

Universidad Autónoma Metropolitana

Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Ingeniería en Computación

Reporte Final del Proyecto de Integración

Proyecto Tecnológico

Aplicación Android para gestionar notas médicas

Jonathan Palatto Páez

209330179

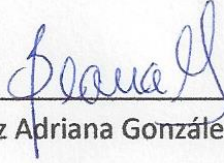
Dra. Beatriz Adriana González Beltrán

Dr. Alejandro Reyes Ortiz

Trimestre: 2016-O

5 de diciembre de 2016

Yo, Dra. Beatriz Adriana Gonzáles 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.



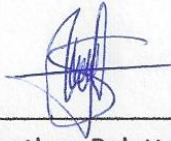
Beatriz Adriana González Beltrán

Yo, Dr. Alejandro Reyes Ortiz, 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.



Alejandro Reyes Ortiz

Yo, Jonathan Palatto Páez, 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.



Jonathan Palatto Páez

Resumen

Los médicos elaboran diferentes tipos de notas médicas de acuerdo al estado de salud del paciente. Un médico debe llenar una nota médica por cada consulta que realice con un paciente. También a un médico le interesa conocer los síntomas que ha tenido el paciente. La originalidad de este proyecto radica principalmente en la construcción de plantillas médicas para que un médico tenga los apartados correspondientes a la nota que desea redactar, el filtrado de las notas y las gráficas de los síntomas.

El objetivo principal de este proyecto fue diseñar e implementar una aplicación móvil en Android para gestionar notas médicas. Los objetivos fueron: 1) Analizar, diseñar y construir el módulo Gestionar notas médicas. 2) Analizar, diseñar y construir el módulo ver y filtrar síntomas. 3) Analizar, diseñar y construir el módulo Graficar síntomas. 4) Diseñar y construir el servicio web de notas médicas. Las aplicaciones que sean desarrolladas no cumple con todos los objetivos de este proyecto terminal.

Como se puede apreciar en los objetivos, el desarrollo de este proyecto tiene dos módulos importantes, el primero es la aplicación móvil con el que va interactuar directamente el médico y el segundo es el servicio web donde se van almacenar y recuperar las notas y los síntomas.

Para el desarrollo de la aplicación se utilizó como metodología el Proceso Unificado. La aplicación sigue una arquitectura Modelo-Vista-VistaModelo. Para su implementación se utilizó Android Studio y las librerías MPAAAndroidChar y ResTemplate. El servidor web utiliza una arquitectura REST y para su implementación se utilizó Netbeans. Para las pruebas se utilizaron celulares con Android 4.4 y 5.1.

Contenido

1. Introducción	7
2. Antecedentes	7
2.1. Proyectos de integración o terminales	7
2.2. Propuestas de proyecto de integración o terminal	7
2.3. Aplicaciones.....	8
3. Justificación	9
4. Objetivos	10
4.1. Objetivo General	10
4.2. Objetivos Específicos.....	10
5. Marco Teórico	10
5.1. Notas Médicas.....	11
5.1.1. Notas de Diagnóstico	11
5.1.2. Notas de Evolución.....	11
5.1.3. Notas Simple	11
5.2. Android Studio	11
5.2.1. Empezar un proyecto	11
5.2.2. AndroidManifest	11
5.2.3. Actividad.....	12
5.2.4. Fragmentos.....	13
5.2.5. SQLITE.....	14
5.3. RESTful.....	15
6. Desarrollo	15
6.1. Diseño del sistema	15
6.1.1. Diagramas de casos de uso	15
6.1.2. Casos de uso de Texto	16
6.1.3. Diagramas de Clases.....	18
6.1.4. Arquitectura del sistema	22
Arquitectura de la aplicación móvil.....	22

Arquitectura del Servicio Web	24
6.1.5. Estructura de la base de datos	24
6.1.6. Uso del Sistema	26
Iniciar la aplicación	26
Crear nueva nota	27
Actualizar Pacientes	28
6.2. Hardware y Software.....	30
6.2.1. Tecnología para el desarrollo de la aplicación	30
6.2.2. Tecnología para la instalación y puesta en marcha de la aplicación.....	31
7. Resultados	31
8. Análisis y discusión de resultados	31
9. Conclusiones.....	31
10. Perspectivas del Proyecto	32
11. Bibliografía	33
Diagrama 1. Diagrama casos de uso de la aplicación móvil.....	16
Diagrama 2. Diagrama casos de uso del servicio web.	16
Diagrama 3. Diagrama de Clases del modelo.....	19
Diagrama 4. Diagrama de clase de VistaModelo.	22
Diagrama 5. Entidad-Relación de la base de datos de la aplicación.	26
Ilustración 1. Pirámide del ciclo de vida.....	12
Ilustración 2. Ejemplo de fragmentos.	13
Ilustración 3. Ciclo de vida de un fragmento.	14
Ilustración 4. Módulos de MVVM.	22
Ilustración 5. Carpetas de la aplicación.....	23
Ilustración 6. Modelo de dos capas.....	24
Tabla 1.Caso de uso de texto "Nueva Nota".	17
Tabla 2. Caso de uso de texto "Actualizar pacientes"	17
Pantalla 1. Pantallas de la aplicación ColorNote Block de notas.	8
Pantalla 2. Pantallas de la aplicación ManageMyPain Pro.....	9
Pantalla 3. Pantallas de la aplicación Ofimedic.	9
Pantalla 4. Inicio de la aplicación.	26

Pantalla 5. Notas médicas.	26
Pantalla 6. Nueva nota.	27
Pantalla 7. Nueva nota simple diagnóstico.	28
Pantalla 8. Guardar Nota.....	28
Pantalla 9. Conectando con el servicio web.....	29
Pantalla 10. Éxito al actualizar pacientes.	29
Pantalla 11. Error de conexión.....	30

1. Introducción

El objetivo de este reporte es explicar de forma general los antecedentes y justificación del desarrollo de mi aplicación móvil, así como los objetivos, el marco teórico y la arquitectura que se utilizaron para el desarrollo de la aplicación. Mostrar y explicar diagramas, casos de uso y pantallas que se obtuvieron en el desarrollo. Explicar y analizar los resultados que obtuvieron durante la implementación de la aplicación. Y al final se dicen las conclusiones y perspectivas de este proyecto de integración.

