Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Reporte Final del Proyecto de Integración

Licenciatura en Ingeniería en Computación

Modalidad de Proyecto Tecnológico

Prototipo de una aplicación móvil para la gestión de síntomas de un paciente

Francisco Javier García Maldonado
209300190

Asesoras:
Ma. Lizbeth Gallardo López
Beatriz Adriana González Beltrán

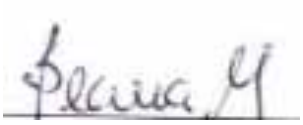
Trimestre 2014 Otoño

Lunes 15 de diciembre de 2014

Yo, Ma. Lizbeth Gallardo López, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Ma. Lizbeth Gallardo López
glizbeth@azc.uam.mx

Yo, Beatriz Adriana González Beltrán, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Beatriz Adriana González Beltrán
bgonzalez@azc.uam.mx

Yo, Francisco Javier García Maldonado, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Francisco Javier García Maldonado
fjgm-@hotmail.com

Agradecimientos

La conclusión de este Proyecto de Integración solo representa el inicio de una nueva etapa de mi vida. Representa los cimientos que sostendrán una pirámide de sueños. Sueños que son producto de la imaginación. Sueños que comparto con las personas que más estimo y que daría mi propia vida por verles sonreír una vez más.

Agradezco a mi alma mater, la Universidad Autónoma Metropolitana y a todos los profesores por haberme brindado la oportunidad de ser uno más de sus alumnos.

A mis asesoras, Beatriz y Lizbeth, por la atención que me ofrecieron, no solo durante la realización de este proyecto, sino también por la educación como persona que siempre me brindaron.

A mis padres, Flavio y Simona, decirles que no existen mejores padres que los míos. Les doy las gracias por haberme concebido, haberme formado y convertido en el hombre que ahora soy. Gracias por su apoyo, sus consejos y sus regaños y, principalmente, gracias por el amor y paciencia que siempre me brindaron en estos 5 años que estuve lejos de ustedes. Les dedico este trabajo como muestra del gran amor que les tengo.

A mis hermanas, Yaneth y Alicia, por haberme escuchado y aconsejado. Ustedes, junto con mi madre, son las tres mujeres más importantes de mi vida. Les doy las gracias por ser mis hermanas y decirles que para cualquier cosa que necesiten estaré allí para ustedes. Las amo con todo el corazón a las tres.

Y especialmente, decir gracias a una persona que siempre me ha apoyado en todas ocasiones. A mi admirable hermano Ben, decirte que tú siempre has sido mi inspiración, tus acciones, tú conducta y determinación han sido factores imprescindibles que hacen que todo esto sea posible. Espero haberte hecho sentir orgulloso con este logro que también es tuyo. Decirte que nunca nadie tendrá mejor hermano que yo. A ti ¡Gracias!

Si lo puedes imaginar, lo puedes crear.
Javier García

Resumen

Las características que nos proporcionan los dispositivos móviles han permitido el desarrollo de aplicaciones en el ámbito de la salud tales como *GoodRx*, *Itriage* y *Fooducate*; sin embargo ninguna de ellas está enfocada a la gestión de síntomas de un paciente. La originalidad de este proyecto radica principalmente en permitir registrar un nuevo síntoma y obtener un resumen de los síntomas registrados para ayudar al paciente a explicar, en su próxima cita con su médico tratante, los síntomas que presentó en un periodo de tiempo.

El objetivo principal de este proyecto fue desarrollar un prototipo de aplicación móvil, llamado P-MOSIP, para la gestión de los síntomas de un paciente. Los objetivos particulares fueron: 1) implementar y realizar pruebas de los módulos editar, modificar, eliminar y obtener resumen de síntomas, 2) analizar, diseñar, implementar y realizar pruebas de los módulos respaldo y recuperación de síntomas mediante un servicio web y 3) integrar los módulos en la aplicación.

Existen diversas aplicaciones enfocadas a la salud. Por ejemplo, *iTriage* es una aplicación móvil que ofrece al usuario una serie de descripciones de síntomas y proporciona al usuario las posibles causas y emite una serie de recomendaciones para mejorar su estado de salud. Otro ejemplo es SICODIVE, un proyecto de integración de la UAM-A que permite registrar y administrar datos sobre el estado patológico de los pacientes.

Para el desarrollo de P-MOSIP se utilizó como metodología el Proceso Unificado. La aplicación *Android* sigue una arquitectura Modelo-Vista-Controlador y la comunicación con el servidor sigue un modelo de dos capas. Para su implementación se utilizaron diversas tecnologías, entre ellas: *SQLite*, *MySQL*, *GSON*, *Jersey* y *JavaConnector*.

Tabla de contenido

Índice de figuras

1. Introducción

En la actualidad existen múltiples aplicaciones para dispositivos móviles y en particular para teléfonos inteligentes (*smartphones*), es así como el teléfono pasó de ser solo un medio para realizar llamadas telefónicas a ser un dispositivo con una gran variedad de aplicaciones tales como mensajería, juegos, agendas, navegación web, mapas, etc.

Un área de oportunidad para desarrollar aplicaciones móviles es el área de la salud. En particular, para un paciente es importante llevar un registro de los síntomas experimentados puesto que forman parte de su historial clínico y más aún si se trata de un paciente con una enfermedad crónica. El objetivo principal de este proyecto fue realizar un prototipo de aplicación móvil para la gestión de síntomas de un paciente. El usuario es un paciente diagnosticado con alguna enfermedad crónica. Es importante mencionar que este proyecto forma parte de un proyecto de investigación interdisciplinario formado por profesores en computación del Departamento de Sistemas de la División de Ciencias Básicas e Ingeniería y por profesores del Departamento de Investigación y Conocimiento de la División de Ciencias y Artes para el Diseño. Este grupo de investigación está interesado en proponer soluciones computacionales a problemas en el ámbito de la salud.

El prototipo desarrollado (en lo sucesivo lo llamaremos P-MOSIP) presenta cuatro funciones generales:

1. El registro de un síntoma se puede realizar de dos maneras: i) con ayuda del teclado y ii) por la selección de un área del cuerpo humano. En el registro de un síntoma con ayuda del teclado, el usuario escribe las letras de algún síntoma en un cuadro de texto, el sistema va filtrando los síntomas y presenta una lista de aquellos cuyo nombre inicie con dicha(s) letra(s).El registro de un síntoma por la selección de un área del cuerpo humano permite que el sistema presente una lista de síntomas asociados a esa parte del cuerpo y posteriormente se elija el síntoma.
2. P-MOSIP permite consultar un resumen de los síntomas registrados hasta el momento. Para cada síntoma registrado, se presenta su nombre, fecha y hora de inicio, fecha y hora de término, intensidad y un comentario.
3. P-MOSIP ofrece la opción de eliminar alguna entrada de la lista a través del resumen de los síntomas.
4. P-MOSIP permite respaldar los síntomas registrados en el *Smartphone* en un servidor remoto y recuperarlos desde este servidor.

P-MOSIP se implementó para dispositivos móviles con sistema operativo *Android*, para las versiones *4.3 Jelly Bean* o superiores.

A continuación se describen las secciones que conforman el presente documento. La sección 2 presenta los trabajos relacionados con este proyecto de integración. La sección 3

describe la motivación, la importancia y la originalidad de este proyecto. La sección 4 menciona los objetivos del proyecto. La sección 5 explica el marco teórico del proyecto. La sección 6 se explica la metodología que se siguió para el desarrollo de P-MOSIP. La sección 7 expone los resultados obtenidos. La sección 8 presenta un análisis sobre los objetivos alcanzados. La sección 9 presenta las conclusiones del proyecto. Finalmente, la sección 10 menciona las dificultades enfrentadas durante el desarrollo de P-MOSIP y se sugieren algunas modificaciones y algunos consejos para mejorarlo.

2. Antecedentes

Las secciones 2.1 y 2.2, detallan brevemente los trabajos que preceden al proyecto P-MOSIP.

2.1 Referencias internas

Expediente clínico ECE [1]

Este sistema permite la automatización en la elaboración de expedientes clínicos electrónicos por parte de los médicos. Está enfocada a la salud y es para el médico, no para el paciente; sigue la Norma Oficial Mexicana NOM-168 SSA1-1998. Se registran los signos vitales y síntomas de un paciente. Se desarrolló bajo la plataforma Java para una computadora de escritorio.

Este expediente clínico registra los signos vitales de una persona. Por su parte P-MOSIP no registra los signos vitales, sino los síntomas de los padecimientos de una persona con alguna enfermedad crónica.

Reingeniería del sistema de consulta y diagnóstico médico veterinario SICODIVE [2]

SICODIVE es una aplicación que apoya al médico veterinario para: i) la gestión de consultas médicas y ii) el diagnóstico de pacientes mediante la sugerencia de tratamientos. En este último módulo, SICODIVE opera con una base de conocimiento sobre elixires (del área de homeopatía) que se corresponden con una serie de síntomas. El sistema sugiere al veterinario un tratamiento a partir de los síntomas del paciente.

Tanto P-MOSIP como *SICODIVE* son sistemas enfocados al ámbito de la salud. Sin embargo, *SICODIVE* está enfocado en el médico mientras que P-MOSIP está enfocado en el paciente. Además, SICODIVE es una aplicación WEB mientras que P-MOSIP es una aplicación móvil.

2.2 Referencias externas

Existe actualmente varias aplicaciones para la salud, algunas están a la venta y otras son gratuitas. A continuación, se presentan algunas aplicaciones que son similares a P-MOSIP.

iTriage Health

iTriage tiene las funcionalidades siguientes: i) ofrece al usuario una serie de descripciones de síntomas y proporciona al usuario las posibles causas y emite una serie de recomendaciones para mejorar su estado de salud; ii) permite ubicar el hospital más cercano al sitio donde se encuentra el paciente; iii) permite consultar a una enfermera consejera; iv) permite consultar la descripción de más de mil medicamentos. La figura 1 muestra la interfaz principal de la aplicación *iTriage Health*.

