación a través del botón de "Cerrar sesión" que le redirigirá a la pantalla principal pero sin estar logueado.

4.1.2. Funcionalidad de la pantalla principal de la aplicación tanto del usuario paciente como del profesional médico.

4.1.2.1. Pantalla inicial del usuario paciente:

Desde la pantalla principal de paciente este puede realizar las siguientes funciones:

-"PEDIR CITA": El paciente puede pedir una cita con los diferentes profesionales médicos que oferta el centro eligiendo ente el periodo de fechas deseado. Una vez que se confirma la cita el sistema muestra un cuadro de dialogo con los datos de la cita y le remite un correo electrónico al paciente con los detalles de la misma.

-"VER MIS CITAS": El paciente puede comprobar todas las citas que han sido anuladas, que están pendientes o que han sido anuladas.

-"HISTORIAL": Muestra todas las asistencias del paciente al centro médico y puede visualizar una a una el tratamiento o las recomendaciones del profesional médico.

-"CUADRO MÉDICO: El paciente puede ver el cuadro médico con los profesionales médicos que oferta el centro seleccionando por especialidad interesada.

-"MIS DATOS" : Permito al usuario acceder a sus datos de la cuenta donde puede modificar sus datos personales, cambiar la contraseña o eliminar su perfil.

-"CONTACTO": El paciente puede ver los datos de contacto del centro de salud.

4.1.2.2. Pantalla inicial del usuario profesional médico:

-"CITAS": El especialista puede ver todas las citas con los pacientes según fecha, diariamente , semanales o mensuales.

Puede asignar nuevas citas a los pacientes desde la misma ventana y con cada cita reservada el sistema envía un correo electrónico al paciente con los detalles de la cita.

-"PACIENTES": El especialista puede consultar todos sus pacientes y comprobar cuales han sido sus visitas completadas y acceder al detalle de la ficha de ese paciente con los resultados de cada una de esas visitas.

Puede ver las citas que tiene pendientes con ese paciente y añadir el motivo de la asistencia, el diagnósticos, recomendaciones o lo que considere oportuno.

Puede eliminar una cita pendiente con ese paciente y pasa a estado anulada. Todas la citas anuladas de ese paciente también figurarán en su histórico de citas.

Puede añadir un paciente nuevo y registrarlo.

4.1.3. He considerado necesario que la aplicación tuviese una interfaz un tanto atractiva, sencilla, estructurada e intuitiva procurando evitar los errores; rápida y con un buen rendimiento a la hora de hacer las consultas a la base de datos que no generen tiempos de respuesta muy elevados para que no suponga un esfuerzo al usuario a la hora de hacer uso de ella.

4.1.4. La seguridad.

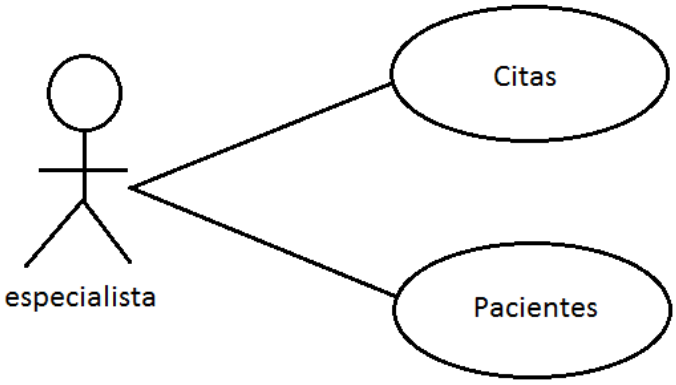
En cualquier proyecto de la actualidad se debe prestar mucha atención en la seguridad, para ello se tiene que disponer de acceso como mínimo con usuario y contraseña. En mi caso, se habilitará la autenticación mediante dni y contraseña tanto para los pacientes como para los profesionales médicos.

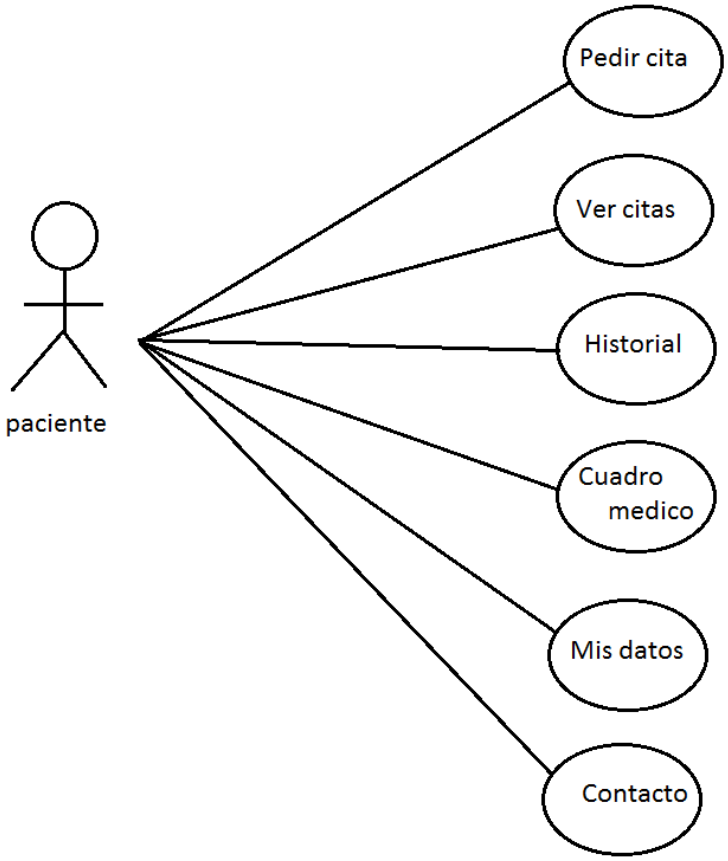
En mi aplicación hay una ligera capa de seguridad. La aplicación se comunicará con el servidor por medio de llamadas HTTP para enviar y recibir información. Esta información puede verse en peligro si el atacante consigue interceptar las cabeceras enviadas al servidor y podría obtener datos muy valiosos como contraseñas y nombres de usuario.
En mi aplicación he usado md5, que es un algoritmo que no se usa mucho actualmente pero consideraba necesaria una pequeña capa de seguridad que en este caso lo que hace es que si alguien logra entrar en la base de datos no podrá visualizar las contraseñas de los pacientes ni de los profesionales médicos, es más, ni tan siquiera el administrador habitual de la base de datos puede verlas.

Lo recomendables sería cifrar los parámetros de las peticiones http pero para esta ocasión se decide evitar ataques y brechas de seguridad cifrando las contraseñas con md5.

4.1.5. Identificación casos de uso.

En este apartado se recoge el conjunto de situaciones en las que puede encontrarse el usuario paciente y el usuario especialista mientras utiliza la aplicación. Para reflejarlo utilizo un actor que representa a ese usuario que interactúa con las distintas actividades.


Citas
especialista
Pacientes


Pedir cita
Ver citas
Historial
paciente
Cuadro
medico
Mis datos
Contacto

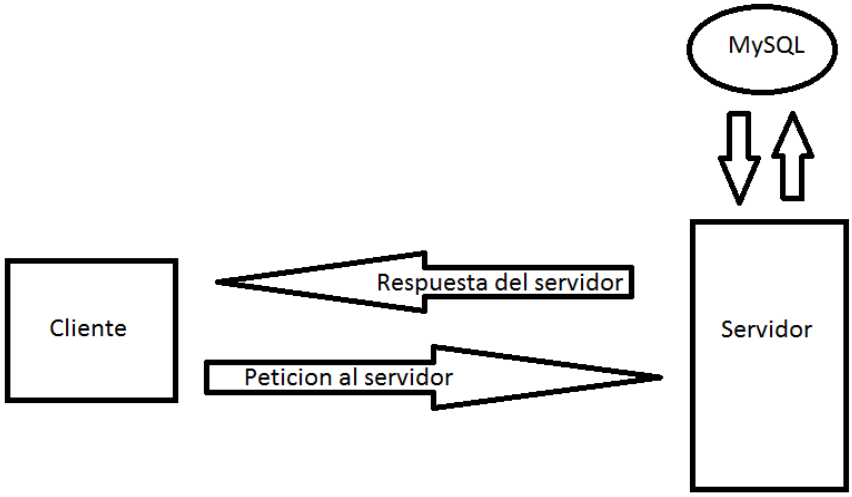
4.2. DISEÑO

El diseño de la aplicación ha ido sufriendo cambios a lo largo de la evolución por diferentes motivos como por ejemplo al adquirir más práctica y conocimientos he comprobado que se podía reducir la carga de trabajo y las tareas repetitivas y por otro lado, por organización y tiempos establecidos para la entrega he tenido que reducir algunas funcionalidades tanto en la parte del paciente como del profesional y decidir por lo necesario.

En esta fase he determinado cómo llevar a cabo la fase de análisis y decidir cómo construir el sistema, arquitectura, diseño de la base de datos, lenguaje utilizado, diseño de la interfaz... etc.

4.2.1 Arquitectura de la aplicación.

La arquitectura de la aplicación se basa en el modelo cliente-servidor. En el caso de este proyecto, la parte del cliente será el dispositivo móvil con la aplicación instalada, y la base de datos estará alojada en un servidor al que se accederá mediante peticiones http.



En mi aplicación es necesaria una base de datos centralizada que está ubicada en el servidor, y no local en el dispositivo móvil, por lo que es necesaria una separación entre cliente y servidor.

Esta base de datos de la aplicación es la que se ocupa de guardar todos los datos de los pacientes y especialista, de la mutuas, de los citas, de las fichas de citas de los pacientes. El acceso a esta base de datos es a través de internet por medio de peticiones http a través de una url que envía y recibe la información.

El cliente es quien inicia las solicitudes, teniendo un papel activo en la comunicación realizando una acción que conlleva el envío de una solicitud al servico web y la aplicación queda a espera la respuesta. El servicio web trata la petición y produce una respuesta que remite de vuelta a la aplicación cliente. Una vez obtenida la respuesta se muestra la información al cliente en la interfaz gráfica.

El usuario hace una petición de información al servidor que se empaqueta gracias a los métodos get o post. Una vez en el servidor esta información es procesada por el archivo php que lo que hace es leer la información que le llega y la contrasta con la información de la base de datos del servidor y este empaqueta la respuesta en formato Json que devuelve al cliente.

En resumen, el cliente interactúa con el usuario a través de una interfaz gráfica y el receptor de la solicitud, el servidor, espera a que lleguen solicitudes de los clientes, desempeñando un papel pasivo en la comunicación. Tras la recepción, procesa y envía los datos al cliente.

Las ventajas que supone dicha separación son notables, ventajas de tipo organizativo debidas a la centralización de la gestión de la información y separación de responsabilidades, lo que clarifica y facilita el sistema.