2. Antecedentes

2.1. Proyectos de integración o terminales

-“Prototipo de una aplicación móvil para la gestión de síntomas de un paciente” [1] es un proyecto de integración que se elaboró en la UAM en trimestre 2014 otoño. Se trata de una aplicación implementada en Android, donde se puede añadir síntomas e información del síntoma, como intensidad, duración, descripción lugar del cuerpo donde está el síntoma entre otros. La información del síntoma se guarda en una base de datos que se accede a ella a través de un servicio web.

Esta aplicación guarda los síntomas del paciente, más no los enseña al médico, mi aplicación muestra los síntomas por cada paciente, poder filtrarlos y mostrarle una gráfica por cada síntoma.

2.2. Propuestas de proyecto de integración o terminal

-“Sistema para identificar la ubicación e intensidad de los síntomas de un paciente a partir de notas médicas” [2] es un proyecto que toma una nota médica, previamente digitalizada, y le aplica algunas técnicas de algoritmos y minería de datos, para reconocer los síntomas su ubicación e intensidad, luego los almacena en una base de datos.

La principal diferencia, es que el proyecto busca los síntomas y datos de ellos para almacenarlos, y descarta la nota médica, mi aplicación puede crear notas médicas, editarlas, almacenarlas y recuperarlas.

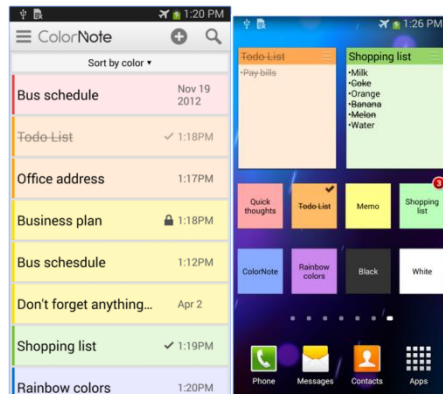
-“Sistema para la identificación de fechas de los síntomas de un paciente a partir de notas médicas” [3] es un proyecto que almacena notas médicas, luego las recupera para extraer fechas, síntomas y otros datos de la nota, tiene un módulo para relacionar las fechas con los síntomas y quitar ambigüedad, y al final los almacena en una base de datos.

Aunque este proyecto si almacena las notas médicas, no es para mostrárselas al médico, objetivo que si tiene mi aplicación. Otra diferencia es que el proyecto no tiene ninguna organización para las notas médicas, mi aplicación las organizara de acuerdo al paciente.

2.3.Aplicaciones

-ColorNote Bloc de notas [4] es una aplicación que almacena notas en el dispositivo, se pueden crear y programar la hora y fecha para que funcionen como recordatorios, también se pueden marcar las notas con colores, darles contraseña, buscarlas, enviarlas y crear listas. La Pantalla 1 muestra algunas pantallas de la aplicación ColorNote Block de notas.

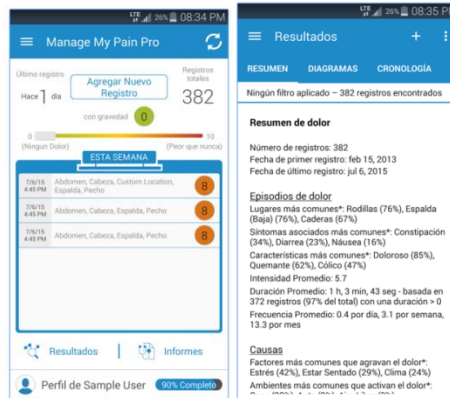
La principal diferencia es que no está enfocada a los médicos, las notas no las puede agrupar y solo las almacena en el mismo dispositivo, si las quisiera ver otro médico tendría que enviarlas una por una.



Pantalla 1. Pantallas de la aplicación ColorNote Block de notas.

-ManageMyPain Pro [5] es una aplicación que permite administrar el dolor. Tiene como objetivo mostrar un informe de los dolores crónicos que presenta el paciente. El paciente es el que interactúa con la aplicación, y el informe se puede imprimir o puede ser enviado al doctor por correo electrónico. La Pantalla 2 muestra algunas pantallas de la aplicación ManageMyPain Pro.

La principal diferencia entre la aplicación y ManageMyPain es la forma de mostrar los síntomas. ManageMyPain solo muestra un informe de los síntomas, en mi aplicación el medico puede filtrar los síntomas y va a mostrar una gráfica por cada síntoma.



Pantalla 2. Pantallas de la aplicación ManageMyPain Pro.

-Ofimedic [6] es una empresa que se dedica a desarrollar software para el médico en gestión de consultas. Tiene una versión de escritorio y una versión móvil, aunque la sincronización de los datos para las dos versiones requiere de un pago extra. Algunas de las funciones de su versión móvil son: gestión de agendas, configuración de agendas por facultativos, determinación de horarios clínicos, generación de notas, situación de pacientes. La Pantalla 3 muestra algunas pantallas de la aplicación Ofimedic.