Tanto *iTriage* como P-MOSIP, presentan el modo de buscar un síntoma a través del teclado y de la selección de una figura humana. Sin embargo, *iTriage* está destinado a proporcionar descripción de los síntomas y sus posibles causas; mientras que P-MOSIP está destinado a la gestión de los síntomas por parte de un paciente con alguna enfermedad crónica.

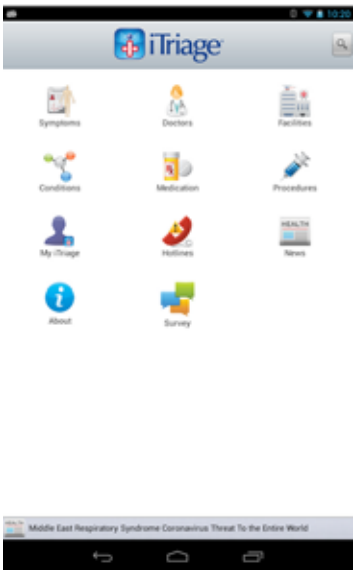


Figura 1. Interfaz principal de la aplicación móvil iTriage Health. (Tomado de [3]).

GoodRx

GoodRx tiene las siguientes funcionalidades: i) compara precios de prácticamente todos los medicamentos aprobadas por el FDA (*Food and Drug Administration*)en Estados Unidos; ii) ofrece información sobre medicamentos, a saber: nombre y presentación del medicamento (líquido o tabletas comprimidas); iii) informa sobre la ubicación de las farmacias. *GoodRx* busca la información en las páginas web disponibles.

Tanto *GoodRx* como P-MOSIP permiten introducir las primeras letras de alguna palabra, su sistema filtra y presenta una lista de aquellas palabras que comiencen con esas letras. La figura 2 muestra la interfaz principal de *GoodRx*. Sin embargo, P-MOSIP presenta una lista de síntomas y *GoodRx* una lista de medicamentos.

Figura 2. Interfaz principal de la aplicación móvil GoodRx. (Tomado de [4]).

Fooducate

Es una aplicación que nos ayuda a conocer mejor los alimentos. Sus características son: i) permite conocer el contenido de azúcar, grasas trans, colorantes y/o aditivos de un producto, ii) presenta la información de la etiqueta de un producto alimenticio con palabras sencillas y iii) puede dar alternativas de productos más saludables. La figura 3 muestra la interfaz de la aplicación *Fooducate*.

Tanto *Fooducate* como P-MOSIP permiten introducir las primeras letras de alguna palabra, su sistema filtra y presenta una lista de aquellas palabras que comiencen con esas letras. P-MOSIP, presenta una lista de síntomas, y *Fooducate* una lista de productos alimenticios.

Figura 3. Interfaz principal de la aplicación móvil Fooducate. (Tomado de [5]).

3. Justificación

El desarrollo de aplicaciones en el área de la salud es un área de oportunidad y la oportunidad es aún mayor si aprovechamos las características de ubicuidad que nos proporcionan los dispositivos móviles.

En los últimos años se ha demostrado un mayor interés en el ámbito de la salud con el surgimiento de aplicaciones tales como *GoodRx*, *Itriage* y *Fooducate*; sin embargo ninguna de ellas está enfocada a la gestión de síntomas de un paciente.

Existen otras aplicaciones móviles de propósito general, tales como agendas y planificadores que brindan una manera de almacenar eventos, sustituyendo a las notas en papel; sin embargo, estas aplicaciones no se adaptan adecuadamente para que un paciente gestione sus síntomas dado que le dan mayor importancia a la fecha que al evento mismo, en este caso el síntoma.

La originalidad de este proyecto radica principalmente, en que permite registrar de manera sencilla un nuevo síntoma (nombre del síntoma, fecha de inicio, intensidad). Además, la aplicación permite obtener un resumen de los síntomas registrados para ayudar al paciente a explicar, en su próxima cita con su médico tratante, los síntomas que presentó en un

periodo de tiempo; de tal manera que el médico conozca con mayor precisión la evolución de la enfermedad y le apoye a emitir un diagnóstico.

4. Objetivos

4.1 Objetivo general

Desarrollar un prototipo de aplicación móvil para la gestión de los síntomas de un paciente.

4.2 Objetivos particulares

1. Implementar y realizar pruebas de los siguientes módulos: Editar, modificar, eliminar síntomas y obtener un resumen de síntomas.
2. Analizar, diseñar, implementar y realizar pruebas de los siguientes módulos: respaldo y recuperación de síntomas mediante un servicio web.
3. Integrar los módulos en la aplicación.

5. Marco teórico

5.1. La plataforma *Android*

Android es un sistema operativo basado en Linux para teléfonos móviles, incluye una máquina virtual llamada *Dalvik Virtual Machine* [6]la cual es capaz de ejecutar una sola aplicación; por lo tanto, por cada aplicación se tiene una instancia de *Dalvik*.

5.1.1. Arquitectura

Android tiene una arquitectura de cinco capas (ver figura 4), donde cada de una de las capas utiliza los servicios brindados por la capa inferior, para realizar sus propias funciones [7].

1. ***Kernel* de Linux.** El núcleo del sistema operativo *Android* está basado en el *kernel* de Linux 2.6, similar al de Ubuntu, solo que adaptado para soportar características del *hardware* de un dispositivo móvil. El núcleo interactúa directamente con el *hardware* del dispositivo móvil. Para cada elemento de *hardware* del teléfono existe un controlador (*driver*) dentro del *kernel* que permite ser utilizado a partir del *software*.

Figura 4. Arquitectura de cinco capas de Android (Modificado de [7]).

2. **Bibliotecas.** Capa situada sobre el *kernel*. Se trata de bibliotecas nativas de *Android* escritas en C++ o C. Las bibliotecas incluidas habitualmente son: *OpenGL*, bibliotecas multimedia, *WebKit* (navegador), *SSL* (cifrado de comunicaciones), *FreeType* (fuentes de texto), *SQLite* (base de datos), entre otras.

3. ***AndroidRuntime.*** El componente principal es la *Dalvik Virtual Machine*. *Dalvik* es una máquina virtual intérprete que ejecuta archivos en el formato *Dalvik Executable* (**.dex*). **.dex* es un formato optimizado para el almacenamiento eficiente y ejecución mapeable en memoria. *Dalvik* permite que el código sea compilado a un *bytecode* independiente del dispositivo en el que se va a ejecutar, e interpreta este *bytecode* a la hora de ejecutar el programa.

4. ***Framework* de aplicaciones.** Formada por todas las clases y servicios que utilizan las aplicaciones, tales como:

    1. `ActivityManager`. Administra el ciclo de vida de la aplicación.
    2. `WindowsManager`. Administra lo que se muestra en pantalla.
    3. `ContentProvider`. Permite compartir información entre aplicaciones.
    4. `Views`. Permitenconstruir las interfaces.
    5. `NotificationManager`. Servicios para notificar al usuario sobre eventos que requieren su atención.
    6. `PackageManager`. Controla los paquetes .apk, que son paquetes instalados en el dispositivo móvil.
    7. `TelephonyManager`. Esta biblioteca permite enviar y recibir SMS/MMS.
    8. `ResourceManager`. Permite controlar elementos, tales como cadenas de texto, imágenes, sonidos o `Layouts`.
    9. `LocationManager`. Determina la posición del dispositivo mediante GPS.

10. `SensorManager`. Manipula elementos *hardware*, tales como el acelerómetro, giroscopio, sensor de luminosidad, entre otras.
11. `Camara`. Para tomar fotografías y videos.
12. Multimedia. Reproducir y visualizar audio y video.

5. **Aplicaciones.** Aquí está la aplicación principal del sistema: Inicio (*Home*) o lanzador (*launcher*). Permite ejecutar otras aplicaciones. En esta capaes donde se aloja P-MOSIP.

5.2. Componentes de una aplicación Android

Todas las aplicaciones *Android*, se componen de cuatro componentes principales: `Activity`, `BroadcastIntentReceiver`, `Service`, `ContentProvideryApplicationContext`. Cada uno de ellos debe ser declarado de forma explícita enun archivo XML llamado "AndroidManifest.xml".

5.2.1.`Activity`

Es el componente más elemental de las aplicaciones *Android*. El usuario interactúa con un `Activity` con el propósito de solicitar, a la aplicación, el realizar una tarea específica. Un *`Activity`* tiene un ciclo de vida que consiste en los estados siguientes: *Created*, *Started, Resumed, Paused, Stopped* y *Destroyed* (ver figura5).

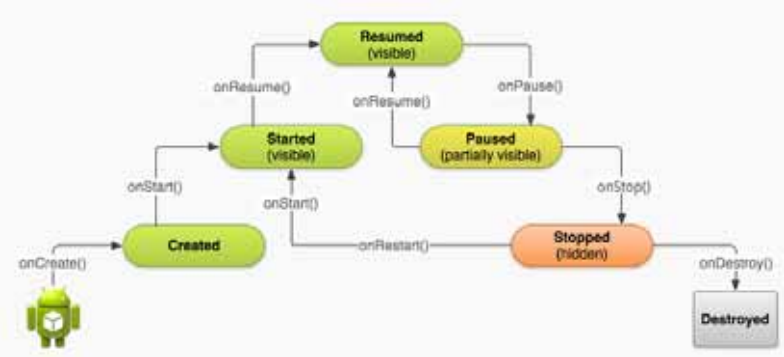


Figura 5. Ciclo de vida de un Activity. (Tomado de [8]).