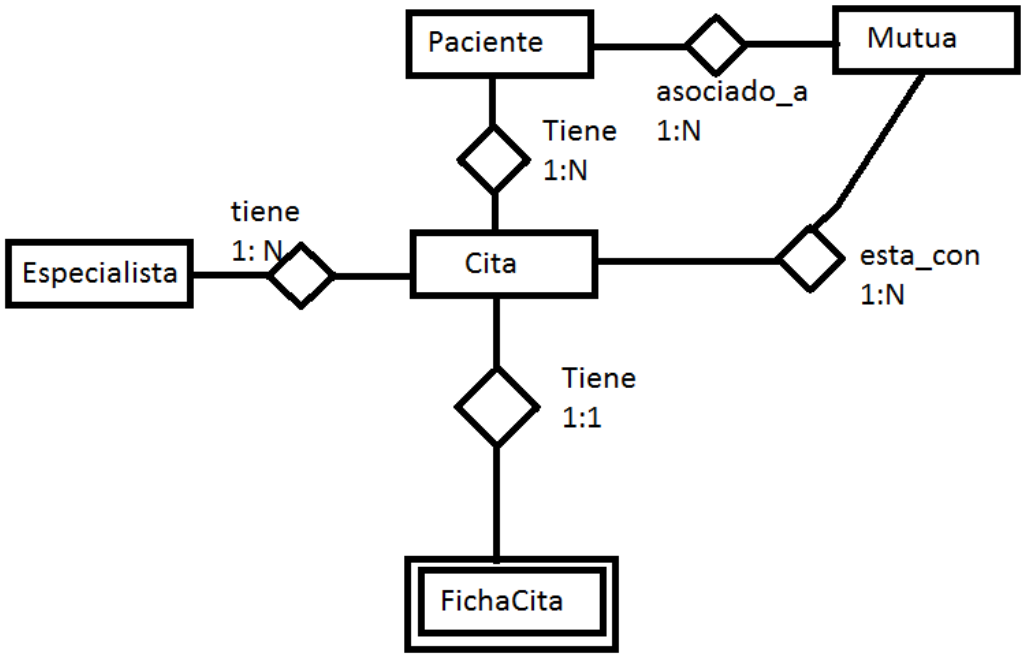
La escalabilidad y encapsulación son otras de las ventajas más destacables de este modelo, permitiendo aumentar funcionalidades o recursos de ambos por separado.

4.2.2 Base de datos

Una base de datos es una colección de información que se organiza de forma ordenada con distintos propósitos y usos.
Para poder almacenar toda la información es imprescindible un sistema de gestión de base de datos y en mi caso he elegido MySQL ya que es motor de bases de datos relacionales y además es un sistema de código abierto bastante extendido y viene incluido en el paquete WAMP que he comentando anteriormente.
La forma de interactuar con los datos almacenados en la base de datos es lo que se llama modelos de bases de datos, y pueden ser relaciones o no relacionales.
He elegido el modelo relacional que es un modelo organizativo y gestor de bases de datos que consiste en almacenar los datos en tablas compuestas por tuplas y campos y que permiten establecer relaciones entre datos que antes estarán almacenados en tablas y podrán conectar los datos entre ellas.
El lenguaje utilizado para las consultas estructurales de los datos de este tipo de bases relacionales es SQL. Este lenguaje de programación ayuda a solucionar problemas específicos o relacionados con la definición, manipulación e integridad de la información representada por los datos almacenados en las bases de datos permitiendo alcanzar una mayor eficiencia y productividad de desarrollo.

Para mi aplicación ha sido necesario crear una base de datos llamada "hospital" donde he definido cinco tablas diferentes: paciente, especialista, mutua, cita y ficha_cita.

En la siguiente imagen muestro un esquema de representación del Modelo E/R de mi base de datos:



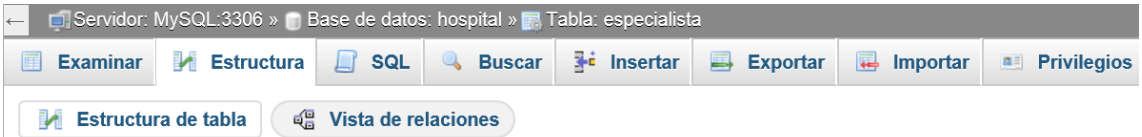
Presentación gráfica de mi tabla entidad-relación

En la siguiente imagen se muestra la estructura de mi base de datos hospital.

Las siguientes imágenes muestran la estructura de cada una de las tablas que forman la base de datos hospital:

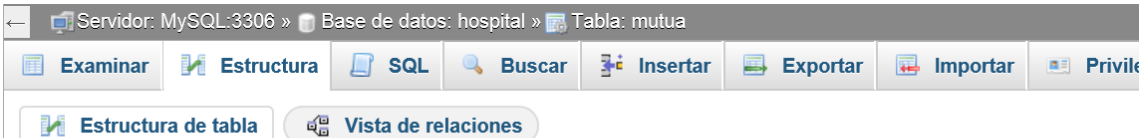
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
2	**nombre**	varchar(50)	utf8_general_ci		No	*Ninguna*		
3	**apellidos**	varchar (255)	utf8_general_ci		No	*Ninguna*		
4	**tipo_documento**	enum ('Pasaporte', 'DNI')	utf8_general_ci		No	*Ninguna*		
5	**valor_documento**	varchar(20)	utf8_general_ci		No	*Ninguna*		
6	**telefono**	varchar(9)	utf8_general_ci		No	*Ninguna*		
7	**email**	varchar (255)	utf8_general_ci		No	*Ninguna*		
8	**sociedad_medica**	int(11)			No	*Ninguna*		
9	**password**	text	utf8_general_ci		No	*Ninguna*		

Estructura tabla paciente



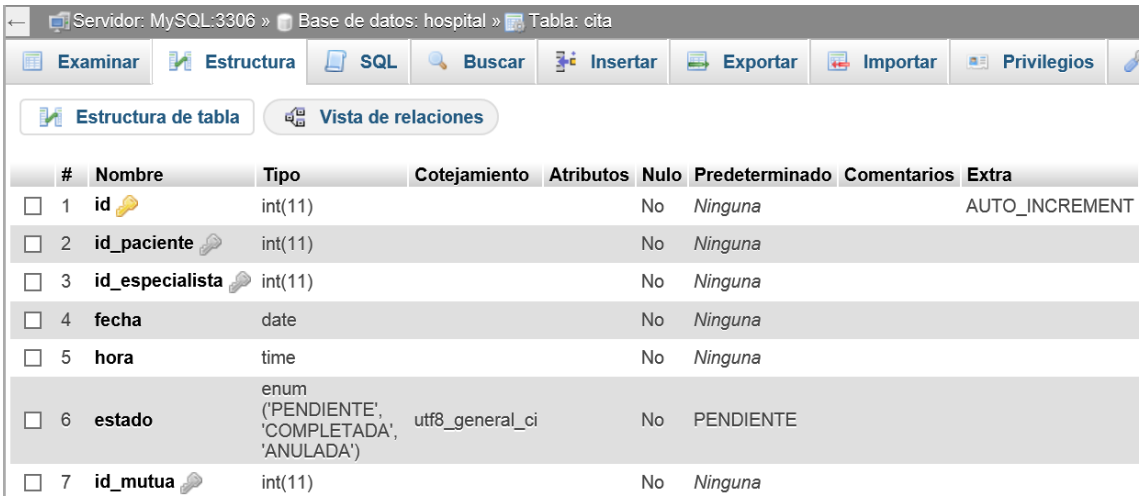
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
2	**nombre**	varchar(50)	utf8_general_ci		No	*Ninguna*		
3	**apellidos**	varchar(255)	utf8_general_ci		No	*Ninguna*		
4	**especialidad**	varchar(50)	utf8_general_ci		No	*Ninguna*		
5	**salario**	decimal(10,2)			No	*Ninguna*		
6	**tipo**	enum('MEDICO', 'ENFERMERO')	utf8_general_ci		No	*Ninguna*		
7	**dni**	varchar(9)	utf8_general_ci		No	*Ninguna*		
8	**password**	varchar(255)	utf8_general_ci		No	*Ninguna*		

Estructura tabla especialista



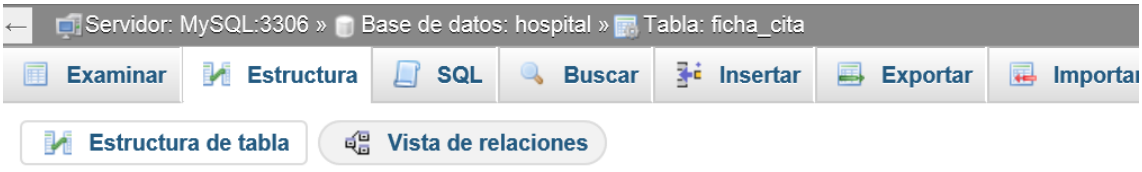
#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	**id**	int(11)			No	*Ninguna*		AUTO_INCREMENT
2	**nombre**	varchar(255)	utf8_general_ci		No	*Ninguna*		
3	**precio_cita**	decimal(10,2)			No	*Ninguna*		

Estructura tabla mutua



#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	id	int(11)			No	*Ninguna*		AUTO_INCREMENT
2	id_paciente	int(11)			No	*Ninguna*		
3	id_especialista	int(11)			No	*Ninguna*		
4	fecha	date			No	*Ninguna*		
5	hora	time			No	*Ninguna*		
6	estado	enum ('PENDIENTE', 'COMPLETADA', 'ANULADA')	utf8_general_ci		No	PENDIENTE		
7	id_mutua	int(11)			No	*Ninguna*		

Estructura tabla cita



#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
1	id_cita	int(11)			No	*Ninguna*		
2	observaciones	text	utf8_general_ci		No	*Ninguna*		
3	recomendaciones	text	utf8_general_ci		No	*Ninguna*		

Estructura tabla ficha_cita

Como se ha comentado anteriormente esta base de datos no puede ser local en el propio dispositivo, por lo que se implementa la base de datos externa y estará situada en el servidor y a la que se accede a través del API del servidor.
Es el servidor es el que se dedica a la parte de la lógica del proyecto y se encarga de toda la manipulación de datos.

Se ha implementado PHP como lenguaje de servidor que será el lenguaje utilizado para la comunicación de la aplicación con el servidor mediante ficheros php. Para ello he creado varios script PHP que estarán localizados en el servidor y a los que se accede mediante peticiones http. Estos ficheros recibirán una serie de parámetros y devolverán un código de retorno o una serie de datos serializados para su posterior tratamiento en la aplicación.

Para asegurar que se puede acceder a cualquier sistema se utiliza una API que es una especificación o mecanismo muy útil para conectar dos softwares entre sí para el intercambio de mensajes o datos en formato XML o JSON. La Api oculta o abstrae al usuario de cómo está hecha la base de datos.

Una Api (interfaz de programación de aplicaciones) es un conjunto de funciones y procedimientos que ofrece una biblioteca para que otro software la utilice como una

capa de abstracción, un espacio de acceso e intercambio de información adicional en la parte superior. Así una se sirve de la información de la otra sin dejar de ser independientes.

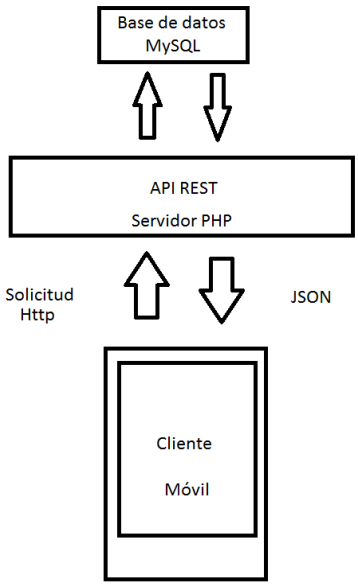
Cada API está diseñada en un lenguaje de programación concreto y con unas especificaciones distintas que la definen.
Existen diferentes tipos de APIs. Para este proyecto se utiliza una Api de servicios web, que son las interfaces de desarrollo de aplicaciones que permiten el intercambio de información entre un servidor web(software que da acceso a un servicio concreto a través de una URL) y una aplicación.