La diferencia principal, es que mi aplicación se enfoca en las notas médicas. Mi aplicación ofrece plantillas las cuales ayuda al médico a interactuar mejor con el paciente, Ofimedic no indica que ofrece plantillas.



Pantalla 3. Pantallas de la aplicación Ofimedic.

3. Justificación

Actualmente el desarrollo de aplicaciones médicas ha ayudado a los médicos, especialmente a gestionar información relevante de cada paciente [1].

Las aplicaciones de notas que se han desarrollado son solo notas para una persona y se guardan en el dispositivo [4]. Hay aplicaciones que están enfocadas a los médicos y tienen notas médicas, pero no contienen plantillas.

Con las plantillas se ayudara al médico a que tenga los apartados necesarios para describir ampliamente la información del paciente, de acuerdo a la situación.

La aplicación almacena las notas médicas en un servicio web y solo se requiere estar conectado a internet. Esto con el fin de que los médicos, por medio de la aplicación, puedan tener las notas médicas disponibles en todo momento y actualizadas, añadiendo nuevas notas o editándolas.

También en la aplicación se ven y se filtran los síntomas de cada paciente, así como una gráfica por cada síntoma, con estas herramientas ayudaran al médico tener una visión más general de los síntomas del paciente.

4. Objetivos

4.1. Objetivo General

Diseñar e implementar una aplicación móvil en Android para gestionar notas médicas.

4.2. Objetivos Específicos

1. Analizar, diseñar y construir el módulo Gestionar paciente.
2. Analizar, diseñar y construir el módulo ver y filtrar síntomas.
3. Analizar, diseñar y construir el módulo Graficar síntomas.
4. Analizar, diseñar y construir el módulo Gestionar notas médicas.
5. Analizar, diseñar y construir el módulo Plantillas de notas médicas.
6. Construir la base de datos para almacenar los pacientes y las notas médicas.
7. Diseñar y construir el servicio de notas médicas.
8. Realizar y documentar pruebas del funcionamiento correcto de la aplicación.

5. Marco Teórico

El marco teórico esta está dividida en tres secciones. La primera sección corresponde a los conceptos relacionados con las notas médicas. La segunda sección describe los conceptos básicos para poder desarrollar una aplicación Android. La última sección, aborda los conceptos relacionados con el desarrollo de servicio web.

5.1. Notas Médicas

Los médicos elaboran notas de diagnóstico, notas de evolución o notas simples de acuerdo al estado de salud del paciente, tales notas forman parte del expediente del paciente.

5.1.1. Notas de Diagnóstico

Una Nota de diagnóstico tiene tres secciones: Diagnóstico, Tratamiento y Observaciones. La sección de Diagnostico se encuentra la descripción de la enfermedad del paciente, la sección de Tratamiento describe los medios que se utiliza para aliviar o curar la enfermedad descrita, y en la sección Observaciones se encuentra información adicional relacionada con la enfermedad del paciente.

5.1.2. Notas de Evolución

Una nota de evolución contiene cuatro secciones: Subjetivo, Objetivo, Análisis y Plan. La sección Subjetivo se encuentra la información proporcionada por el paciente o sus familiares, la sección Objetivo tiene la información obtenida después de haber realizado la exploración física y los resultados de exámenes de laboratorio, la sección Análisis es donde se anotan las conclusiones o ideas diagnósticas y/o terapéuticas del estado actual del paciente, y la sección Plan se apuntan el manejo subsecuente del paciente.

5.1.3. Notas Simple

Una nota simple solo contiene una sección, donde se encuentra información importante relacionada con el paciente.

5.2. Android Studio

Android Studio es un entorno de desarrollo de aplicaciones para Android, utiliza el lenguaje de programación java. Está basado en el software IntelliJ IDEA de JetBrains. Se puede desarrollar para todo los dispositivos Android, ofrece un emulador de aparatos móviles Android, es multiplataforma y es la herramienta oficial para utilizar aplicaciones de google.

5.2.1. Empezar un proyecto

Para empezar un proyecto en Android Studio solo se debe elegir un nombre, una ubicación donde se va aguardar el proyecto, una versión de Android y una plantilla. Android Studio creara e inicializara todos los archivos necesarios para que el proyecto ya funcione en Android.

5.2.2. AndroidManifest

Es un archivo xml que describe la información esencial acerca de la aplicación. Las más importantes son:

- Los componentes de la aplicación, las actividades, los servicios y recursos.
- La actividad principal con que se inicia la aplicación.
- Los permisos de la aplicación

- La versión mínima de Android que se necesita para que la aplicación funcione.

5.2.3. Actividad

Una aplicación Android está conformada de una o más Actividades. Una actividad es un componente que con el usuario puede interactuar, con el fin de solicitar una tarea a la aplicación. Cada actividad tiene un ciclo de vida, que son unos conjuntos de estados que están asociados a métodos, una forma de visualizar el ciclo de vida es por medio de una pirámide tal como se muestra en la Ilustración 1.