Todas las aplicaciones *Android* inician con un *`Activity`* principal definido en el archivo AndroidManifest.xml, este *`Activity`* se crea llamando a su método `onCreate()`, el cual describe como debe crearse la interfaz de usuario y qué otros requisitos necesita la aplicación antes de aceptar cualquier interacción con el usuario(por ejemplo, establecer una conexión con algún servidor). La interfaz del *`Activity`* se hace visible cuando se

llama al método `onStart()` (estado *Started*) e inmediatamente se llama al método `onResume()` (estado *Resumed*). Es en el estado *Resumed* cuando el usuario puede interactuar con la aplicación. Un *`Activity`*en estado *Paused,* significa que otro *`Activity`* fue creado y que se encuentra en estado *Resumed.* El *`Activity`* en estado *Paused* es aún visible para el usuario, pero no recibe entradas del usuario y no ejecuta ningún código. Si un *`Activity`* está en estado *Stopped*, no es visible para el usuario, se encuentra en segundo plano. Mientras estemos en el estado *Stopped*, la instancia del *`Activity`* y toda la información de su estado, contenida en las variables miembro, es retenida, y no puede ejecutar ningún código. El sistema llama al método `onDestroy()` para que la instancia del *`Activity`* sea completamente removida de la memoria del sistema (estado *Destroyed*).

El programador debe determinar el comportamiento en cada estado de un *`Activity`*. Por ejemplo, para un reproductor de video podemos programar que cuando llegue alguna llamada telefónica, el reproductor pase al estado de *Paused* en la reproducción. Así, no nos perderemos de la película mientras estemos atendiendo la llamada.

El sistema operativo *Android*, tiene la facultad de terminar una aplicación; es decir, pasar el `Activity` principal al estado *Destroyed* cuando se presente algún evento de mayor prioridad (por ejemplo, se está agotando la memoria disponible), pero éste tiene la responsabilidad de restablecer a la aplicación en el estado en el cual se encontraba justo antes de haberlo terminado.

5.2.2.BroadcastIntentReceiver

Un `BroadcastIntentReceiver` es el encargado de avisar a la aplicación actual (aquella que se está ejecutando) cuando un evento determinado se ha producido. Por ejemplo, la llegada de una llamada telefónica. No tiene asociado una interfaz, pero puede informarse del evento a través de la barra de estado del dispositivo móvil.

5.2.3. `Service`

Un `Service` es una aplicación sin interfaz gráfica. Su ejecución se hace en segundo plano mientras otras aplicaciones (éstas con interfaz gráfica) están ejecutándose. Por ejemplo, mientras un reproductor de música está en ejecución, es posible buscar una canción mientras se escucha otra; un `service` está encargado de continuar la reproducción durante la búsqueda.

5.2.4. `ContentProvider`

`ContentProvider` son *interfaces* que permiten compartir datos entre aplicaciones. Brinda la posibilidad de guardar información en un archivo o en una base de datos.

5.2.5.`ApplicationContext`

Corresponde al entorno de la aplicación y al proceso dentro del cual sus componentes se están ejecutando.

5.3.*Kit* de desarrollo para *Android*

Para desarrollar aplicaciones móviles para *Android* se emplea el *Android* SDK (*Software Development Kit*), el cual está formado por: un depurador de código, un emulador basado en QEMU, biblioteca, tutoriales, documentación y código de ejemplos.

Eclipse ADT (*Android Development Tools*) permite desarrollar aplicaciones *Android*, incluye las herramientas: Eclipse + ADT *plugin*, el SDK, herramientas de la plataforma *Android*, una versión de la plataforma *Android* y una versión del sistema *Android* para el emulador. Simplemente se descarga [9] y se descomprime en la ruta deseada.

5.3.1 El emulador

El emulador (ver figura 6) incluido en el SDK de *Android* simula un procesador de tipo ARM (*Advanced RISC Machine*)y permite probar las aplicaciones durante el desarrollo; sin embargo no permite probar aplicaciones que hagan uso de los sensores del dispositivo.

Figura 6. Emulador Android 4.0.

5.4. Estructura de un proyecto *Android*

Un proyecto *Android* involucra una serie de carpetas y un archivo. Las carpetas son: src, res y bin; el archivo es *AndroidManifest.xml*; la figura 7 muestra la estructura básica.

5.4.1. Carpeta \src

Dentro de esta carpeta se encuentran todos los archivos fuente .java del proyecto. También es posible crear paquetes dentro de esta carpeta. Dentro de esta carpeta se encuentra un archivo llamado R.java. Este archivo es generado por el ADT y no debe modificarse.

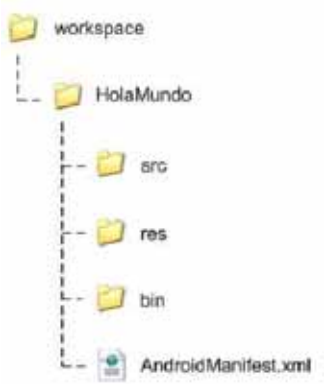


Figura 7. Jerarquía de subcarpetas en un proyecto Android.

5.4.2. Carpeta \res

Esta carpeta contiene todos los recursos utilizados por el proyecto. Por ejemplo, un recurso puede ser una imagen que represente el ícono de nuestra aplicación. La mayoría de los recursos son descritos por archivos XML. Por ejemplo, un recurso de texto que describe las traducciones de una aplicación para distintos idiomas. Dentro de esta carpeta están las siguientes subcarpetas:

- *ani*: contiene archivos XML que describen distintas animaciones.
- *drawable*: contiene imágenes en varios formatos: JPG, PING o GIF.
- *layout*: contiene los diseños que definen la construcción de la interfaz.
- *values*: contiene archivos XML que contiene valores de diferentes tipos: cadenas de texto, colores, dimensiones, *array* de elementos, etc.

5.4.3. Carpeta bin

Contiene todos los archivos binarios generados a partir de archivos fuente.

5.4.4. Archivo *AndroidManifest.xml*

AndroidManifest.xml es un archivo que contiene la descripción de cada uno de los componentes que forman la aplicación que se está desarrollando. Aquí se indica si la aplicación tendrá acceso a internet, cuál será el `Activity` principal, cuál es el tema de la aplicación, entre otros. Debe cuidarse la estructura de este archivo para no incurrir en errores, tales como referencias no encontradas y manejo de permisos. Un ejemplo de referencias no encontradas es la navegación hacia un *Activity* no definida dentro de la jerarquía de *Activities*. Un ejemplo de manejo de permisos consiste en que la aplicación no logre establecer conexión con algún servidor porque en el *AndroidManifest.xml* no se editó el permiso para acceder a la red.

6. Desarrollo de P-MOSIP

P-MOSIP es una aplicación móvil para dispositivos con Sistema Operativo *Android*. Esta aplicación representa un prototipo para un proyecto de investigación dirigido por profesores del Departamento de Sistemas de la División de Ciencias Básicas e Ingeniería y por profesores del Departamento de Investigación y Conocimiento de la División de Ciencias y Artes para el Diseño de la Universidad Autónoma Metropolitana quienes tienen interés en proponer soluciones computacionales en el ámbito de la salud.

6.1. Metodología empleada en el desarrollo del proyecto

La metodología que se siguió para el desarrollo de P-MOSIP fue el proceso unificado, el cual se caracteriza por ser iterativo e incremental.

Cabe señalar que el modelado del negocio y el análisis de requerimientos fueron realizados por el equipo de profesores-investigadores que participan en el proyecto de investigación Proyecto SI001-14 aprobado en la sesión 544 ordinaria del Consejo Divisional de Ciencias e Ingeniería celebrada los días 26 y 30 de septiembre de 2014. Para iniciar el proyecto de integración se sostuvieron varias sesiones con el equipo de profesores investigadores para discutir sobre los requerimientos del sistema previamente identificados.

La explicación de la metodología utilizada se presenta en dos partes: 1) prototipo de aplicación *Android* para la gestión de síntomas de un paciente y 2) servicio Web para el respaldo y recuperación de síntomas.

En esta parte solo interviene la disciplina de implementación y de pruebas. Estas actividades se desarrollaron en cuatro iteraciones:

1. Implementación de los casos de uso: editar síntoma, modificar síntoma, eliminar síntoma; en ese orden.
2. Realización de pruebas de unidad de los módulos editar, modificar y eliminar síntomas.
3. Integración de los tres módulos anteriores.
4. Realización de pruebas del sistema integrado.

Servicio Web para el respaldo y recuperación de síntomas

Esta parte intervienen las disciplinas de análisis, diseño, implementación y pruebas del servicio web. Estas actividades se desarrollaron en tres iteraciones:

1. Análisis, diseño e implementación delos casos de uso respaldar y recuperar síntomas del lado del servidor.
2. Implementación del lado del cliente de los casos de uso respaldar y recuperar síntomas utilizando el servicio web.
3. Pruebas de sistema sobre el sistema integrado.

6.2. Diseño del sistema

El diseño del sistema está expresado a través de diagramas UML. Primero se presenta el diagrama de casos de uso para P-MOSIP; luego se describe el caso de uso "Editar síntoma"; enseguida, se presenta el diagrama de paquetes y el diagrama de clases tanto para la aplicación P-MOSIP como para el servicio Web; después, se presenta la arquitectura Modelo-Vista-Controlador (MVC)que sigue la aplicación *Android* y el modelo de dos capas para la comunicación con el servicio Web; finalmente, se describen los modelos entidad-relación para las bases de datos.

6.2.1. Diagrama de casos de uso.

La figura 8 muestra dos actores, el paciente como actor principal y un servicio Web como actor secundario. El actor principal puede interactuar con las funcionalidades de la aplicación: editar síntoma, modificar síntoma, eliminar síntoma, recuperar síntoma y respaldar síntoma. Mientras que el actor secundario solo puede interactuar con el sistema a través de las funcionalidades: recuperar síntomas y respaldar síntomas.