Normalmente este intercambio se produce a través de solicitudes o peticiones HTTP o HTTPS. En la petición de la aplicación y repuesta, tambien en HTTP del servicio web, se contiene información de todo tipo tanto en los metadatos de la cabecera como en los del mensaje, normalmente en dos tipos de formatos, XML o JSON.

Existen numerosas tecnologías relacionadas con el mundo de los servicios web, nosotros vimos dos, SOAP y REST, pero estudiamos y trabajamos con REST.

REST es el que utiliza el protocolo HTTP para comunicación entre el cliente/servidor, como es nuestro caso en el que el cliente que es la aplicación móvil hace la solicitud del recursos al servidor a través de una URI que sirve de identificador del recurso en la red. Los datos se transmiten en un formato específico identificado mediante nomenclatura MIME; se utilizan las acciones estandar del protocolo HTTP como GET, POST, etc; el servidor expone los recursos que podrán ser devueltos en distintos formatos que en este caso serán en formato JSON; y por cada solicitud que realizar el cliente se obtiene un código de respuesta que es característica propia de HTTP.

A continuación se muestra un gráfico que refleja lo expresado anteriormente:



4.2.3 Diseño de la interfaz

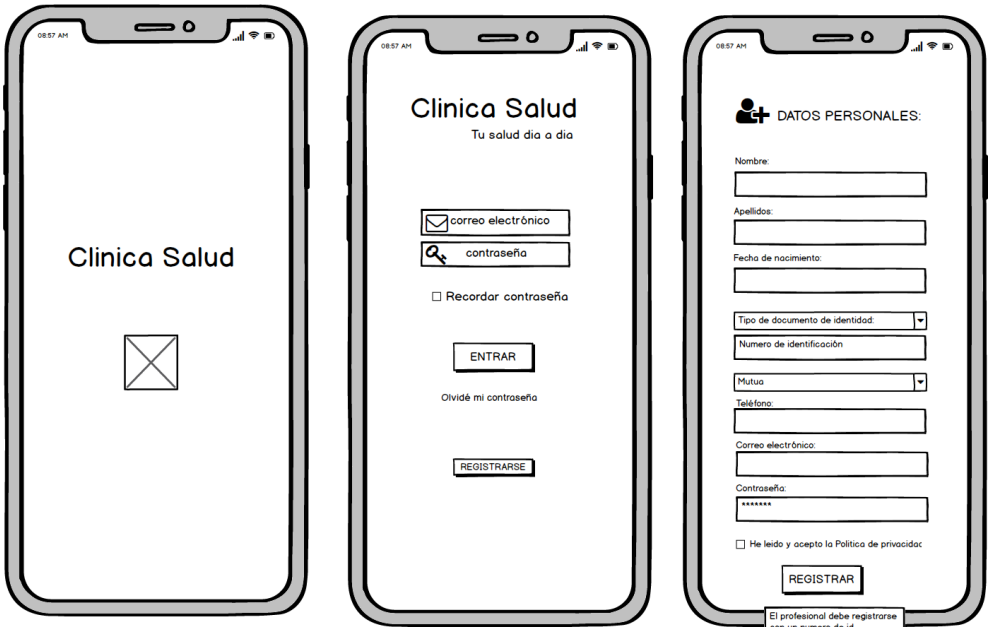
La interfaz será la encargada de interactuar directamente con el usuario . Para el diseño de la esta parte he utilizado la herramienta Balsamic MockUps que había sido recomendada por la tutora durante el curso y me ha resultado muy útil a la hora de diseñar los mockups puesto que es muy intuitiva y disponen del material suficiente para realizar las tareas deseadas.

Estos prototipos me han servido como guía a la hora de definir las interfaces dentro de la aplicación. Si bien es cierto que han existido muchas modificaciones pero se ha seguido el patrón.

He intentado hacer una interfaz gráfica siguiendo los patrones de usabilidad y accesibilidad estudiadas en desarrollo de interfaces.

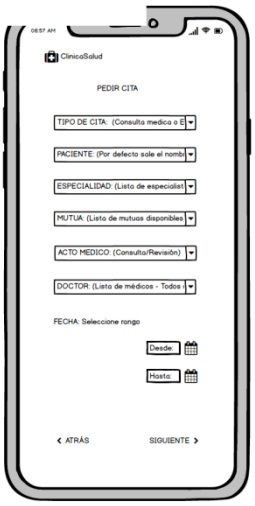
El desarrollo de la interfaz gráfica quizás sea una de las partes que más tiempo requiere para su creación. He tratado de hacer una interfaz sencilla, amigable y fácil de usar. He elegido colores claros usando la psicología del color, en este caso tonalidades verdes y amarillas que acompañan con la materia de la aplicación y botones redondeados para que no resultara demasiado agresiva al usuario.

4.2.3.1 Prediseño de los mockups

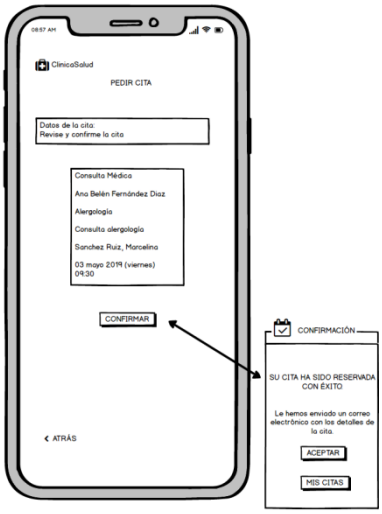


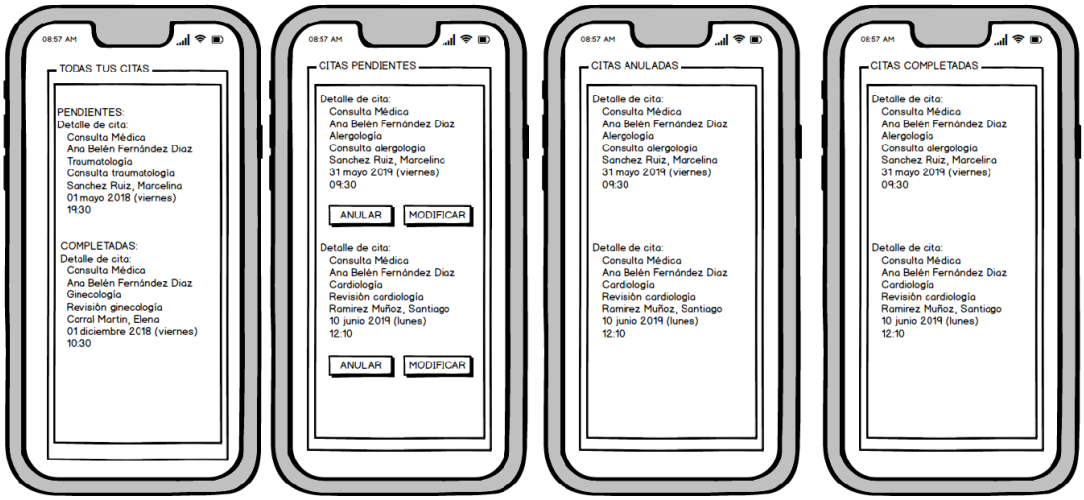

08:57 AM
ClinicaSalud
PEDIR CITA
MIS CITAS
MI HISTORIAL
ESPECIALISTAS
JUSTIFICANTE ASISTENCIA
MIS DATOS (MI FAMILIA)
CONTACTO

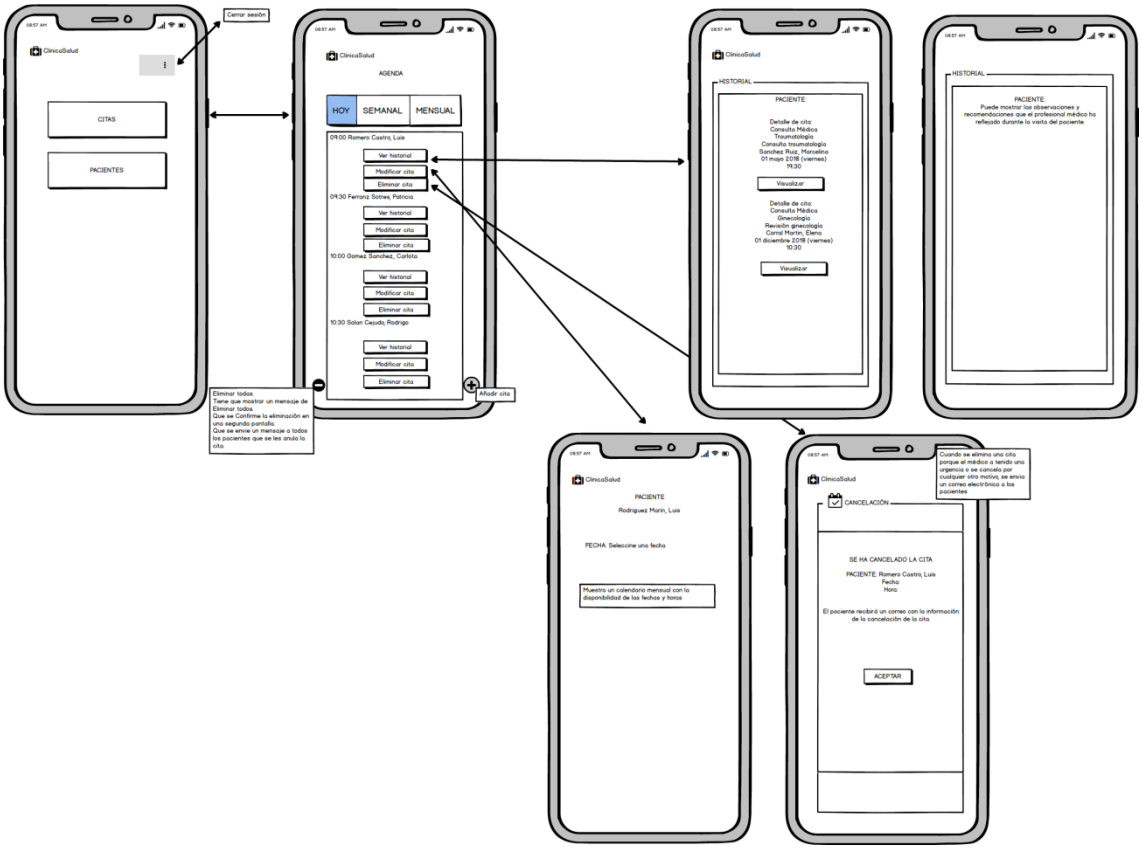

08:57 AM
ClinicaSalud
PEDIR CITA
MIS CITAS
MI HISTORIAL
ESPECIALISTAS
JUSTIFICANTE ASISTENCIA
MIS DATOS (MI FAMILIA)
CONTACTO


08:57 AM
ClinicaSalud
PEDIR CITA
TIPO DE CITA: (Consulta medica o E
PACIENTE: (Por defecto sale el nomb
ESPECIALIDAD: (Lista de especialist
MUTUA: (Lista de mutuas disponibles
ACTO MEDICO: (Consulta/Revisión)
DOCTOR: (Lista de médicos - Todos
FECHA: Seleccione rango
Desde:
Hasta:
ATRÁS
SIGUIENTE