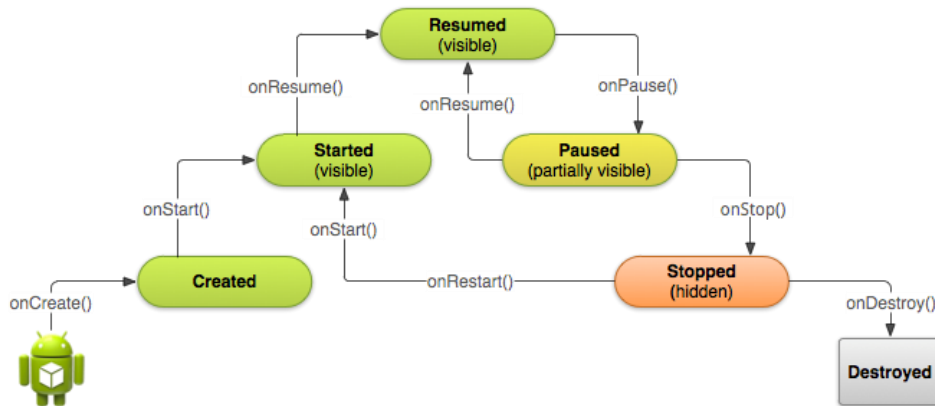


Ilustración 1. Pirámide del ciclo de vida.

Ejemplo: si una actividad se encuentra en el estado `Started` y se va a pasar al estado `Resumed`, entonces debe ejecutar el método `onResume()`.

Para utilizar una actividad se debe extender de la clase `Activity`. Los métodos de la pirámide ya tienen una implementación en la clase `Activity`, para utilizarlos se deben sobre escribir los métodos y ejecutar la implementación.

Al iniciar una aplicación, el sistema operativo Android, carga la actividad principal y la pone en el estado `Started`, para llegar a este estado primero tuvo que estar en `Created`.

Una aplicación puede tener varias razones para pasar a otro estado, las principales son:

- El usuario cerró la aplicación: pasa la actividad a un estado `Destroyed`.
- El usuario minimiza la aplicación: pasa la actividad a un estado `Paused`.
- Una actividad A quiere que se muestre una actividad B: A pasa a un estado `Paused` y B pasa a un estado `Started`.
- Cuando actividad B es finalizada: B pasa a un estado `Destroyed` y A pasa a un estado `Resumed`.

Un desarrollador Android no tiene todos los privilegios para iniciar una actividad, es el sistema operativo Android el único que puede crear las actividades y las pone en el estado `Started`.

Para iniciar una actividad se debe utilizar la clase `Intent`, donde se le indica que actividad vamos a inicializar, y se manda a llamar al método `startActivity()` de un objeto de la clase `Context`

Si una actividad A inicializa a una actividad B, la actividad A se queda guardada en la pila de actividades. La pila de actividades sirve para regresar entre las actividades.

A las actividades se debe implementar el método `onCreate()`, donde se especifica la interfaz del usuario. Android Studio ofrece una herramienta de diseño de interfaces, el cual genera un archivo xml. Para que la actividad interprete el archivo xml se utiliza al método `setTheme()`.

5.2.4. Fragmentos

Un fragmento es una actividad que solo ocupa el espacio de pantalla designado para él y puede tener comunicación con otro fragmento. Ejemplo: en la Ilustración 2 hay una actividad, que contiene dos fragmentos, el fragmento A hay una lista y en el fragmento B contiene un cuadro de texto, cuando el usuario seleccione un elemento de la lista, entonces el fragmento B muestra el contenido de la misma.

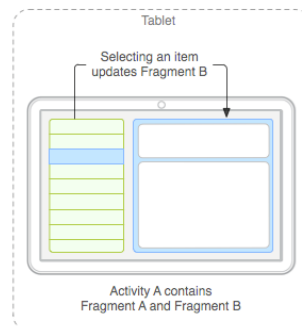


Ilustración 2. Ejemplo de fragmentos.

Una ventaja obtiene al utilizar fragmentos es la facilidad de añadir nuevos fragmentos a la actividad. En otras palabras se le puede añadir funcionalidades a la aplicación sin alterar lo existente.

También hay interfaces que solo se pueden programar con fragmentos como la interface de tipo `TabHost`.

Las características más importantes de los fragmentos son:

- Un fragmento no puede existir si no tiene una actividad que lo contenga.
- Un fragmento también puede contener uno o varios fragmentos.
- Cada fragmento tiene su propio ciclo de vida como en la Ilustración 3.

A diferencia de una actividad, el método `onCreate()` no construye la vista que se va a mostrar al usuario. Se construye hasta que una actividad mande a llamar al método `onCreateView()` y le regresa la vista que construyó.

Una interfaz que contenga fragmentos debe estar dividida por `FrameLayout`, cada `FrameLayout` debe estar nombrada por un id.

Existen dos maneras para que una actividad inicie un fragmento.

- En la interfaz de la actividad, busca una clase que este nombrada igual que su `FrameLayout` y la inicia (la clase debe ser un fragmento).
- Por medio de una transacción donde se indica el id del `FrameLayout` y cual clase es el fragmento.

Si una actividad es destruida se van a destruir los fragmentos que contenga. Así mismo si la actividad es pausada se van a pausar los fragmentos.

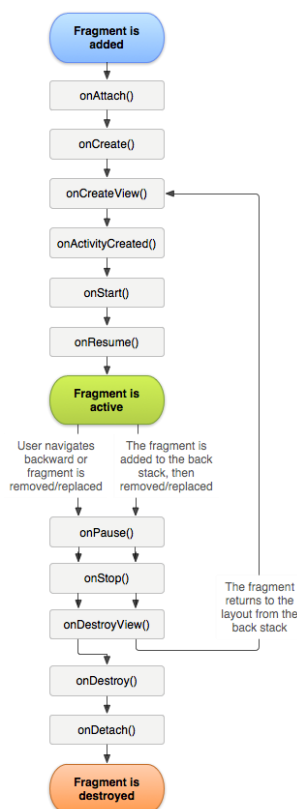


Ilustración 3. Ciclo de vida de un fragmento.