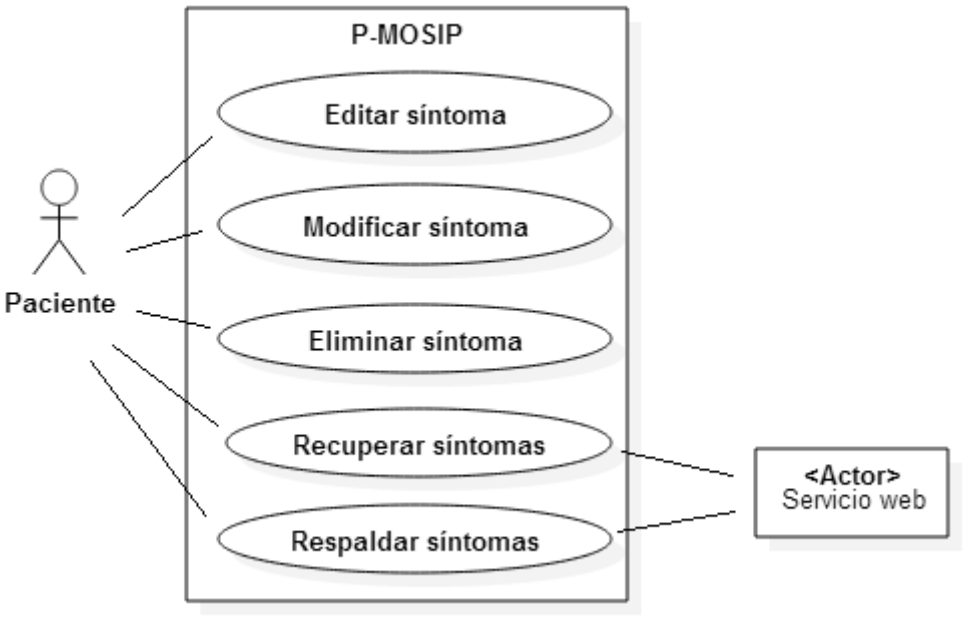


Figura 8. Diagrama de casos de uso.

A continuación se presentan el caso de uso texto "Editar síntoma", el cual servirá de ejemplo para las secciones de diseño e implementación. Para conocer el diseño del resto de la aplicación P-MOSIP, referirse a la memoria de diseño de este proyecto.

Caso de uso: Editar síntoma.

Resumen	Este caso de uso permite al paciente añadir un nuevo síntoma a su historial.
Actor	Principal Paciente
Precondiciones	Haber iniciado el sistema.
Descripción	Flujo principal: 1. El sistema muestra una figura humana. 2. El paciente selecciona, sobre la figura humana, alguna parte del cuerpo. 3. El sistema muestra una lista de síntomas. 4. El paciente selecciona un síntoma. 5. El sistema muestra un formulario para capturar las características del síntoma. 5.1 El paciente selecciona una intensidad. 5.2 El paciente edita la descripción del malestar. 6. El paciente selecciona la opción "Guardar".

	7. El sistema guarda el síntoma nuevo. 8. El sistema muestra el mensaje "Síntoma nuevo guardado". Flujo alternativo Paso 1 1.1 El usuario selecciona la opción "Buscar por texto". 1.2 El sistema muestra una lista de síntomas. 1.2.1 Se da la opción de buscar un síntoma por medio de un cuadro de texto. 1.3 Continúa en el paso 4 del flujo principal. Paso 6 6.1 El paciente selecciona la opción "Cancelar". 6.2 El sistema muestra nuevamente la figura humana.
Postcondiciones	El nuevo síntoma queda registrado en el sistema y se actualiza el historial del paciente.

6.2.2. Diagrama de paquetes y diagrama de clases

Los diagramas de clase y de paquetes que se presentan en las siguientes secciones representan los elementos de software para P-MOSIP.

6.2.2.1. Descripción del diagrama de paquetes para la aplicación P-MOSIP

La aplicación P-MOSIP está compuesta por tres paquetes (ver figura 9). El paquete `com.myspace.proyectoterminal.controlador` contiene todas las clases controlador que están a la espera de cualquier evento que ocurra cuando el usuario interactúe con el dispositivo. El paquete `com.myspace.proyectoterminal.modelo` contiene todo el código para actualizar y consultar la base de datos de la aplicación y el paquete `com.myspace.servicio.mocajersey` contiene componentes *Beans* encargados de almacenar información del paciente y el síntoma. A partir de estos *Beans* se genera el formato JSON de los objetos para poder ser enviados a través del protocolo HTTP.

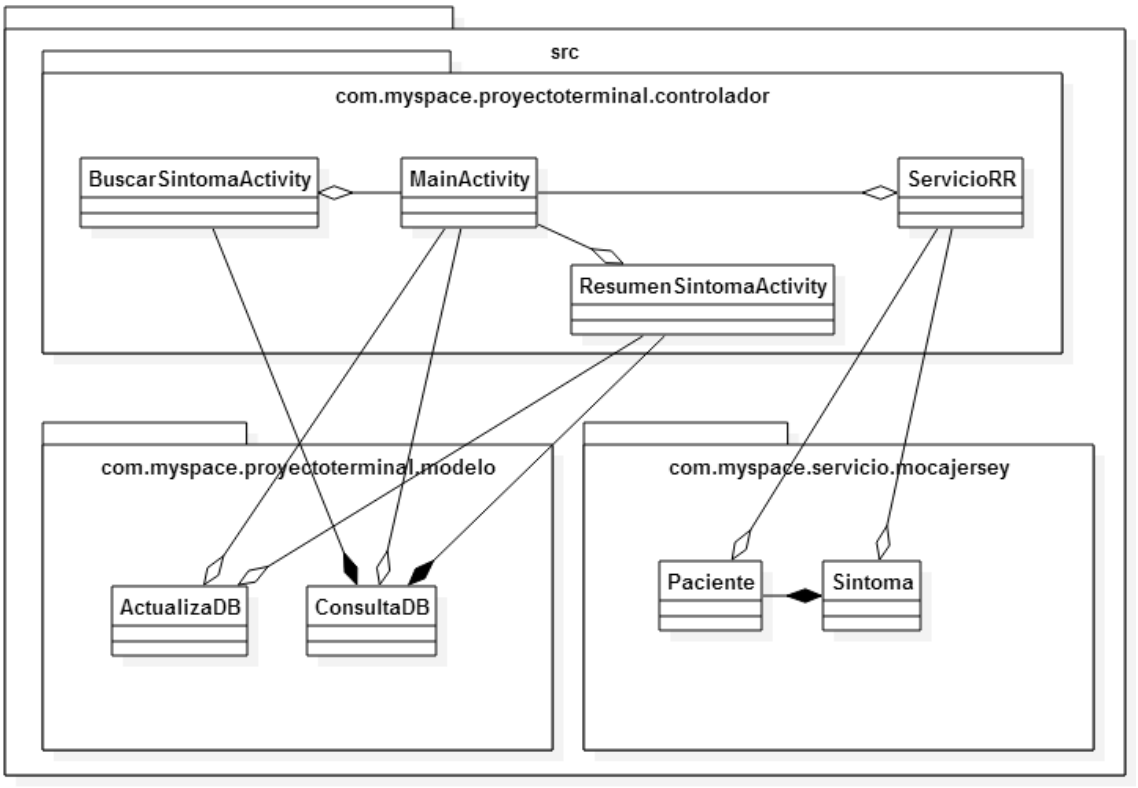


Figura 9. Diagrama de paquetes.

Diagrama de clases del paquete:
`com.myspace.proyectoterminal.controlador`

La figura 10 muestra el diagrama de las clases controlador. El punto de entrada para la aplicación P-MOSIP es la *Activity*`MainActivity`. `MainActivity` la cual está compuesta por *Fragments*. Se crea una instancia de la clase `DialogodeBienvenida` desde `MainActivity` cuando se termina de instalar P-MOSIP.

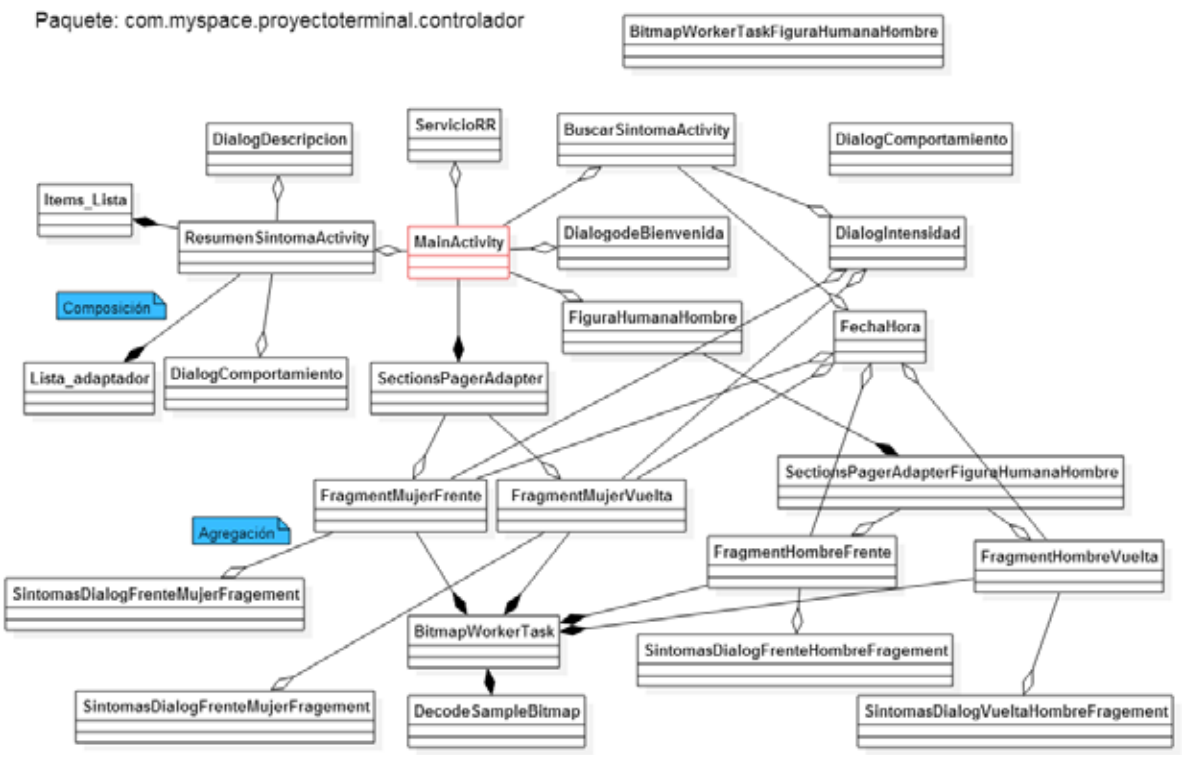


Figura 10. Diagrama de clases. Paquete controlador.