08:57 AM
ClinicaSalud
PEDIR CITA
Estos son las citas disponibles:
Desde 03 de mayo hasta 06 junio
03 mayo 2019 (viernes)
09:50
Sanchez Ruiz, Marcelina
SELECCIONAR
15 mayo 2019 (martes)
12:30
Molina Molina, Pedro
SELECCIONAR
17 mayo 2019 (viernes)
10:20
Campero Gonzalez, Jorge
SELECCIONAR
21 mayo 2019 (martes)
11:40
Sanchez Ruiz, Marcelina
SELECCIONAR
24 mayo 2019 (viernes)
09:50
Sanchez Ruiz, Marcelina
SELECCIONAR
ATRÁS
SIGUIENTE


08:57 AM
ClinicaSalud
PEDIR CITA
Datos de la cita:
Revise y confirme la cita
Consulta Médica
Ana Belén Fernández Diaz
Alergología
Consulta alergología
Sanchez Ruiz, Marcelina
03 mayo 2019 (viernes)
09:30
CONFIRMAR
ATRÁS
CONFIRMACIÓN
SU CITA HA SIDO RESERVADA CON ÉXITO
Le hemos enviado un correo electrónico con los detalles de la cita.
ACEPTAR
MIS CITAS

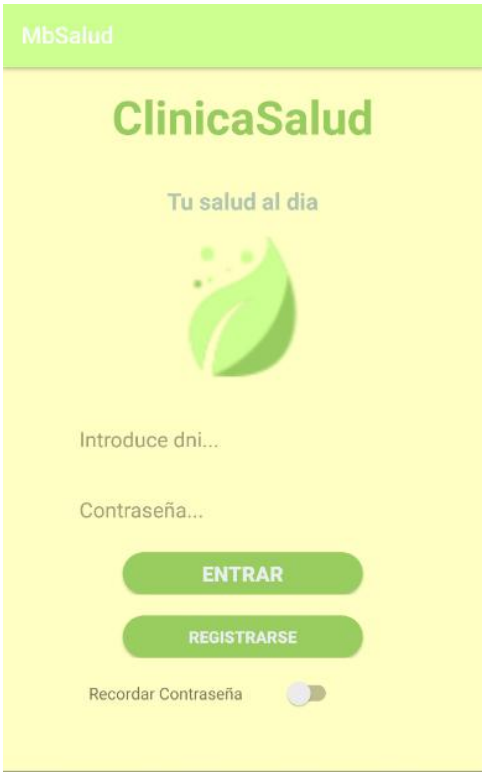

08:57 AM
TODAS TUS CITAS
PENDIENTES:
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Traumatología
Consulta traumatologia
Sanchez Ruiz, Marcelina
01 mayo 2018 (viernes)
19:30
COMPLETADAS:
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Ginecología
Revisión ginecologia
Corral Martin, Elena
01 diciembre 2018 (viernes)
10:30
CITAS PENDIENTES
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Alergología
Consulta alergologia
Sanchez Ruiz, Marcelina
31 mayo 2019 (viernes)
09:30
ANULAR
MODIFICAR
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Cardiología
Revisión cardiologia
Ramirez Muñoz, Santiago
10 junio 2019 (lunes)
12:10
ANULAR
MODIFICAR
CITAS ANULADAS
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Alergología
Consulta alergologia
Sanchez Ruiz, Marcelina
31 mayo 2019 (viernes)
09:30
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Cardiología
Revisión cardiologia
Ramirez Muñoz, Santiago
10 junio 2019 (lunes)
12:10
CITAS COMPLETADAS
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Alergología
Consulta alergologia
Sanchez Ruiz, Marcelina
31 mayo 2019 (viernes)
09:30
Detalle de cita:
Consulta Médica
Ana Belén Fernández Diaz
Cardiología
Revisión cardiologia
Ramirez Muñoz, Santiago
10 junio 2019 (lunes)
12:10


ClinicaSalud
Cerrar sesión
CITAS
PACIENTES
AGENDA
HOY
SEMANAL
MENSUAL
09:00 Romero Castro, Luis
Ver historial
Modificar cita
Eliminar cita
09:30 Ferranz Sotres, Patricia
10:00 Gomez Sanchez, Carlota
10:30 Solan Cejuda, Rodrigo
Añadir cita
Eliminar todas
Tiene que mostrar un mensaje de Eliminar todas
Que se Confirme la eliminación en una segunda pantalla.
Que se envie un mensaje a todos los pacientes que se les anula la cita.
HISTORIAL
PACIENTE
Visualizar
PACIENTE
Puede mostrar las observaciones y recomendaciones que el profesional médico ha reflejado durante la visita del paciente.
PACIENTE
Rodriguez Marin, Luis
FECHA: Seleccine una fecha
Muestra un calendario mensual con la disponibilidad de las fechas y horas
CANCELACIÓN
SE HA CANCELADO LA CITA
PACIENTE: Romero Castro, Luis
Fecha:
Hora:
El paciente recibirá un correo con la información de la cancelación de la cita.
ACEPTAR
Cuando se elimina una cita porque el médico a tenido una urgencia o se cancela por cualquier otro motivo, se envia un correo electrónico a los pacientes

4.2.3.2 Diseño final de las diferentes interfaces

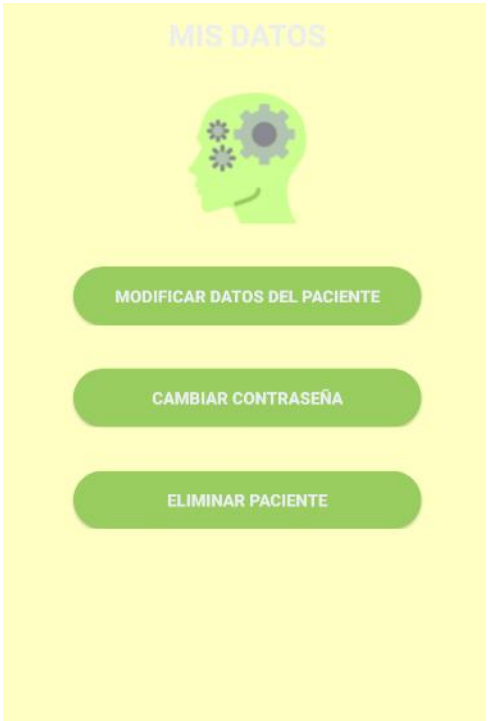
4.2.3.2.1 - Interfaces del paciente

Pantalla de login



Pantalla principal pacientes

Modificar perfil de usuario:

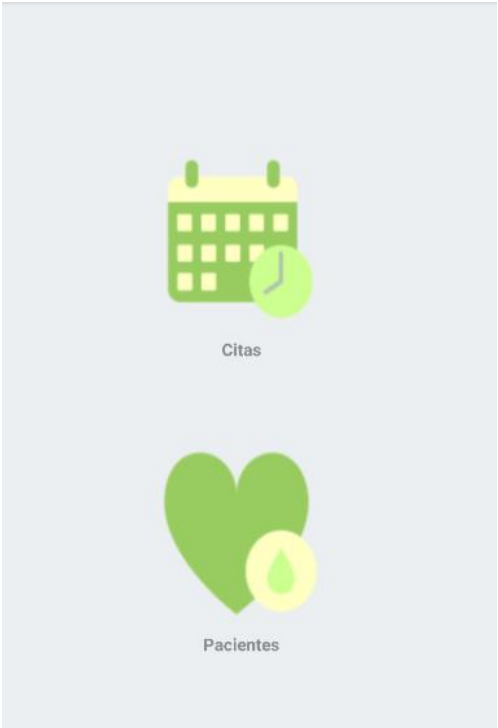


Datos de contacto del centro:

Consulta mis citas:

4.2.3.2.2 - Interfaces del especialista

Pantalla principal especialista:



Vista citas semanales del especialista:

Solicitar cita para un paciente desde la sesión del especialista:

4.3 IMPLEMENTACIÓN

En este punto se empieza a codificar algoritmos y estructuras de datos, definidos en las etapas anteriores, en el correspondiente lenguaje de programación para un determinado sistema gestor de bases de datos.

A la hora de realizar la implementación del desarrollo de un proyecto , es interesante realizar una estructura del código ordenada y que esté agrupado según diferentes características específicas.

Una adecuada estructura del código aporta muchas ventajas. Algunas de ellas son las siguientes:

- Permite una mejor comprensión del código y un aprendizaje más rápido de su manejo.
- Facilita el mantenimiento y actualización de la aplicación. Al dividirlo en partes, es

posible sustituir más rápidamente partes del programa que se quieran mejorar o corregir sin afectar al resto.

- Mejora el proceso de testeo del software al organizar el código en función de sus

características. Separando la parte visual de la aplicación de su funcionalidad es posible realizar pruebas automáticas que simulen el funcionamiento de la aplicación e identifiquen posibles errores.

- Reduce el tiempo de creación de software al poder reutilizarse el código entre aplicaciones según las necesidades que se tengan.

A continuación se muestra por separado la parte del cliente y del servidor con un resumen o parte más destacada de codificación que se ha realizado en cada parte.

4.3.1 CLIENTE

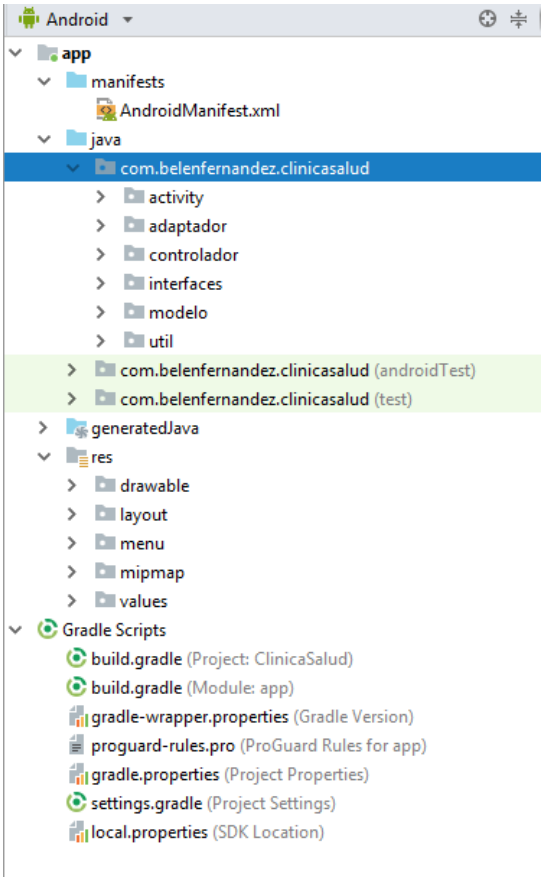
4.3.1.1 - Estructura del proyecto.

Siguiendo la dinámica de Android, una aplicación generalmente consiste en múltiples actividades vinculadas de forma flexible entre sí. En mi aplicación he creado una activity para cada pantalla, lo que supone una colección de activities como las que se muestran en la imagen.

Las activities están conformadas por dos partes: la parte lógica y la parte gráfica.

La parte lógica es una archivo .java que es la clase que se crea para poder manipular, interactuar y colocar el código de esa actividad y la parte gráfica es el layout que es el archivo.xml

La siguiente imagen muestra la estructura del proyecto:



La carpeta activity contiene todas las clases.java que contienen el código de funcionamiento de la activity.