5.2.5. SQLITE

Android Studio tiene la clase `sqlite` que utiliza el lenguaje `sql` para construir una base de datos entidad-relación. Contiene métodos de consulta y escritura hacia la base de datos.

Un objeto de la clase Context, muestra una ruta en el dispositivo Android donde permite guardar datos.

5.3.RESTful

REST es una arquitectura que permite a un servidor ofrecer recursos por medio de la red. Para acceder a los recursos se utiliza el protocolo HTTP [7].

A la arquitectura REST se le llama RESTful cuando cumple los siguientes 4 principios:

- El servidor no mantiene un estado: El cliente es el encargado de saber que recurso va utilizar. El servidor no mantiene ninguna información del cliente.
- Se utilizan los métodos HTTP: para acceder a un recurso se debe indicar por cual método se acceder, los más importantes son PUT, GET, SET, y DELETE que corresponden a crear, leer, actualizar y borrar respectivamente. Esto es para mantener las operaciones bien definidas.
- Cada recurso es direccionable únicamente a través de su URI [8].
- La transmisión y recepción de datos debe tener un formato definido en HTTP: los formatos más comunes son: HTML, XML, JSON. El formato de la transmisión no necesariamente debe ser el mismo que el de la recepción.

6. Desarrollo

Para el desarrollo de esta aplicación móvil y el servicio web se utilizó la metodología UP (Proceso Unificado).

A continuación se describen las herramientas que se utilizaron para el desarrollo.

6.1.Diseño del sistema

6.1.1. Diagramas de casos de uso

En el Diagrama 1 representa al diagrama de caso de usos de la aplicación móvil, donde se muestran dos actores. El primer actor es el médico que interactúa con las funcionalidades: gestionar notas médicas, ver y filtrar síntomas y graficar síntomas. El segundo actor es el servicio web que utiliza las funcionalidades: actualizar síntomas, actualizar notas médicas e insertar pacientes.

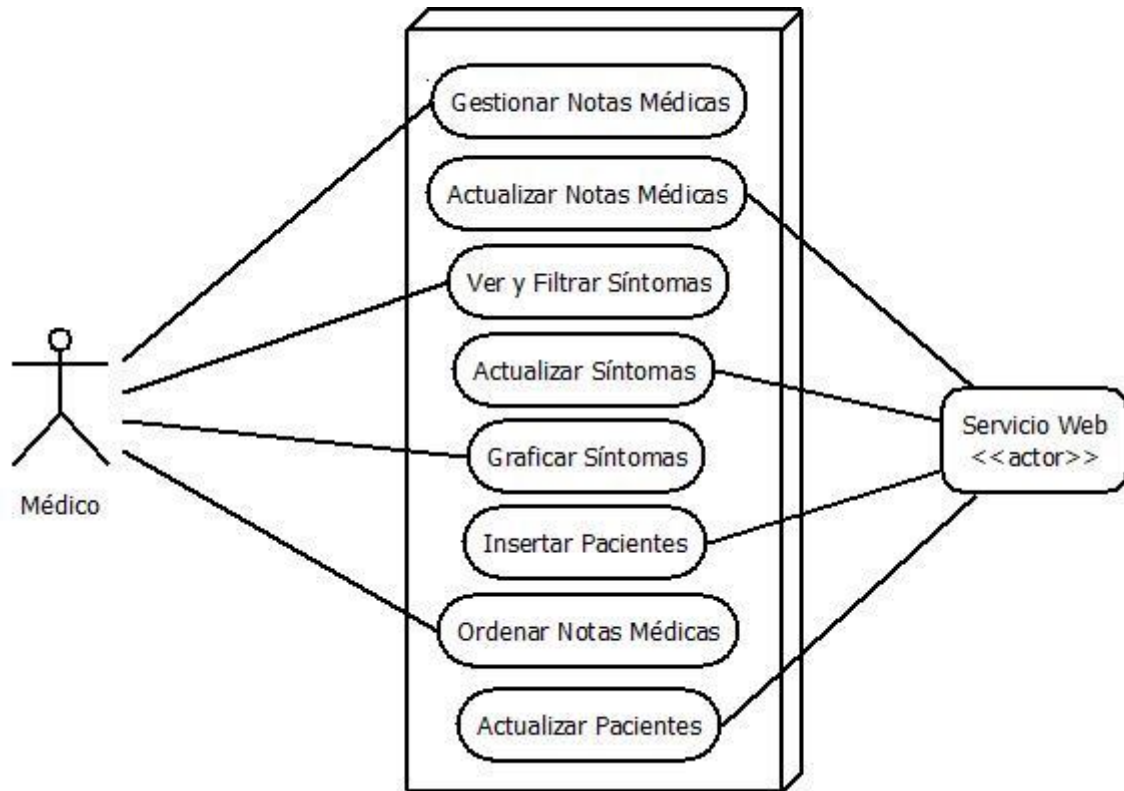


Diagrama 1. Diagrama casos de uso de la aplicación móvil.

En el Diagrama 2 representa al diagrama de caso de usos del servicio web, donde una o varias aplicaciones pueden interactuar con las funcionalidades.

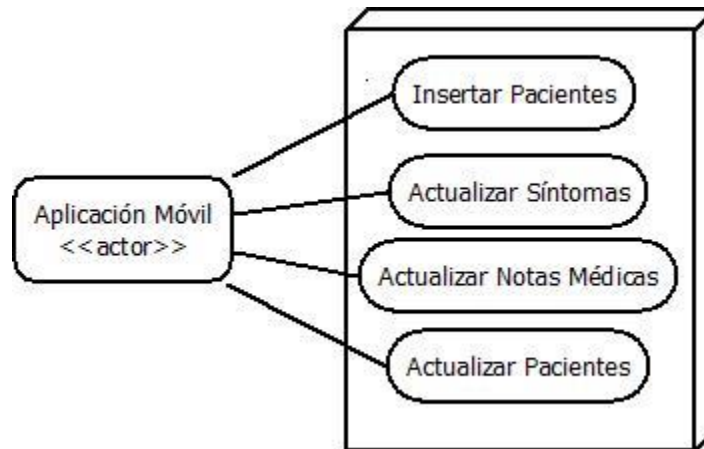


Diagrama 2. Diagrama casos de uso del servicio web.