`SectionsPagerAdapter` y `SectionsPagerAdapterFiguraHumanaHombre` funcionan como una especie de libro, cuyas páginas son: `FragmentMujerFrente`, `FragmentMujerVuelta`, `FragmentHombreFrente` y `FragmentHombre-Vuelta`. Estos cuatro *`Fragments`* son los encargados de mostrar las figuras humanas y proporcionar el efecto de deslizamiento cuando el usuario desplaza la yema del dedo sobre la pantalla del dispositivo.

Las clases: `SintomasDialogFrenteHombreFragement`, `SintomasDialog-FrenteMujerFragement`, `SintomasDialogVueltaHombreFragement` y `SintomasDialogVueltaMujerFragement`, son cuadros de diálogo representados por *Fragments*. Al seleccionar alguna parte del cuerpo de la figura, el sistema crea alguna instancia de estos *Fragments*. Los cuadros de diálogo muestran una lista de los síntomas disponibles en la base de datos para esa parte del cuerpo seleccionado. Cuando selecciona un síntoma de la lista, se crea una instancia de `DialogIntensidad`. Esta última crea una instancia de la clase `FechaHora`, encargada de leer la hora del dispositivo.

`BuscarSintomaActivity` es un *Activity* que representa la interfaz del diccionario del sistema. Una vez seleccionado un síntoma, crea una instancia de `DialogIntensidad` y `FechaHora` para hacer el registro del nuevo síntoma.

`ResumenSintomaActivity` representa la interfaz gráfica que muestra todos los síntomas registrados hasta el momento. Para cada miembro de la lista se crea una instancia de la clase `Items_Lista`. Esta instancia es mapeada con su respectivo evento, con la ayuda de `Lista_adaptador`. Cuando se selecciona un parámetro (fecha y hora término, comportamiento o comentario) del algún miembro de la lista, se muestra un cuadro de diálogo (instancias de `DialogComportamiento`, *`DialogDescripcion`*), este cuadro de diálogo permite modificar dicho parámetro.

`ServicioRR` es un *Activity* que define métodos para el acceso al servicio *web*. La conexión se realiza a través de una instancia de `TareaWSPedirSintomas`. Esta clase representa una tarea en segundo plano; es decir, un hilo de ejecución.

`DecodeSampleBitmap` es una clase que toma un mapa de bits y reduce su calidad. Por ejemplo, una imagen de 1024x758 pixeles se reduce a 100x100 pixeles. Es necesario reducir la calidad de las imágenes para no provocar un desbordamiento de la memoria.

`BitmapWorkerTask` representa un hilo que crea una instancia de `DecodeSampleBitmap`, y cuando esta última termina su labor, guarda la imagen en la memoria caché específica para la *Activity* actual.

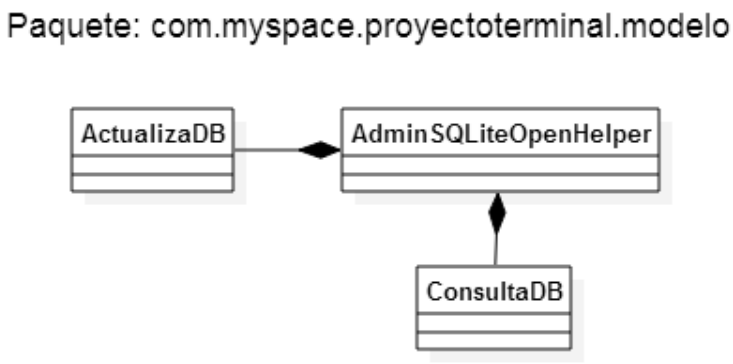


Figura 11. Diagrama de clases. Paquete modelo.

Diagrama de clases del paquete:
`com.myspace.proyectoterminal.modelo`

La figura 11 ilustra el diagrama de clases del modelo. Tanto `ConsultaDB` como `ActualizaDB` crean una instancia de `AdminSQLiteOpenHelper`, el cual es una extensión de `SQLiteOpenHelper`. `SQLiteOpenHelper` permite el acceso a una base de datos SQLite. Los síntomas recuperados por `ConsultaDB` varían dependiendo de si la figura es un hombre o una mujer y si la imagen está de frente o de espalda.

`ActualizaDB`tiene la facultad de modificar o en su caso de agregar un nuevo registro a la base de datos.

Diagrama de clases del paquete:
`com.myspace.servicio.mocajersey`

La figura 12 ilustra el diagrama de clases correspondiente al modelo que forma parte del servicio `mocajersey`. Las clases `Paciente` y `Sintoma` son clases *Beans*. Con la ayuda de una instancia de `Paciente` y una instancia de `Sintoma`, se envía y recibe información hacia y desde el servidor remoto.

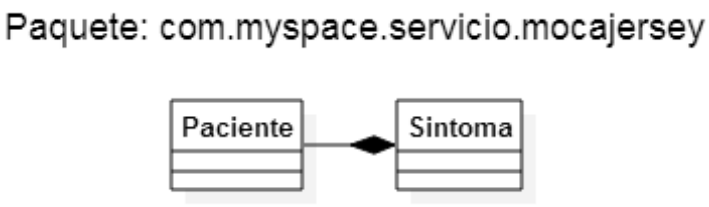


Figura 12. Diagrama de clases. Paquete mocajersey.

6.2.2.2 Descripción del diagrama de paquetes y del diagrama de clases para el servicio Web mocajersey

El servicio web `mocajersey` de la aplicación P-MOSIP está compuesto por dos paquetes (ver figura 13). El paquete `serviciomoca` contiene la clase que está a la espera de recibir peticiones. El paquete `base.datos` contiene las clases para el acceso a la base de datos.

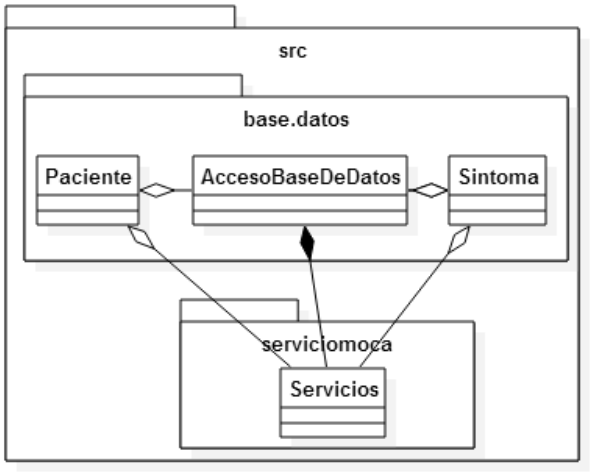


Figura 13. Diagrama de paquetes y clases del servicio web mocajersey.

Diagrama de clases del paquete: `serviciomoca`

Figura 14. Diagrama de clases. Paquete serviciomoca.

La figura 14 ilustra el diagrama de clases `serviciomoca`. Este paquete contiene únicamente la clase `Servicios`. Por medio de esta clase se publican los recursos. Por ejemplo, un recurso pudría ser "pacientenuevo".

Diagrama de clases del paquete: `serviciomoca`

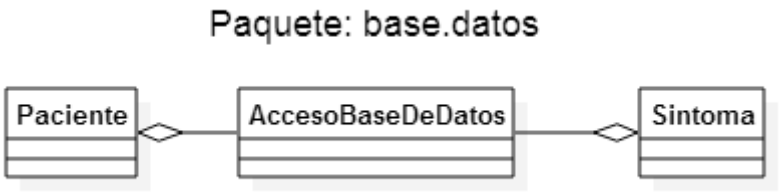


Figura 15. Diagrama de clases. Paquete base.datos.

La figura 15 ilustra el diagrama de clases `basedatos`. Paradeserializar los objetos que se proporcionan con la solicitud se crean instancias de los *Beans* `Paciente` y `Sintoma`. Con estos *Beans* y con una instancia de `AccesoBaseDeDatos` el servicio registra o recupera información de la base de datos.

6.2.3. Arquitectura del sistema

El patrón de arquitectura de software de la aplicación P-MOSIP es el Modelo-Vista-Controlador (MVC). En la figura 16 se observa el árbol de directorios de la aplicación con las distintas partes de la arquitectura.

El modelo está representado por las clases contenidas en el paquete `com.myspace.proyectoterminal.modelo`. Estas clases crean, consultan y actualizan la base de datos. En este conjunto de clases se encapsula la lógica de negocio de P-MOSIP.

La vista está formada por la carpeta \res y sus subcarpetas: *drawable*, *layout*, *menu* y *values*. Contienen todos los archivos de los recursos necesarios para el proyecto, a saber:

imágenes, videos, cadena de texto, etc. Aquí es donde se encuentran las clases de la interfaz gráfica que se presenta al usuario.

El controlador está formado por las clases del paquete `com.myspace.proyecto-terminal.controlador`. Se define la respuesta del sistema a determinados eventos generados por el usuario a través del dispositivo móvil. En el controlador se determinó la forma en que deben ser intercambiados los fragmentos, y cuándo y cómo actualizar la interfaz de usuario si el modelo hace sido modificado.

Por otra parte, se optó en establecer una arquitectura de dos capas para comunicarse con el servidor (ver figura 17), lo que nos permite encapsular o separar elementos de nuestra aplicación en partes claramente definidas.