La carpeta adaptador contiene las clases donde se han creado los tres adaptadores personalizados.

La carpeta controlador contiene las clases necesarias para la gestión de la comunicación.

Las carpeta modelo contiene las clases modelo.

La carpeta util contiene una clase para la conversión de fechas y horas entre otras cosas.

La carpeta rest contiene todos los ficheros de recursos necesarios para el proyecto que se distribuyen en entre las siguientes subcarpetas:

drawable: que contiene las imágenes usadas en la aplicación. Para definir diferentes recursos dependiendo de la resolución y la densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas, dependiendo del dispositivo en el que se vaya a ejecutar la aplicación. En esta carpeta tambien he incluido los botones y los gradientes que he creado.

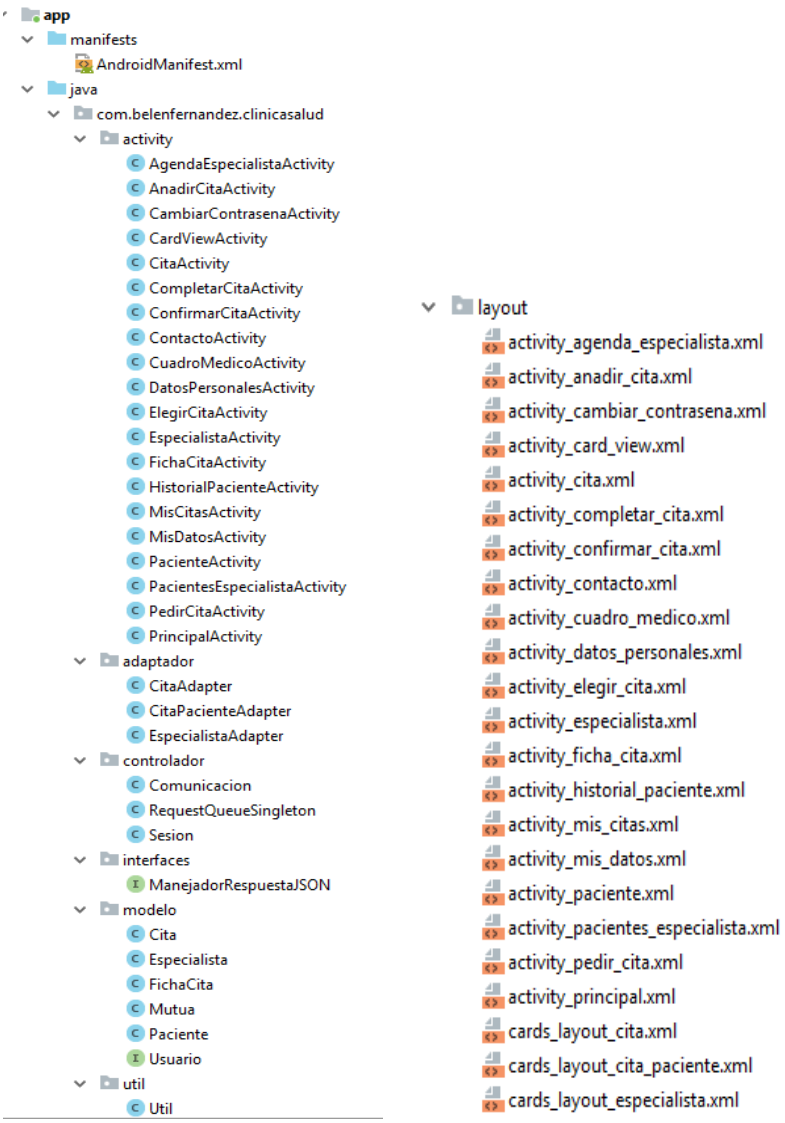
layout: Contine los ficheros de definicion de XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos layouts dependiendo de la orientación del dispositivo se pue de dividir en dos subcarpetas, dependiendo de como se muestre la aplicación, pero en este proyecto no ha sido necesario.

menu: contiene la definicion XML de los menús que se mostrarán en diferentes pantallas de la aplicación. Se han creado dos menus (uno para "Cerrar sesión" y "Añadir pacientes" y otro para "Cerrar sesión")

values: contine los ficheros XML de recursos de la aplicación, como cadenas de texto, estilos, colores y arrays de valores.

AndroidManifest.xml contiene la definición en XML de los aspectos principales de la aplicación, como su identificación(nombre, versión, icono..), sus componentes (pantallas, mensajes,...), las lbrerías auxiliares utilizadas, o los permisos necesarios para su ejecución.

A continuación se muestra de manera gráfica las diferentes activities y layots que se han creado en mi proyecto Android:



Activities y layouts creados en la implementación del cliente.

4.3.1.2.- Librería Volley

Volley es una librería desarrollada por Google para optimizar el envío de peticiones Http desde las aplicaciones Android hacia servidores externos.

Este componente actúa como una interfaz de alto nivel, liberando al programador de la administración de hilos y procesos tediosos de parsing, para permitir publicar fácilmente resultados en el hilo principal.

Volley está totalmente enfocado en las peticiones, evitando la creación de código repetitivo para manejar tareas asíncronas por cada petición o incluso para parsear los datos que vienen del flujo externo.

Entre las características más importantes de volley se puede destacar:

- Procesamiento concurrente de peticiones.

- Priorización de las peticiones, lo que permite definir la preponderancia de cada petición.
- Cancelación de peticiones, evitando la presentación de resultados no deseados en el hilo principal.
- Gestión automática de trabajos en segundo plano, dejando de lado la implementación manual de un framework de hilos.
- Implementación de caché en disco y memoria.
- Capacidad de personalización de las peticiones.
- Provee información detallada del estado y flujo de trabajo de las peticiones en la consola de depuración.

Volley posee varios componentes que optimizan la administración de las peticiones generadas desde las aplicaciones Android. La gestión comienza en una cola de peticiones que recibe cada una de las peticiones generadas, donde son previamente priorizadas para su realización.

Los pasos a seguir para el uso de la librería son los que siguen:

Es necesario añadir la librería Volley al proyecto Android Studio en el modulo build.gradle y añadir la siguiente dependencia.

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 28
    defaultConfig {
        applicationId "com.belenfernandez.clinicasalud"
        minSdkVersion 21
        targetSdkVersion 28
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    implementation 'com.android.volley:volley:1.1.1'
    implementation 'com.android.support:cardview-v7:28.0.0'
    implementation 'com.android.support:recyclerview-v7:28.0.+'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'
    implementation 'com.android.support:design:28.0.0'
}
```

Añadir el permiso para conexión en el Android Manifest. Para solicitar un permiso hay que declararlo en este fichero xml.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.belenfernandez.clinicasalud">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="MbSalud"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".activity.PrincipalActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
```

Para hacer una llamada con Volley se necesitan dos objetos: un objeto para declarar la petición (Request) y un objeto que procese las peticiones(RequesQueue)

Existen cuatro tipos de peticiones estándar para uso frecuente en Volley que reducen el tiempo a la hora de estructurar la petición, añadiendo hilos y generando procesos de parsing extenso :

1. StringRequest: Este es el tipo más común, ya que permite solicitar un recurso con formato de texto plano, como son los documentos HTML.
2. ImageRequest: Como su nombre lo indica, permite obtener un recurso gráfico alojado en un servidor externo.
3. JsonObjectRequest: Obtiene una respuesta de tipo JSONObject a partir de un recurso con este formato.
4. JsonArrayRequest: Obtiene como respuesta un objeto del tipo JSONArray a partir de un formato JSON.

En mi aplicación usaremos la petición JsonObjectRequest.

Cuando declaramos una petición usando un objeto Request le indicamos si la llamada es con el método GET o POST. En este caso hacemos el método POST. Le indicamos la url a la que queremos hacer la llamada y le pasamos el resto de parámetros.

También es necesario indicar un par de listener, uno que se ejecutará cuando la petición hay ido bien y otro que se ejecutará cuando surja algún error.

Se hace uso los métodos getInstance(), getRequestQueue().

El método getInstance() simplemente asigna memoria a la única instancia del singleton, donde se llama al constructor privado de la clase. Este método debe tener la propiedad synchronized, ya que la instancia será accedida desde varios hilos por lo que

es necesario evitar bloqueos de acceso. El método getRequestQueue() obtiene la instancia de la cola de peticiones que se usará a través de toda la aplicación.

Clase Comunicación que hace peticiones volley:

```
public class Comunicacion {

    public static void solicitudJSONObject(Context ctx, String url, JSONObject json, final ManejadorRespuestaJSON manejador, final int request)

        // solicitar
        JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST, url, json,
                (response) -> {
                        // aqui gestiono la respuesta correcta
                        Log.d( tag: "XXX", msg: "OK: " + response);
                        manejador.procesarRespuestaJSONObject(response, request);
                }, (error) -> {
                // aqui gestiono la respuesta si hubo error
                Log.d( tag: "XXX", msg: "Error: " + error.getMessage());
        });

        RequestQueueSingleton.getInstance(ctx).getRequestQueue().add(jsonObjectRequest);
    }
    public static void solicitudJSONArray(Context ctx, String url, JSONObject json, final ManejadorRespuestaJSON manejador, final int request) {
        // solicitar
        JsonObjectRequest jsonObjectRequest = new JsonObjectRequest(Request.Method.POST, url, json,
                (response) -> {
                        // aqui gestiono la respuesta correcta
                        try {
                            // La clave es que aqui se sabe que los datos de la respuesta son un jsonarray
                            // y no un jsonobject
                            // con esta linea se obtiene un jsonarray de un campo de jsonobject
                            JSONArray jsonArray = response.getJSONArray( name: "datos");
                            Log.d( tag: "XXX",  msg: "OK: " + response);
                            manejador.procesarRespuestaJSONOArray(jsonArray, request);
                        } catch (JSONException js) {

                        }
                }, (error) -> {
                // aqui gestiono la respuesta si hubo error
                Log.d( tag: "XXX", msg: "Error: " + error.getMessage());
        });

        RequestQueueSingleton.getInstance(ctx).getRequestQueue().add(jsonObjectRequest);
```

Esta clase en conjunción con la siguiente interface ManejadorRespuestaJSON devuelve la respuesta al método procesarRespuestaJsonObject o procesarRespuestaJSONArray que le hubiera hecho la llamada.

Interface ManejadorRespuestaJson:

Interface que deben impletar todas las activities que haga una llamada a volley.

```
public interface ManejadorRespuestaJSON {

    void procesarRespuestaJSONObject(JSONObject response, int request);
    void procesarRespuestaJSONOArray(JSONArray response, int request);

}
```

Las respuestas por parte de los servicios web son en Json. Esta clase Comunicacion.java se encarga de leer todas las respuestas y construir un objeto JSon mediante la función JSonObject o es capaz de construir un array de objetos Json a través de la función JSonArray.

Hay que tener en cuenta que cada activity que deseen manejar una respuesta del servidor deben implementar la interfaz ManejadorRespuestaJSON e implementar sus métodos, bien sea procesarRespuestaJSONObject o procesarRespuestaJSONArray.