6.1.2. Casos de uso de Texto

La Tabla 1 muestra el caso de uso de texto "Nueva Nota".

Actores	Médico, Aplicación Móvil (Sistema).
Condición	Estar habilitado el botón de <i>Nueva nota</i> .

Flujo Principal	<ol style="list-style-type: none"> 1. El médico presiona el botón <i>Nueva Nota</i>. 2. El sistema desplegar un menú, con los tipos de notas médicas. 3. El medico selecciona un tipo de nota. 4. El sistema crea una nota vacía los campos correspondientes, de acuerdo al tipo de nota seleccionado. 5. El Sistema deshabilita los botones de <i>Ordenar Por</i>, <i>Nueva Nota</i> y <i>Editar Nota</i>, y habilita los botones de <i>Guardar Nota</i>, <i>Eliminar Nota</i> y <i>Cancelar Nota</i>. 6. El sistema deshabilita la <i>Lista de las Notas Médicas</i>. 7. El sistema cambia campos para que puedan ser editados. 8. El médico edita los campos de la nota médica. 9. El médico presiona el botón <i>Guardar Nota</i>. 10. El sistema guarda la nota creada. 11. El sistema le indica a la nota que guarde los cambios. 12. El sistema cambia los campos para que no puedan se editados. 13. El Sistema habilita los botones de <i>Ordenar Por</i>, <i>Nueva Nota</i> y <i>Editar Nota</i>, y deshabilita los botones de <i>Guardar Nota</i>, <i>Eliminar Nota</i> y <i>Cancelar Nota</i>. 14. El sistema habilita la <i>Lista de las Notas Médicas</i>.
Flujo Alternativo	<ol style="list-style-type: none"> 9.1.1. El médico presiona el botón <i>Cancelar Nota</i>. 9.1.2. El sistema cambia los campos para que no puedan se editados. 9.1.3. El Sistema habilita los botones de <i>Ordenar Por</i>, <i>Nueva Nota</i> y <i>Editar Nota</i>, y deshabilita los botones de <i>Guardar Nota</i>, <i>Eliminar Nota</i> y <i>Cancelar Nota</i>. 9.1.4. El sistema habilita la <i>Lista de las Notas Médicas</i>. 9.2.1. El médico presiona el botón <i>Eliminar Nota</i>. 9.2.2. El sistema muestra una ventana de confirmación. 9.2.3. El médico acepta la eliminación de la nota. 9.2.4. El sistema cambia los campos para que no puedan se editados. 9.2.5. El Sistema habilita los botones de <i>Ordenar Por</i>, <i>Nueva Nota</i> y <i>Editar Nota</i>, y deshabilita los botones de <i>Guardar Nota</i>, <i>Eliminar Nota</i> y <i>Cancelar Nota</i>. 9.2.6. El sistema habilita la <i>Lista de las Notas Médicas</i>. 9.2.7. El sistema muestra la primera nota de la <i>Lista de las Notas Médicas</i>. 9.2.3.2. El médico no acepta la eliminación de la nota médica. 9.2.3.3. Se continúa en el flujo principal desde el punto 8.

Tabla 1.Caso de uso de texto "Nueva Nota".

La Tabla 1 muestra el caso de uso de texto "Actualizar Pacientes".

Actores	Servicio Web (Servicio), Aplicación Móvil (Sistema).
Flujo principal	<ol style="list-style-type: none"> 1. El servicio recibe los pacientes registrados del sistema. 2. El servicio filtra los pacientes y los envía al sistema. 3. El servicio recibe notificación que envíen las notas médicas. 4. El servicio envía las notas médicas. 5. El servicio recibe notificación que envíen los síntomas. 6. El servicio envía los síntomas.

Tabla 2. Caso de uso de texto "Actualizar pacientes"

6.1.3. Diagramas de Clases

A continuación se describen las clases que corresponden al **Modelo** del sistema y en el Diagrama 3 se muestran cómo se relacionan.

Pacientes: Contiene todos los pacientes registrados en el sistema.

Paciente: Contiene al paciente que se está consultando e información del paciente, su nombre, folio, notas médicas y síntomas.

Sintomas: Contiene todos los síntomas de paciente, los cuales están separados por el nombre del síntoma, en esta clase contiene un método que se puede conocer los nombres de los síntomas y otro que recupera los síntomas de acuerdo al nombre.

Sintoma: Contiene toda la información de síntoma.

OrdenarporTipo: Esta clase ordena los índices para que las notas queden ordenadas por los tipos de notas.

OrdenarporFecha: Esta clase ordena los índices para que las notas queden ordenadas por la fecha menor.

Nota: Contiene todas las notas del paciente, tiene como atributos unos índices para saber cómo ordenar las notas. Hay métodos que regresan títulos que corresponden a los títulos de las notas, estos métodos se implementaron porque la interfaz los requería.

Clase abstracta Nota: Contiene la información mínima y los métodos mínimos que necesita una nota. Algunos métodos requieren como parámetros unos objetos que solo los tiene Android Studio, esto hace que el **Modelo** este acoplado a la plataforma pero en la implementación se reduce mucho el trabajo y hace más entendible el código.