La primera capa está compuesta por la aplicación P-MOSIP, la cual se ejecuta en el dispositivo móvil. La segunda capa está compuesta por: la base de datos y el servicio web. Ambos alojados en el servidor remoto. La comunicación entre estas dos capas se realiza a través del protocolo HTTP.

El servidor Web *Apache Tomcat* es el encargado de ejecutar el servicio Web.

Figura 16. Estructura de un proyecto de aplicación *Android*.

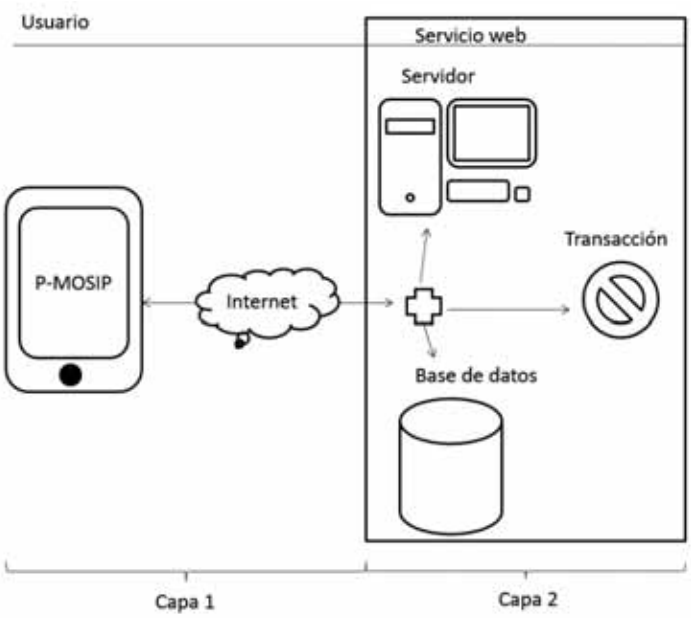


Figura 17. Modelo de dos capas. (Modificado de [10]).

6.2.4. Estructura de la base de datos

En el proyecto se utilizaron dos bases de datos, una base de datos instalada en el servidor y otra instalada en el dispositivo móvil.

Base de datos en el servidor

El modelo entidad-relación esquematizado en la figura 18 fue implementado para la base de datos que utiliza el servicio Web. Se utilizó el Sistema Gestor de Base de Datos (SGBD) MySQL.

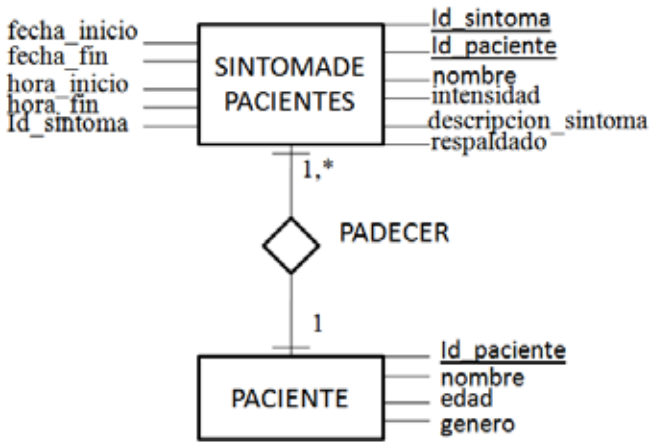


Figura 18. Modelo ER para la base de datos ubicada en el servidor dedicado.

Dado que varias personas pueden utilizar el servicio Web, la entidad PACIENTE aloja los datos de cada uno de ellos. Se eligió a Id_paciente como *PRIMARY KEY*. Cada paciente que hace uso de la aplicación tiene un identificador único. La relación PADECER establece que un paciente ha padecido uno o más síntomas.

La tabla SINTOMASDEPACIENTES es la que tendrá modificaciones constantes. Para cada entrada (síntoma) en la tabla deberá existir una sola persona que lo padeció. Se definió una llave primaria (*PRIMARY KEY)* compuesta porque un paciente puede padecer más de una vez el mismo síntoma, pero no en el mismo lapso de tiempo.

Base de datos en el dispositivo móvil

La figura 19 muestra el modelo entidad-relación para la base de datos que se implementó en el dispositivo móvil. El sistema gestor de la base de datos que viene por defecto en las aplicaciones Android es *SQLite*.

La tabla TIPOSINTOMA recoge la lista de síntomas que se desplegará al usuario cuando busque un nuevo síntoma. Esta tabla es estática y nunca se modifica una vez instalada en el dispositivo móvil.

La tabla SINTOMA contiene los síntomas que el paciente ha registrado hasta el momento. Se eligió a Id_sintoma como llave primaria (*PRIMARY KEY)* porque cada registro es único. Aunque un mismo síntoma se puede padecer más de una vez, cada padecimiento tiene un identificador único. Esta tabla almacena además del nombre del síntoma, la fecha y hora en la que inició y terminó, la intensidad, la descripción y un indicador de si el síntoma ya ha sido respaldado en el servidor remoto.

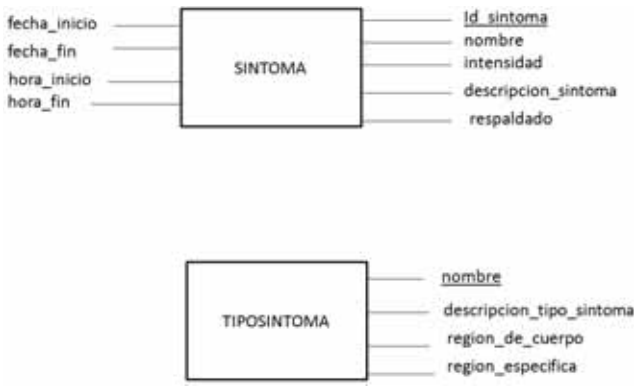


Figura 19. Modelo ER para la base de datos ubicada en el dispositivo móvil.

6.3. Uso del sistema: Caso de uso Editar síntoma

El uso del sistema lo explicaremos a partir del caso principal de la aplicación P-MOSIP, a saber: Editar síntoma.

Inicie P-MOSIP presionando el icono 

. Aparecerá la interfaz de la figura 20a.

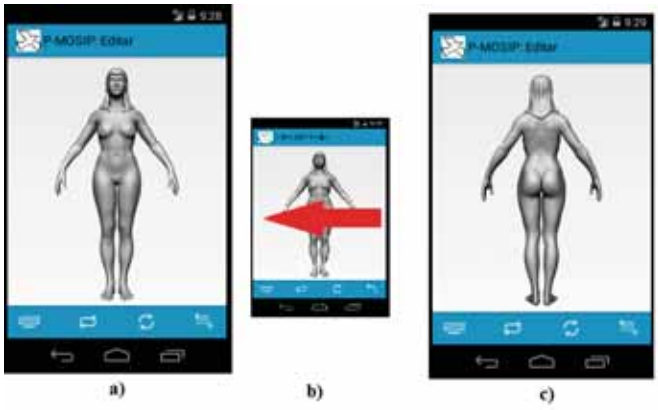


Figura 20. Interfaz que muestra la imagen de una mujer.

Desplace los dedos sobre la pantalla del dispositivo para obtener la imagen de la figura 20c.

El siguiente botón  cambia la imagen de una mujer ala de un hombre (ver figura 21).

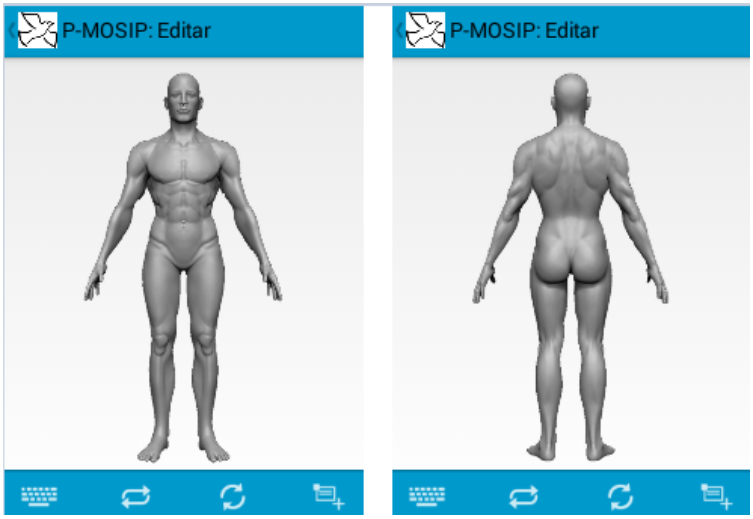


Figura 21. Interfaz que muestra la imagen de un hombre.

El botón  permite regresar a la interfaz de la figura 20a.

Para seleccionar un síntoma presione alguna parte del cuerpo de la imagen yse desplegará una lista (ver figura 22).

Figura 22. Seleccionar un síntoma.

La lista se desliza de manera vertical, siempre y cuando la cantidad de síntomas sea extensa. Al seleccionar un síntoma se desplegará el diálogo de la figura 23.

Figura 23. Seleccionar intensidad y editar un comentario al síntoma.

Un síntoma puede tener una intensidad de 1 a 10. Seleccione una intensidad con la línea azul. En la parte inferior se indica la intensidad seleccionada.

Una vez configurado el síntoma, presione "Aceptar" para guardar o "Cancelar" para salir del diálogo. Si selecciona "Aceptar", el sistema informará que ha guardado el nuevo síntoma (ver figura 24).

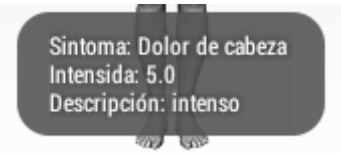


Figura 24. Diálogo que informa que se ha guardado el nuevo síntoma.

6.4. *Hardware* y *software* necesarios

6.4.1. Software

A continuación se exponen brevemente las distintas herramientas que se utilizaron para el desarrollo de la aplicación P-MOSIP.