Clase PacientesEspecialistaActivity.java donde se ve la implementación del ManejadorRespuestaJSON, con el método solicitarPacientes() que prepara la solicitud para el servidor y será esta misma activity(this) la que procesa la respuesta a través de procesarRespuestaJSONArray.

```
public class PacientesEspecialistaActivity extends AppCompatActivity implements ManejadorRespuestaJSON {

    public static final int REQ_LISTA_PACIENTES = 1;
    public static final int REQ_REGISTRO= 2;

    private Spinner spPacientes;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pacientes_especialista);

        spPacientes = (Spinner) findViewById(R.id.spPacientes);

        solicitarPacientes();
    }
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_pacientes_especialista_activity, menu);
        return true;
    }
```

```
private void solicitarPacientes() {
    String url = "http://192.168.1.36/hospital2/api.php";

    // Pedir al servidor (esto se haria realmente cuando se pulsara un boton, una opcion....)
    try {
        JSONObject json = new JSONObject();
        json.put( name: "operacion",  value: "lista_pacientes");
        JSONObject datos = new JSONObject();
        json.put( name: "datos", datos);

        // como he querido lista de pacientes, llamo a solicitud de array
        Comunicacion.solicitudJSONArray( ctx: this,url, json, manejador: this, REQ_LISTA_PACIENTES);
    } catch (JSONException js) {
        Log.d( tag: "XXX",js.getMessage());
    }
}
```

```
@Override
public void procesarRespuestaJSONOArray(JSONArray response, int request) {
    if(request==REQ_LISTA_PACIENTES) {
        List<Paciente> pacientes = new ArrayList<>();

        for(int i=0;i<response.length();i++) {
            try {
                JSONObject o = response.getJSONObject(i);
                // Para cada objeto del jsonarray, lo convierto en un Paciente y lo anado al arraylist
                // de pacientes
                pacientes.add(Paciente.pacienteFromJSON(o));
            } catch (JSONException js) {
                Log.d( tag: "XXX",js.getMessage());
            }
        }
        ArrayAdapter<Paciente> adapter = new ArrayAdapter<>( context: this,android.R.layout.simple_list_item_1,pacientes);
        this.spPacientes.setAdapter(adapter);
    }
}
```

Una de las características adicionales de Volley es la creación del patrón Singleton para que tenga un alcance global dentro de la aplicación.

Normalmente una aplicación basada en servicios web necesita realizar peticiones a lo largo de todas sus actividades y componentes. Esta situación haría que se declararan colas de peticiones por todos lados o incluso pasar como parámetro la cola entre clases, lo cual llega a ser repetitivo, poco eficiente y confuso.

Para desmontar este complejo enfoque, Google recomienda crear un Patrón Singleton que encapsula las funcionalidades necesarias de Volley.

Este patrón se caracteriza por limitar el alcance de la clase a un solo objeto, es decir, solo puede existir un solo objeto controlador que represente la existencia de la clase, restringiendo la instanciación de nuevos elementos.

Esto hace que singleton será presente en todo el proyecto Android y se podrán usar las funcionalidades en cualquier lugar. Esta clase debe contener como atributo la cola de peticiones y el contexto de la aplicación.

Clase RequestQueuSingleton:

```
import ...

public class RequestQueueSingleton {
    private static RequestQueueSingleton requestQueueSingleton;

    private RequestQueue requestQueue;
    private static Context context;

    private RequestQueueSingleton(Context ctx) {
        context = ctx;
        requestQueue = getRequestQueue();
    }
    public static synchronized RequestQueueSingleton getInstance(Context context) {
        if (requestQueueSingleton == null) {
            requestQueueSingleton = new RequestQueueSingleton(context);
        }
        return requestQueueSingleton;
    }

    public RequestQueue getRequestQueue() {
        if (requestQueue == null) {
            requestQueue = Volley.newRequestQueue(context.getApplicationContext());
        }
        return requestQueue;
    }
}
```

4.3.1.3.- Uso de Adaptadores, ReciclerView y CardView

Un adaptador (Adapter) es un mecanismo de Android que hace de puente entre nuestros datos y las vistas contenidas en un ListView. Es un objeto View que se puede añadir a cualquier activity igual que cualquier otro complemento de la interfaz de usuario.

Es un objeto de una clase que implementa la interfaz Adapter. Este actúa como un enlace entre un conjunto de datos y un adaptador vista, un objeto de una clase que extiende a la clase abstracta AdapterView.

El conjunto de datos de la lista puede ser de cualquier cosa que presente datos de manera estructurada. Un adaptador es responsable de recuperar datos desde un conjunto de datos y de generar objetos View mediante esos datos de manera muy eficiente sin generar ningún retraso notable.

Los adaptadores solo reproducen los objetos que están listos en pantalla o los que están a punto de moverse, de modo que la memoria consumida por el adaptador vista puede ser constante e independiente del tamaño del conjunto de datos a mostrar.

Android dispone de sus propios adaptadores pero también es posible crear adaptadores personalizados como es en el caso de este proyecto para el que se han creado adaptadores customizados.

El ejemplo que se muestra a continuación es un adaptador que permite mostrar una lista con todas las citas de un paciente.

Para obtener este resultado se han usado widgets nuevos que he puesto en práctica en el desarrollo de esta aplicación.

Se trata de RecyclerView que es una lista que me permite mostrar información en una colección de vistas de una forma eficiente y potente y los CardViews que son tarjetas que permiten agrupar información de manera personalizada y mostrarla de una forma más elegante.

Ambos widgets son usados en conjunto, el RecyclerView muestra una lista de CardViews los cuales se van replicando de acuerdo a la cantidad de datos que hay en la colección que se desea mostrar.

Para poder usarlos es necesario incorporarlos en las dependencias de compatibilidad del archivo build.gradle del proyecto.

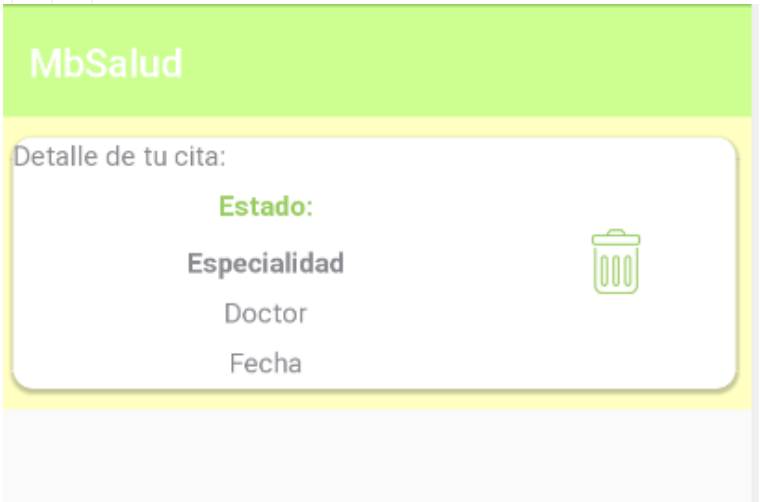
```
implementation 'com.android.support:cardview-v7:28.0.0'
implementation 'com.android.support:recyclerview-v7:28.0.+'
```

Las siguientes imágenes muestran como se ha usado el adaptador con el RecyclerView y el CardView.

Creo el layout para el cardview:

activity card_layout_cita_paciente.xml

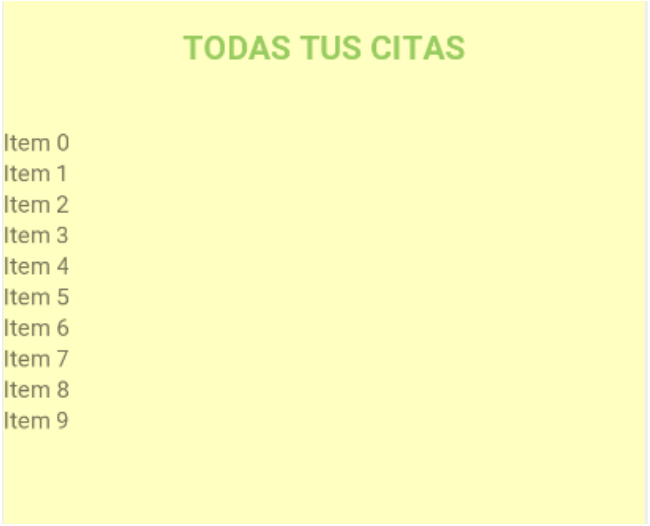
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/gradient">

    <android.support.v7.widget.CardView
        xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        card_view:cardCornerRadius="10dp"
        card_view:cardElevation="5dp"
        card_view:cardUseCompatPadding="true">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal"
            android:weightSum="100">

            <LinearLayout
                android:layout_width="0dip"
                android:layout_height="match_parent"
                android:layout_weight="70"
                android:orientation="vertical"
                android:weightSum="100">

                <TextView
                    android:id="@+id/tvDetalle"
                    android:layout_width="match_parent"
                    android:layout_height="0dip"
                    android:layout_gravity="left|center_vertical"
                    android:layout_weight="20"
                    android:text="Detalle de tu cita:"
                    android:textAppearance="?android:attr/textAppearanceSmall"
                    android:textColor="@color/colorTexto2" />
```



Creo el layout para la lista que contiene los datos que se desean mostrar:

activity_mis_citas.xml

```
    <TextView

        android:layout_width="0dip"
        android:layout_height="match_parent"
        android:layout_weight="90"
        android:gravity="center|center_horizontal"
        android:text="TODAS TUS CITAS"
        android:textColor="@color/colorPrimaryDark"
        android:textSize="20sp"
        android:textStyle="bold" />
</LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="2"></LinearLayout>

    <android.support.v7.widget.RecyclerView
        android:id="@+id/rvCitasPaciente"
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="40"
        android:scrollbars="vertical" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dip"
        android:layout_weight="5"></LinearLayout>

</LinearLayout>
```



Creo el adaptador personalizado:

El Adapter debe extender de RecyclerView.Apadater para que use una instancia de RecyclerView.ViewHolder.

El método onBindViewHolder se invoca para cada objeto de la lista y genera una vista grafica para cada uno de ellos.

CitaPacienteAdapter.java

```
public class CitaPacienteAdapter extends RecyclerView.Adapter<CitaPacienteAdapter.CitaPacienteViewHolder> {

    private MisCitasActivity citasActivity;
    private HistorialPacienteActivity historialPacienteActivity;
    private AppCompatActivity context;
    private ArrayList<Cita> citas;

    public CitaPacienteAdapter(MisCitasActivity citasActivity, HistorialPacienteActivity historialPacienteActivity, ArrayList<Cita> citas) {
        this.citasActivity = citasActivity;
        this.historialPacienteActivity = historialPacienteActivity;
        this.context = this.citasActivity!=null ? citasActivity : historialPacienteActivity;
        this.citas = citas;
    }

    public static class CitaPacienteViewHolder extends RecyclerView.ViewHolder {

        //adaptar
        TextView tvEstado, tvEspecialidad, tvDoctor, tvFecha;
        ImageView ivCancelar;

        public CitaPacienteViewHolder(View itemView) {
            super(itemView);
            this.tvEstado = (TextView) itemView.findViewById(R.id.tvEstado);
            this.tvEspecialidad = (TextView) itemView.findViewById(R.id.tvEspecialidad);
            this.tvDoctor = (TextView) itemView.findViewById(R.id.tvDoctor);
            this.tvFecha = (TextView) itemView.findViewById(R.id.tvFecha);
            this.ivCancelar = (ImageView) itemView.findViewById(R.id.ivCancelar);
        }
    }
```