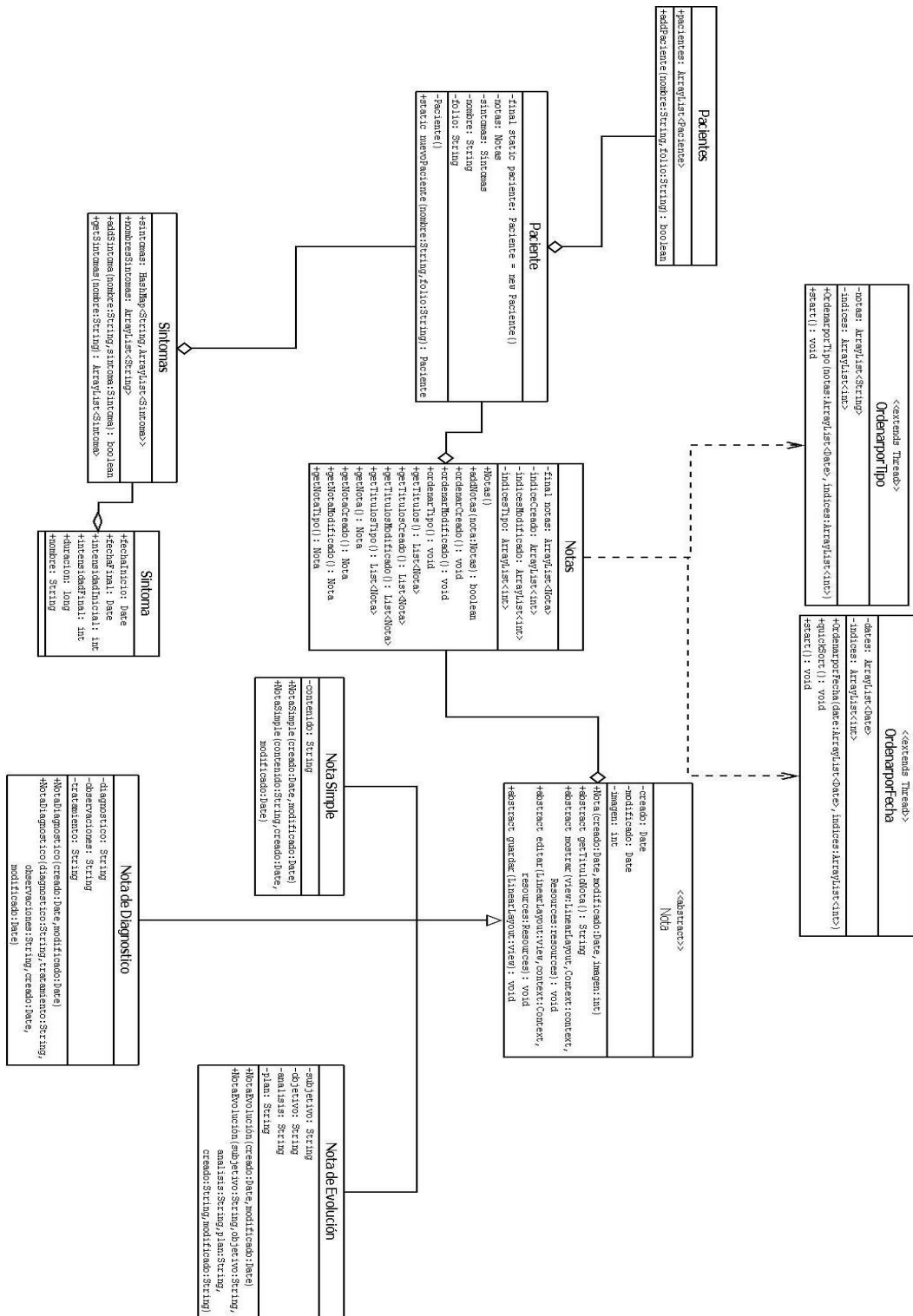


Diagrama 3. Diagrama de Clases del modelo.

Las Clases `NotadeEvolucion`, `NotaSimple` y `NotadeDiagnostico` contienen información específica de su nota y dos métodos constructores. Uno se utiliza cuando se crea una nota vacía y el otro cuando la nota ya contiene información.

A continuación se describen clases **VistaModelo** y en el Diagrama 4 se muestran cómo se relacionan.

MainActivity: Esta clase es la que se muestra al Médico cuando se abre la aplicación. Tiene como objetivos mostrar los pacientes y filtrarlos por nombre o folio, contener crear y ejecutar las clases para la conexión y transmisión de datos con el servicio web.

ExpedienteMedico: Se dio ese nombre a la Clase porque contiene una TabHost, la cual se puede ir añadiendo nuevas pestañas y se puede ir añadiendo información al paciente (igual que los expedientes médicos de un paciente). El objetivo de esta clase es tener las pestañas de la información del que se puede mostrar al paciente. En este proyecto se trabajaron tres pestañas (notas, Síntomas y Graficar Síntomas).

BarraDeMenuNotas: Tiene como objetivo gestionar las notas médicas (Añadir, editar, eliminar y cancelar).

NotasFragment: Tiene como objetivo poder mostrar una lista de notas y el contenido de una nota al mismo tiempo.

NotaContenido: Tiene como objetivo mostrar información de una de las notas y poder editarlas.

MenuNotas: Tiene como objetivo mostrar una lista de todas las notas y ordenar la lista.

DatosSintomas: Tiene como objetivo poder mostrar una lista y al mismo tiempo unos botones al mismo tiempo.