6.4.1.1. Tecnología para el desarrollo de la aplicación

Eclipse ADT [9]. Este ADT de eclipse incluye todo lo necesario para empezar a desarrollar aplicaciones para Android, a saber:

- Eclipse + ADT *plugin*.
- *Android* SDK.
- Herramientas para la plataforma *Android*.
- Una imagen ISO del sistema operativo *Android* para el emulador.

*Android Action Bar Style Generator [11].*Esta herramienta permite diseñar temas para las aplicaciones sin entrar en detalle en los archivos XML. Cabe mencionar que *Android Action Bar Style Generator* solo permite definir el tema de la aplicación; es decir, la apariencia. Si queremos definir explícitamente alguna característica especial, por ejemplo, la transparencia de la barra de título, se debe editar dicha característica directamente en el archivo XML.

6.4.1.2. Tecnología para la instalación y puesta en marcha de la aplicación

Apache Tomcat 7[12]. *Apache Tomcat* es un servidor de aplicaciones web de código abierto que implementa *Java Servlets* y páginas web con tecnología *JavaServer*.

Se optó por la versión 7.x, porque implementa las especificaciones de *Servlet* 3.0, el cual es indispensable para definir el Servicio Web (SW) con tecnología *REST*.

MySQL 5 [13]. *MySQL* es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia.

Jersey [14]. es una implementación del*Java Specification Request* (JSR) 311. Jersey proporciona una biblioteca para implementar RESTful *web services* en el *java servlet container.* RESTful web services está basado en los métodos HTTP y en conceptos de REST (*RepresentationalState Transfer*). RESTful *web services* define la URI de base para los servicios, *MIME-types* (XML, text, JSON, entre otros) y un conjunto de operaciones (POST, GET, PUT y DELETE).

Gson de Google [15]. Gson es una biblioteca de código abierto, que permite la serialización/deserialización entre objetos Java y su representación en notación *JSON*. Esta biblioteca permite enviar información en formato texto a través del protocolo http.

Connector/J [16]. EL API JDBC (*Java Data Base Connectivity*), es el estándar entre el lenguaje de programación Java y un amplio rango de bases de datos. Este API hace posible la conexión con un DBMS (*Data Base Management System*), permite enviar enunciados SQL y procesar los resultados.

6.4.2 *Hardware*

Para el desarrollo de la aplicación P-MOSIP se empleó una computadora de escritorio con las siguientes especificaciones:

- Un procesador de 3.0 GHz
- 4 GB de memoria RAM.
- Disco dura de 512 GB.
- Un monitor de 19 pulgadas.

Se utilizó el emulador de dispositivos móviles incluido en el ADT durante el desarrollo.

7. Resultados

En este proyecto de integración se implementó la aplicación P-MOSIP que tiene los siguientes módulos: Editar síntoma, Modificar síntoma, Eliminar síntomas, Obtener un resumen de síntomas.

Además, cuenta con los módulos Respaldar síntomas y Recuperar síntomas. El almacenamiento y recuperación de síntomas lo lleva a cabo un servicio web.

8. Análisis y discusión de resultados

El prototipo desarrollado en este proyecto permite la gestión de síntomas de un paciente. P-MOSIP cumple con los objetivos planteados. No obstante, P-MOSIP aún puede ser mejorado, tanto en su código como en el diseño de su interfaz de usuario.

Uno de los problemas que se presentan es cuando se selecciona un área del cuerpo. Por ejemplo, si el usuario selecciona el brazo, aparecerá una imagen color naranja que no coincide con el brazo. Sería recomendable hacer un algoritmo que identifique cuando ha sido seleccionada una parte del cuerpo y colorear exactamente esa parte del cuerpo.

Por otra parte, es necesario rediseñar los íconos y los gráficos. En las pruebas de usabilidad resultó que no todos los íconos y gráficos son adecuados en términos de tamaño, color y dimensión.

9. Conclusiones

P-MOSIP es un prototipo para la gestión de síntomas de un paciente. P-MOSIP está compuesto por los siguientes módulos: 1) editar síntomas, permite a un usuario registrar un nuevo síntoma; 2) modificar síntoma, permite alterar los siguientes parámetros: fecha y hora de término, comportamiento y comentario; 3) eliminar síntoma, elimina un síntoma registrado previamente ; 4) obtener un resumen de síntomas, presenta una lista de los síntomas registrados hasta el momento y 5) respaldar y recuperar síntomas, respalda en y recupera síntomas desde un servidor remoto mediante un servicio web. Todos los módulos se integraron y probaron para verificar su correcto funcionamiento.

El reto en el desarrollo de aplicaciones *Android* es entender su filosofía. Además, es necesario tener en mente que el desarrollo de aplicaciones para computadoras de escritorio es distinto para dispositivos móviles.

Un equipo móvil se caracteriza por su escasa memoria y tamaño pequeño. Además, es necesario tener en mente que pueden ocurrir eventos como la entrada de una llamada telefónica o el agotamiento de la batería.

Al principio, se tenía como objetivo crear interfaces de usuario más dinámicas. Sin embargo, este objetivo no se cumplió porque durante el desarrollo de este proyecto inicié mi aprendizaje de la plataforma *Android*.

Durante el desarrollo de este proyecto se inició el desarrollo del servicio web utilizando SOAP. Sin embargo, SOAP es costoso para el dispositivo móvil porque se crean varias instancias de clases que no fueron necesarias para este proyecto al momento de realizar el envío y recepción de información. Considerando esto, posteriormente, el servicio web se desarrolló con REST.

Otro problema que se presentó fue el tratamiento de las imágenes. Al inicio, se consideraba cargar todas imágenes de fondo al mismo tiempo; es decir, en el momento en que el usuario iniciara la aplicación. Esta estrategia generaba una excepción y provocaba el aborto de la aplicación. Después, se hicieron pruebas con distintas imágenes para no agotar la memoria ya que esto causaba la excepción. Al final, se desarrolló un algoritmo que carga las imágenes de fondo de manera dinámica; es decir, se cargan cuando se necesitan. Esto evitó el problema del agotamiento de la memoria.

Se espera que este documento sirva de guía para los desarrolladores de Android. El objetivo es apoyarles a mostrar lo que se puede y no se puede hacer con *Android*. Además, indica al lector las herramientas que puede utilizar para facilitar su trabajo.

10. Perspectivas del proyecto

P-MOSIP está en su primera fase, es un prototipo. Se espera que en un futuro el propio usuario pueda añadir sus propios síntomas y no solo estar restringido a seleccionar únicamente las que el sistema le presente.

También, sería un gran reto dotar a P-MOSIP con la capacidad de estar disponible no solo para *Android* sino también para otros sistemas operativos como *Windows Mobile e iOS*.

Se anima al futuro desarrollador a crear interfaces más dinámicas, con diseños personales y no sólo limitarse a usar las que *Android* nos proporciona por defecto. Es importante poder diseñar íconos propios para una aplicación, así como utilizar animaciones y temas.

Otro aspecto que debe ser mejorado en P-MOSIP es la interacción con la base de datos. Se alienta al desarrollador a definir un mejor diseño de la base de datos.

Por último, se hace una atenta invitación al futuro desarrollador del proyecto P-MOSIP, a dar una detallada lectura a este documento para conocer el uso de las herramientas y de las capacidades de la plataforma *Android*. Si usted desea una referencia amplia sobre *Android* vaya a la referencia [8].

Referencias bibliográficas

[1] D. Costrejon, "*Expediente clínico ECE*", Proyecto terminal de Ingeniería en Computación, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México D.F., 2010.

[2] G. Monroy, *"Reingeniería del sistema de consulta y diagnóstico médico veterinario SICODIVE"*, Proyecto terminal de Ingeniería en Computación, División de CBI, Universidad Autónoma Metropolitana Azcapotzalco, México D.F., 2012.

[3] "*iTriageHealth*" [Online]. Disponible en: https://play.google.com/store/apps/details?id=com.healthagen.iTriage&hl=es_419

[4] "*GoodRx*" [Online]. Disponible en: http://www.goodrx.com/

[5] "*Fooducate*" [Online]. Disponible en: http://www.fooducate.com/

[6] A. T. Jaime, "*Desarrollo de aplicaciones para dispositivos móviles sobre la plataforma Android de Google*", Proyecto de fin de carrera, ingeniería en informática, Universidad Carlos III de Madrid, enero de 2009.

[7] "*Androideity*" [Online]. Disponible en: http://androideity.com/esto-es-androideity/

[8] "*Android Developers*" [Online]. Disponible en: http://developer.android.com/index.html

[9] "*Eclipse ADT*" [Online]. Disponible en: https://developer.android.com/sdk/index.html?hl=i

[10] Antonio J. Martín Sierra, "*Struts*", RA-MA Editorial, Segunda edición, Madrid España, 2007.

[11] "*Android Action Bar Style Generator*" [Online]. Disponible en: http://jgilfelt.github.io/android actionbarstylegenerator/#name=example&compat=holo&theme=light&actionbarstyle=solid&texture=0&hairline=0&neutralPressed=1&backColor=E4E4E4%2C100&secondaryColor=D6D6D6%2C100&tabColor=33B5E5%2C100&tertiaryColor=F2F2F2%2C100&accentColor=33B5E5%2C100&cabBackColor=FFFFFF%2C100&cabHighlightColor=33B5E5%2C100

[12] "*Apache Tomcat*" [Online]. Disponible en: *http://tomcat.apache.org/*

[13] "*MySQL*" [Online]. Disponible en: *http://www.mysql.com/*

[14] "*Jersey*" [Online]. Disponible en: *https://jersey.java.net/*

[15] "*Gson*" [Online]. Disponible en: *https://code.google.com/p/google-gson/*

[16] "*Connector/J*" [Online]. Disponible en: http://dev.mysql.com/downloads/connector/j/

Apéndice A. Listado de parte del código fuente desarrollado

A continuación se lista parte del código desarrollado. Se invita a ver el código fuente completo del proyecto para una mejor comprensión.