```
@Override
public void onBindViewHolder(@NonNull CitaPacienteAdapter.CitaPacienteViewHolder holder, final int position) {
    String fecha = Util.dateToString(citas.get(position).getFecha());
    String especialidad = citas.get(position).getEspecialista().getEspecialidad();
    String doctor = citas.get(position).getEspecialista().getNombre() + " " + citas.get(position).getEspecialista().getApellidos();
    String estado = citas.get(position).getEstado();

    holder.tvEspecialidad.setText(especialidad);
    holder.tvDoctor.setText(doctor);
    holder.tvFecha.setText(fecha);
    holder.tvEstado.setText(estado);

    Log.d( tag: "XXX", msg: "Posicion: " + position + " Estado: " + estado);

    Drawable imgAnular = context.getDrawable(R.drawable.cancelar);
    Drawable imgVer = context.getDrawable(R.drawable.ver);
    if(estado.equalsIgnoreCase( anotherString: "COMPLETADA")) {
        Log.d( tag: "XXX", msg: "Posicion: " + position + " Estado: " + estado + " y pongo imagen completada");
        holder.ivCancelar.setImageDrawable(imgVer);
        holder.ivCancelar.setOnClickListener((v) -> {
                // se invoca la accion de eliminar cita...
                verInformeCita(citas.get(position).getFicha());
        });
    }
    else if(estado.equalsIgnoreCase( anotherString: "PENDIENTE")){
        Log.d( tag: "XXX", msg: "Posicion: " + position + " Estado: " + estado + " y pongo imagen cancelar");

        holder.ivCancelar.setImageDrawable(imgAnular);
        holder.ivCancelar.setOnClickListener((v) -> {
                // se invoca la accion de eliminar cita...
                anularCita(citas.get(position).getId());
        });
    }
    else {
        Log.d( tag: "XXX", msg: "Posicion: " + position + " Estado: " + estado + " y anulo");
        holder.ivCancelar.setVisibility(View.INVISIBLE);
    }
```

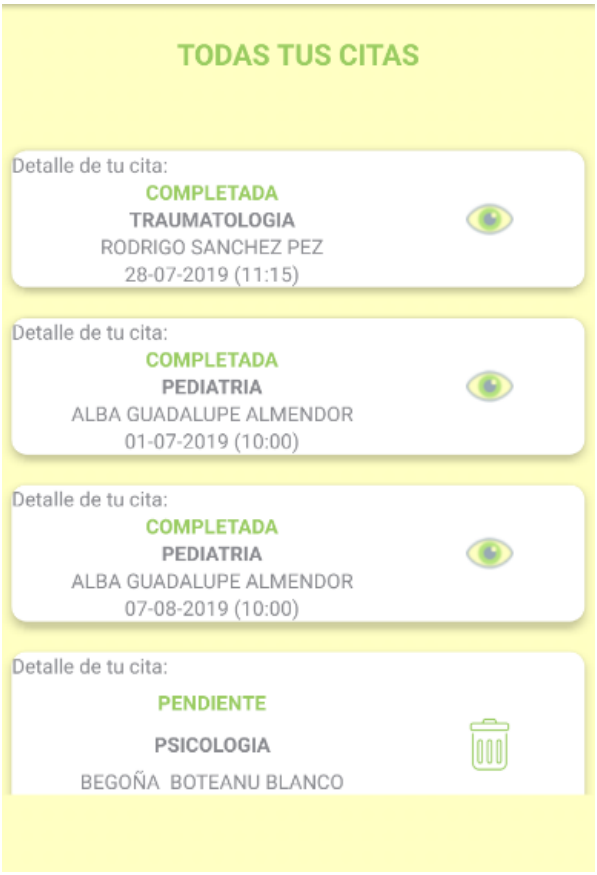
La clase MisCitasActivity es quien va a hacer uso del adaptador que será el que muestre la respuesta del servidor:

MisCitasActivity.java

```
public class MisCitasActivity extends AppCompatActivity implements ManejadorRespuestaJSON {

    public static final int REQ_LISTA_CITAS = 1;
    public static final int REQ_ANULAR_CITA = 2;

    private static RecyclerView.Adapter adapter;
    private RecyclerView.LayoutManager layoutManager;
    private static RecyclerView recyclerView;
    private static ArrayList<Cita> citas;

    private Paciente paciente;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_mis_citas);

        this.paciente = (Paciente) Sesion.getInstance().getUsuario();

        // obtener referencia a RecyclerView grafico
        recyclerView = (RecyclerView) findViewById(R.id.rvCitasPaciente);
        recyclerView.setHasFixedSize(true);

        // Poner una animacion estandar para cada vez que se muestre un card
        layoutManager = new LinearLayoutManager( context: this);
        recyclerView.setLayoutManager(layoutManager);
        recyclerView.setItemAnimator(new DefaultItemAnimator());

        solicitarMisCitas();
    }


    private void solicitarMisCitas() {
        String url = getString(R.string.url_servidor);

        // Pedir al servidor (esto se haria realmente cuando se pulsara un boton, una opcion....)
        try {
            JSONObject json = new JSONObject();
            json.put( name: "operacion",  value: "citas_paciente");
            JSONObject datos = new JSONObject();
            datos.put( name: "id_paciente",this.paciente.getId());
            datos.put( name: "estado", value: "TODAS");
            json.put( name: "datos", datos);

            // como he querido lista de especialistas, llamo a solicitud de array
            Comunicacion.solicitudJSONArray( ctx: this,url, json, manejador: this, REQ_LISTA_CITAS);
        } catch (JSONException js) {
            Log.d( tag: "XXX",js.getMessage());
        }
    }
```

```java
@Override
public void procesarRespuestaJSONOArray(JSONArray response, int request) {
    if(request==REQ_LISTA_CITAS) {
        citas = new ArrayList<>();

        for(int i=0;i<response.length();i++) {
            try {
                JSONObject o = response.getJSONObject(i);
                // Para cada objeto del jsonarray, lo convierto en un especialista y lo anado al arraylist
                // de especialistas
                citas.add(Cita.citaFromJSON(o));
            } catch (JSONException js) {
                Log.d( tag: "XXX",js.getMessage());
            }
        }

        Log.d( tag: "XXX", msg: "Lista de citas: " + citas.size());
        // con el arraylist formado, creo el adaptador y lo asocio al recyclerview
        adapter = new CitaPacienteAdapter( citasActivity: this,  historialPacienteActivity: null, citas);
        recyclerView.setAdapter(adapter);
    }
}
```

Resultado del adaptador y el cardView:



4.3.2 SERVIDOR

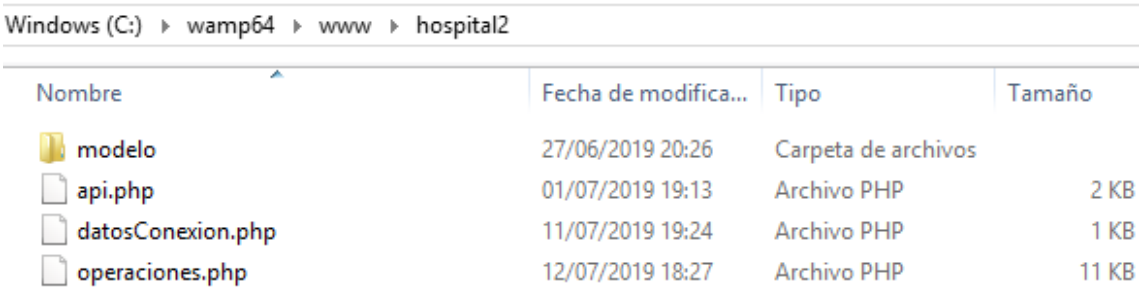
PHP es el lenguaje utilizado del lado del servidor. PHP es el acrónimo de Hipertext Preprocesor. Se trata de uno de los lenguajes de programación del lado del servidor más utilizados actualmente. Es gratuito e independiente y muy rápido. Se creó a mediados de los 90 y desde entonces, debido a que es un lenguaje de código abierto, ha recibido cientos de contribuciones por parte de desarrolladores de todo el mundo.

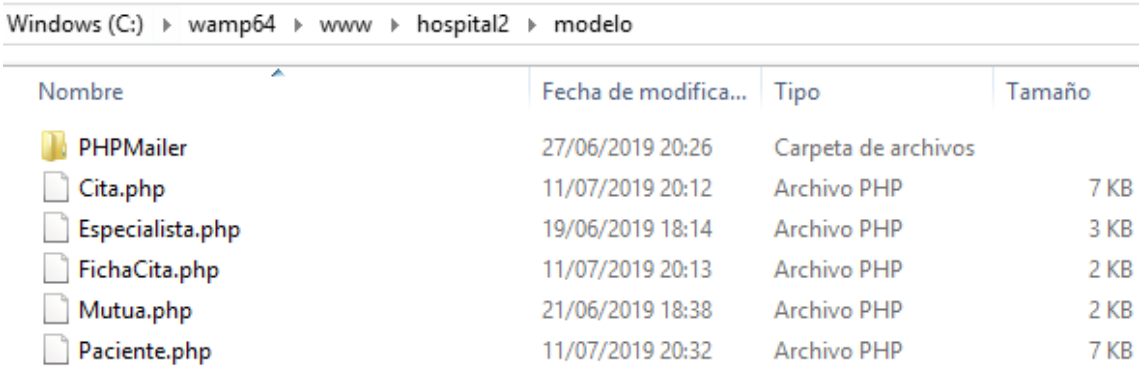
Una de las principales ventajas que ofrece es que es muy fácil de aprender; además, es multiplataforma. Incorpora una gran selección de funciones y ofrece una gran facilidad para establecer conexión con todo tipo de bases de datos, como MySQL.

Respecto a las desventajas, en PHP todo el trabajo recae sobre el servidor; no delega ni una sola tarea sobre el lado del cliente.

A continuación se muestran los distintos archivos .php que se han creado en el lado del servidor.

Hay que recordar que en este proyecto la base de datos hospital está ubicada en el servidor.





```php
define("SERVERNAME", "localhost");
define("USER", "root");
define("PASSWORD", "");
define("DBNAME", "hospital");

function abrir_conexion() {
    $enlace =  new mysqli(SERVERNAME, USER, PASSWORD, DBNAME);
    if ($enlace->connect_error) {
        return null;
    }
    else {
        $enlace->set_charset("utf8");
        return $enlace;
    }
}

function cerrar_conexion($enlace) {
    if($enlace)
        $enlace->close();
}

?>
```

Archivo datosConexión que hace la conexión con la base de datos hospital.

```php
require("operaciones.php");


// Leer peticion json del cliente
$json = file_get_contents('php://input');

// Convertirlo a objeto PHP para leerlo mas facilmente
$data = json_decode($json);


if($data->operacion == 'login') {
    echo login($data->datos);
}
else if($data->operacion == 'registro_paciente') {
    echo registro_paciente($data->datos);
}
else if($data->operacion == 'editar_paciente') {
    echo editar_paciente($data->datos);
}
else if($data->operacion == 'eliminar_paciente') {
    echo eliminar_paciente($data->datos);
}
else if($data->operacion == 'modificar_contrasena') {
    echo modificar_contrasena($data->datos);
}
else if($data->operacion == 'citas_paciente') {
    echo citas_paciente($data->datos);
}
else if($data->operacion == 'citas_especialista') {
    echo citas_especialista($data->datos);
}
else if($data->operacion == 'lista_especialistas') {
    echo lista_especialistas(null);;
```

api.php que es la interfaz de comunicación principal que recibe la petición del cliente y en función de lo que tenga el campo operación llama a una función u otra del módulo operaciones.php al que le pasa el campo datos.