Algoritmo para el tratamiento de imágenes.

Este algoritmo está contenido en dos clases.

1. La clase `DecodeSampleBitmap` se encarga de obtener una imagen en miniatura a partir de la imagen original.

```
/**Clase encargada de codificar la imagen, es decir, se busca reducir la calidad
 * de la imagen en busca de un ahorro de memoria.
 * @see BitmapWorkerTask*/
public class DecodeSampleBitmap
{
	/**Calculamos las nuevas dimensiones de la imagen*/
	public static int calculateSampleSize(BitmapFactory.Options options, int
reWidth,int reHeight)
	{
		//Alto y ancho de la imágen en pixeles
		final int height=options.outHeight;
		final int width=options.outWidth;
		int inSampleSize=1;

		if(height > reHeight || width > reWidth)
		{
			final int halfHeight = height / 2;
			final int halfWidth = width / 2;
			//Calculamos el valor mas grande en potencia de dos y lo
guardamos.
			while( ( halfHeight / inSampleSize) > reHeight && ( halfWidth
/ inSampleSize ) > reWidth)
			{
				inSampleSize *= 2;
			}
		}
		return inSampleSize;
	}
	/**Obtenemos una subimagen*/
	public static Bitmap decodeSampledBitmapFromResource(Resources res, int
resId, int reWidth, int reHeight)
	{
		//Hacemos el tratamiento de la imagen pero sin cargarla, es decir,
sin decodificar
		final BitmapFactory.Options options = new BitmapFactory.Options();
		options.inJustDecodeBounds = true;
		BitmapFactory.decodeResource(res, resId, options);
```

```
            //Calculamos cuanto se reducirá el tamaño de la imágen en pixeles.
            options.inSampleSize = calculateSampleSize(options, reWidth,
reHeight);

            //Decodificamos la imagen pero con el tamaño de pixeles calculado.
Obtenemos una imágen degradada
            options.inJustDecodeBounds = false;
            return BitmapFactory.decodeResource(res,  resId, options);

      }
}
```

2. La clase `BitmapWorkerTask`, se encarga de la creación de un hilo. Este hilo está asociado a su vista respectiva `ImageView`. Cuando el hilo termine su tarea y la `ImageView` aún esté disponible, se asocia la imagen correspondiente.

```
/**Esta clase es similar a BitmapWorkerTaskFiguraHumanaHombre
 * @see BitmapWorkerTaskFiguraHumanaHombre*/
public class BitmapWorkerTask extends AsyncTask<Integer, Void, Bitmap>
{
      // Mantenemos una referencia débil.
      private final WeakReference<ImageView> imageViewReference;
      public int data = 0;
      private Resources resource;
      private int height, width;
      // Necesario, ya que la actividad es la que contiene la memoria caché
sobre la
      // que se está trabajando.
      private MainActivity mainActivity;
      // Representa nuestra memoria caché
      private LruCache<String, Bitmap> mMemoryCache;
      public void setLruCache(LruCache<String, Bitmap> lruCache)
      {
            this.mMemoryCache = lruCache;
      }
      public void setActivity(MainActivity mainActivity)
      {
            this.mainActivity = mainActivity;
      }
      public void setResource(Resources r)
      {
            resource = r;
      }
      public void setHeightWidth(int height, int width)
      {
            this.height = height;
            this.width = width;
      }
      public BitmapWorkerTask(ImageView imageView)
      {
            //Usamos una referencia débil para asegurar que el recolector de
            // basura pueda recuperarlo aún cuando el hilo sea cancelado.
```

```
            imageViewReference = new WeakReference<ImageView>(imageView);
      }

      //la tarea la realizamos en segundo plano, es decir, en un hilo.
      @Override
      protected Bitmap doInBackground(Integer... params)
      {
            data = params[0];
            //Decodificamos la imagen
            final Bitmap bitmap=
DecodeSampleBitmap.decodeSampledBitmapFromResource(resource, data, height,
width);
            // Añadimos la imagen a la memoria caché
            mainActivity.addBitmapToMemoryCache(String.valueOf(data), bitmap,
mMemoryCache);
            return bitmap;
      }
      //Cuado termine la tarea en segundo plano y la ImageView esté disponible,
      // se fija la imagen a la ImageView.
      @Override
      protected void onPostExecute(Bitmap bitmap)
      {
            // Si el hilo fué cancelado, no necesitamos mas el bitmap
            if(isCancelled())
            {
                  bitmap = null;
            }
            if(imageViewReference != null && bitmap != null)
            {
                  //Obtenemos la referencia de la ImageView
                  final ImageView imageView = imageViewReference.get();
                  //Obtenemos el bitmap procesado por el hilo asociado
                  final BitmapWorkerTask bitmapWorkerTask =
MainActivity.getBitmapWorkerTask(imageView);
                  if(this == bitmapWorkerTask && imageView != null)
                  {
                        //Establecemos el bitmap a la ImagenView
                        imageView.setImageBitmap(bitmap);
                  }
            }
      }
}
```

Implementación de un “libro”

El siguiente fragmento de código es parte de la clase `MainActivity` y muestra cómo se codifica un “libro” en *Android*.

```
      /**
       * La clase SectionsPagerAdapter será la encarga de proporcionarnos las
páginas
       */
      public class SectionsPagerAdapter extends FragmentPagerAdapter
```

```
{
	/**Constructor*/
	public SectionsPagerAdapter(FragmentManager fm)
	{
		super(fm);
	}

	@Override
	public Fragment getItem(int position)
	{
		if(position==0)
			return FragmentMujerFrente.newInstance();
		return FragmentMujerVuelta.newInstance();
	}

	@Override
	public int getCount()
	{
		// Sólo se crearan 2 páginas.
		return 2;
	}
}
```

Dependiendo en que "página" se encuentre el usuario, el sistema cargará el `Fragment` correspondiente.

Clase ResumenActivity

La clase `ResumenSintomaActivity` contiene el siguiente fragmento de código. Este código es el encargado de cargar los elementos de la lista.

```
…
Final ArrayList<Items_Lista> datos= new ArrayList<Items_Lista>();
…
```

El objeto `datos` es el encargado de mapear los datos leídos de la base de datos a cada miembro de la lista.

```
	Private void cargaDatos()
	{
		if(sintomasPadecidos==null) return;
		for(int k=0;k<sintomasPadecidos.length;k++)
		{
			Integer n=k+1;
			StringstringK=n.toString();

	datos.add(newItems_Lista((Integer.parseInt(sintomasPadecidos[k][0])),//id
del sintoma
```

```
        sintomasPadecidos[k][1],//Nombre del síntoma

        sintomasPadecidos[k][5],//Fecha inicio

        sintomasPadecidos[k][2],// Intensidad

        sintomasPadecidos[k][6],//Fecha término

        sintomasPadecidos[k][9],//Comportamiento

        sintomasPadecidos[k][3],// Descripción

        sintomasPadecidos[k][7],// Hora inicio

        sintomasPadecidos[k][8],//Hora fín
                                                        stringK));
              }
        }
```

A cada elemento [i,j] de la lista le corresponde su propio evento.

```
                                    // Obtenemos el TextView que mostrará
información.
                                    TextView textViewstringNombreSintoma= (TextView)
view.findViewById(R.id.TextViewNombreSintoma);

                                    if(textViewstringNombreSintoma!=null)
                                    {
                                          // recuperamos el objeto/registro
recuperado de la base de datos
                                          Items_Listail=(Items_Lista)entrada;
                                          // Necesitamos una manera de identificar
inequívocamente a cada elemento, esto lo hacemos
                                          // con una etiqueta.
                                          Stringtag= il.getidSintomaRegistrado() +
il.getstringColumna();
                                          // Escribimos información dentro del
TextView

        textViewstringNombreSintoma.setText(((Items_Lista)
entrada).getstringNombreSintoma());
                                          // Asociamos el evento [i,j] al item de
la columna j en la fila i.

        textViewstringNombreSintoma.setOnClickListener(newNombreSintomaOnClickLis
tener());
```

Diseño de la interfaz Resumen de síntomas

Una de las características de las aplicaciones *Android* es que las interfaces de usuario pueden desarrollarse en archivos XML. Por ejemplo, el siguiente código define la interfaz Resumen de la aplicación P-MOSIP.

```
<?xmlversion="1.0"encoding="utf-8"?>

<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
>

<HorizontalScrollView
android:id="@+id/horizontalScrollView1"
android:layout_width="wrap_content"
android:layout_height="match_parent"
>

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

<LinearLayout
android:layout_width="wrap_content"
android:layout_height="match_parent"
android:orientation="vertical"
>

<LinearLayout
android:layout_width="match_parent"
android:layout_height="wrap_content"
      android:background="#24BBDC"
>

<ImageView
android:id="@+id/imageView2"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="15dp"
android:src="@drawable/ic_action_settings"
/>

<ImageView
android:id="@+id/imageView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_upload"
      android:padding="15dp"
```

```
/>

<ImageView
android:id="@+id/imageView3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_sort_by_size"
       android:padding="15dp"
/>

<ImageView
android:id="@+id/imageView4"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_download"
       android:padding="15dp"
/>

<ImageView
android:id="@+id/imageView5"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_forward"
       android:padding="15dp"
/>

<ImageView
android:id="@+id/imageView6"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_chat"
       android:padding="15dp"
/>

<ImageView
android:id="@+id/imageView7"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/ic_action_discard"
       android:padding="15dp"
/>

</LinearLayout>

<FrameLayout
android:layout_width="match_parent"
android:layout_height="match_parent">

<ListView
android:id="@+id/ListViewListado"
android:layout_width="match_parent"
android:layout_height="match_parent"
       android:background="#24BBDC"
>
</ListView>
```

```
</FrameLayout>

</LinearLayout>

</LinearLayout>
</HorizontalScrollView>

</FrameLayout>
```