```php
function login($datos) {

    $respuesta = [];
    $respuesta["estado"] = "ERR";
    $respuesta["mensaje"] = "mensaje";
    $respuesta["datos"]["tipo"] = "";
    $respuesta["datos"]["usuario"] = "";

    $paciente = Paciente::buscar_dni($datos->dni);
    $especialista = Especialista::buscar_dni($datos->dni);

    $usuario = ($paciente!=null) ? $paciente : $especialista;

    $md5_pass = md5($datos->password);

    if($usuario->password==$md5_pass) {
        $respuesta["estado"] = "OK";
        if($usuario instanceof Paciente)
            $respuesta["datos"]["tipo"] = "paciente";
        else
            $respuesta["datos"]["tipo"] = "especialista";

        $respuesta["datos"]["usuario"] = $usuario;
    }
    else {
        // no encontro, login incorrecto
        $respuesta["mensaje"] = "DNI o password invalidos";
    }

    return json_encode($respuesta);
}
```

operaciones.php llama a cada modelo que toque en cada acción.

```php
class Especialista {

    public  $id;
    public  $nombre;
    public  $apellidos;
    public  $especialidad;
    public  $salario;
    public  $tipo;
    public  $dni;
    public  $password;

    function __construct($id, $nombre, $apellidos, $especialidad, $salario, $tipo, $dni, $password) {
       $this->id = $id;
       $this->nombre = $nombre;
       $this->apellidos = $apellidos;
       $this->especialidad = $especialidad;
       $this->salario = $salario;
       $this->tipo = $tipo;
       $this->dni = $dni;
       $this->password = $password;
    }

    public static function buscar_id($id) {
        $obj = null;
        $sql = "SELECT * from especialista WHERE id='" . $id . "'";
        $conn = abrir_conexion();
        if(!$conn)
            return null;

        $result = $conn->query($sql);

        if(!empty($result) && $result->num_rows==1) {
            $row = $result->fetch_assoc();
            $obj = Especialista::from_row($row);
```

especialista.php.

Cada uno de estos archivo de la carpeta modelo es llamado por operaciones.

4.4. -PRUEBAS DE VALIDACION Y VERIFICACIÓN

4.4.1.-Pruebas de servidor.

Para hacer estas pruebas he usado Advanced REST client o ARC.

Acción	**Login al sistema mediante dni y contraseña**	
URL	*/hospital/api.php*	
Método	POST	
Formato de entrada	JSON	
Formato de respuesta	JSON	
Campos de la entrada	*operacion*	String
	datos	Objeto JSON
	datos.dni	String
	datos.password	String
Objeto JSON respuesta	*estado*	String
	mensaje	String
	datos	Objeto JSON
	datos.tipo	String
	datos.usuario	Objeto JSON Paciente
Ejemplo de entrada	{ "operacion": "login", "datos": { "dni": "00000003H", "password": "3333" } }	
Ejemplo salida correcta	{ "estado": "OK", "mensaje": "mensaje", "datos": { "tipo": "paciente", "usuario": { "id": "1", "nombre": "Juan", "apellidos": "Sanchez Peralta", "tipo_documento": "DNI", "valor_documento": "00000003H", "telefono": "918726252", "email": "mi@email.es", "sociedad_medica": { "id": "1", "nombre": "ADESLAS", "precio_cita": "5.00" }, "password": "3333" } } }	
Ejemplo de salida error	{ "estado": "ERR",	

	"mensaje": "DNI o password invalidos",
	"datos": {
	"tipo": "",
	"usuario": ""
	}
	}

Acción	**Confirmar un cita de un paciente**	
URL	*/hospital/api.php*	
Método	POST	
Formato de entrada	JSON	
Formato de respuesta	JSON	
Campos de la entrada	*operacion*	String
	datos	Objeto JSON
	datos.id_especialista	Int
	datos.id_paciente	Int
	datos.fecha	date
	datos.hora	time
	datos.id	Int
Objeto JSON respuesta	*estado*	String
	mensaje	String
	datos	Objeto JSON
	datos.id	Int
	datos.paciente	Objeto JSON Paciente
	datos.especialista	Objeto JSON Especialista
	datos.fecha	Date
	datos.hora	time
	datos.estado	String
	datos.mutua	Objeto JSON Mutua
	datos.ficha	Objeto JSON ficha

Ejemplo de entrada	{
	"operacion": "confirmar_cita",
	"datos": {
	"id_especialista":9,
	"id_paciente":1,
	"fecha": "2019-07-17",
	"hora" : "09:30",
	"id_mutua":1
	}
	}	

Ejemplo salida correcta	{
	"estado": "OK",
	"mensaje": "Cita realizada",
	"datos": {
	"id": "37",
	"paciente": {	

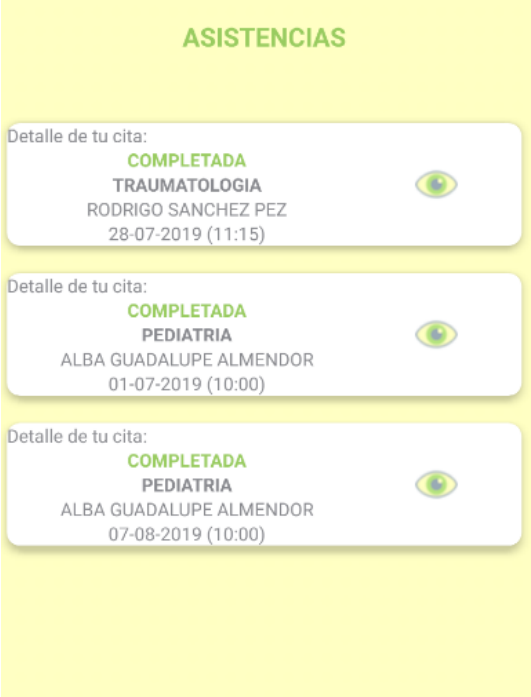
	"id": "1",
	"nombre": "Juan",
	"apellidos": "Sanchez Peralta",
	"tipo_documento": "DNI",
	"valor_documento": "00000003H",
	"telefono": "918726252",
	"email": "mAi@email.es",
	"sociedad_medica": {
	"id": "1",
	"nombre": "ADESLAS",
	"precio_cita": "5.00"
	},
	"password":
	"934b535800b1cba8f96a5d72f72f1611"
	},
	"especialista": {
	"id": "9",
	"nombre": "ALBA",
	"apellidos": "GUADALUPE ALMENDOR",
	"especialidad": "PEDIATRIA",
	"salario": "1900.00",
	"tipo": "MEDICO",
	"dni": "33333333A",
	"password":
	"b59c67bf196a4758191e42f76670ceba"
	},
	"fecha": "2019-07-17",
	"hora": "09:30:00",
	"estado": "PENDIENTE",
	"mutua": {
	"id": "1",
	"nombre": "ADESLAS",
	"precio_cita": "5.00"
	},
	"ficha": {
	"id_cita": "37",
	"observaciones": "",
	"recomendaciones": ""
	}
	}

Ejemplo de salida error	{
	"estado": "ERR",
	"mensaje": "Error en la cita",
	"datos": ""
	}

4.4.2.- Pruebas de cliente.

Estas pruebas las he hecho directamente con cada acción en la aplicación Android.

Resultado al pulsar el icono de historial



4.5.- MANTENIMIENTO

En la fase del proceso de identificación de las necesidades a cubrir por la aplicación y a lo largo del desarrollo de la misma, surgieron un gran abanico de posibilidades que se podrían presentar a los usuarios y se barajaron varias posibilidades que por diferentes motivos no han podido llevarse a cabo. Algunas de estas ideas han sido descartadas y no llegaron a realizarse pero otras muchas podrían incorporarse en futuras actualizaciones. Estas mejoras, novedades o nuevas versiones son las siguientes:

Añadir un splash.

Incluir un botón para que el usuario pudiese descargar sus justificantes de asistencia médica.

Incluir un botón para que el usuario pudiese descargar los informes médicos y pruebas realizadas en el centro.

Posibilidad de solicitar cita para pruebas de diagnostico (Ecografía, RayosX, mamografías...).

Incluir un breve y fácil manual de uso de la aplicación.

Posibilidad de informar que has llegado al centro e informar de ello desde la app y recibir información de la sala donde se tine la cita.

Información sobre posibles cancelaciones, demoras o ausencias del médico especialista habitual que no está en la consulta porque ha tenido una urgencia o imprevisto.

5. CONCLUSIONES FINALES DEL PROYECTO

Hasta llegar aquí ha sido un objetivo bastante laborioso y difícil de alcanzar por cuestiones de tiempo. He tenido que adaptar la aplicación a las limitaciones existentes y dejar atrás otras funcionalidades que requieren más experiencia en la materia.

La finalidad de esta trabajo ha sido:

Poner en práctica los conocimientos adquiridos a lo largo de estos dos años.

Obtener experiencia para afrontar los desafíos que supone llevar a cabo un proyecto completo.

Conocer cómo completar el desarrollo de una aplicación móvil.

Poder documentar y justificar las decisiones tomadas en el desarrollo y los resultados obtenidos.

Estar capacitado de presentar el trabajo realizado en público.

El desarrollo de este trabajo me ha permitido llevar a cabo ese interés personal por la programación para dispositivos móviles y ampliar mis conocimientos al tratar con diferentes plataformas, utilizar diferentes lenguajes y manejar distintos entornos de desarrollo.
Los principales puntos en los que realizar este proyecto me ha permitido mejorar en esta rama de la informática y cubrir tareas en las que no me había visto inmersa.

Como conclusión final a este trabajo puedo destacar que he aprendido a desarrollar un proyecto de principio a fin con todas las fases necesarias, adquiriendo y ampliando nuevos conocimientos de Java, SQL, acceso a datos, PHP, el mecanismo de comunicación del modelo cliente/servidor y la tecnología JSON, comprender todo lo relacionado con las tecnologías móviles y en particular Android y muchos otros necesarios para la consumación del proyecto.

6. COMPONENTES

Ana Belén Fernández Díaz

7. BIBLIOGRAFIA

https://stackoverflow.com/

https://developer.android.com/training/volley/requestqueue

http://www.hermosaprogramacion.com/2015/02/android-volley-peticiones-http/

http://www.hermosaprogramacion.com/2015/08/tutorial-layouts-en-android/

https://www.adictosaltrabajo.com/2014/03/11/android-volley/

http://gpmess.com/blog/2014/05/28/volley-usando-webservices-en-android-de-manera-sencilla/

https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm

https://es.wikipedia.org/wiki/Singleton

-https://jsonlint.com/

https://reactiveprogramming.io/books/design-patterns/es/catalog/adapter

http://tomcat.apache.org/

https://es.wikipedia.org/wiki/WAMP

https://www.lancetalent.com/blog/6-buenos-motivos-para-trabajar-con-php/

http://www.proyectosimio.com/es/programacion-android-base-de-datos-i-modelo-vista-controlador/

https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo/related

https://www.desarrollolibre.net/blog/android/que-son-y-como-se-configuran-los-adaptadores-adapters-para-mostrar-listas-y-grids-en-android-parte-1

https://experto.dev

https://desarrolloweb.com