TablaSintomas: Tiene como objetivo mostrar una lista de los síntomas y ordenar la lista.

Las clases que extienden de `Java.lang.thread` las utiliza la clase `TablaSintomas` y tiene como objetivo ordenar los síntomas

GraficasSintomasFragment: Tiene como objetivo poder mostrar una lista y una gráfica al mismo tiempo. También poder cambiar la lista y grafica por otras.

ComparacionSintomas: Tiene como objetivo crear una gráfica de dispersión.

GraficaSintomaSintomas: Tiene como objetivo crear una gráfica de líneas.

Diagrama 4. Diagrama de clase de VistaModelo.

Las clases que extienden de `android.widget.AdapterView` se utilizan para crear las listas para la interfaz.

Las clases que extienden de `android.os.AsyncTask` se utilizan para la comunicación y transferencia de datos con el servicio web.

6.1.4. Arquitectura del sistema

Arquitectura de la aplicación móvil.

La aplicación móvil fue implementada con el patrón de arquitectura **Modelo-Vista-VistaModelo** (MVVM).

El patrón de arquitectura (MVVM) consta de tres módulos:

- **Vista:** Es el módulo visual de nuestra aplicación, no teniéndose que ocupar en ningún momento en el manejo de datos.
- **Modelo:** Es el módulo que se encarga de mantener la lógica de negocio.
- **VistaModelo:** Es el módulo que se encarga de transmitirle al **Modelo** la peticiones del usuario, recuperar los datos del **Modelo**, procesarlos (si es que se requiere) y notificar a la vista los cambios

La Ilustración 4 muestra de manera gráfica cómo interactúan los módulos.

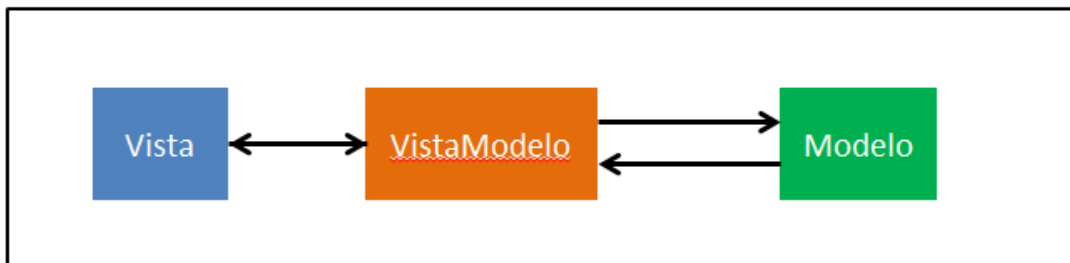


Ilustración 4. Módulos de MVVM.

Como muestra Ilustración 4, el módulo **VistaModelo** es el encargado de establecer la comunicación entre el **Modelo** y la vista.

Es importante reconocer que el módulo **VistaModelo** está fuertemente acoplado a la vista.

En los anteriores diagramas de clases se muestra claramente que clases corresponden al módulo del **Modelo** y **VistaModelo**. Las interfaces representan el módulo de vista, en esta aplicación son algunos archivos con extensión XML.

La Ilustración 5 es una imagen tomada de Android Studio, de cómo están organizadas las carpetas de la aplicación móvil.

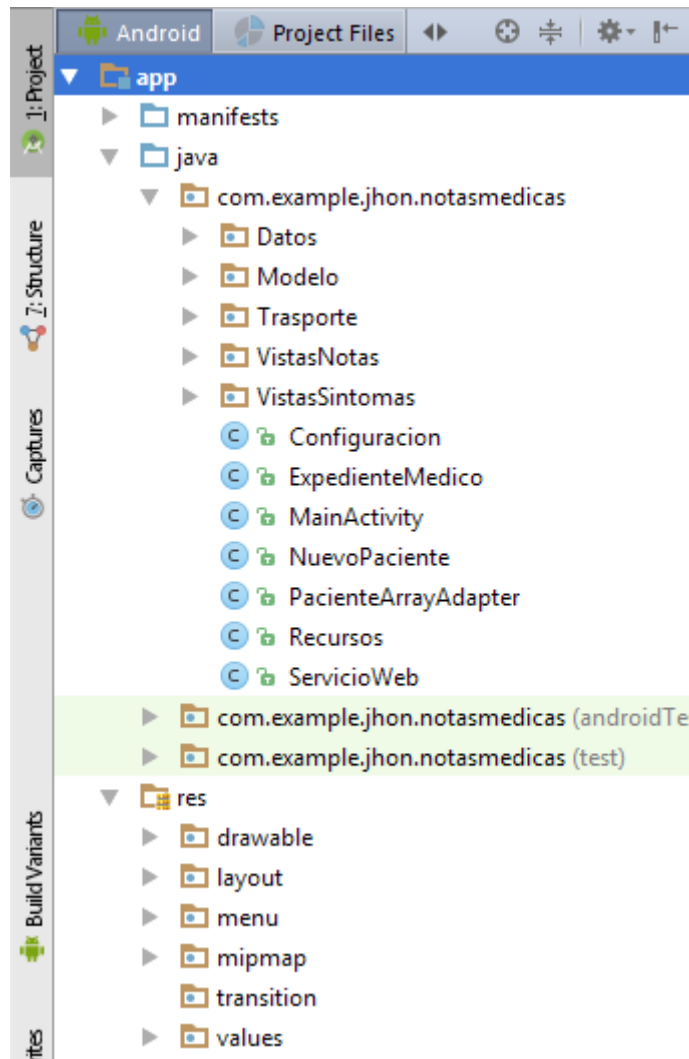


Ilustración 5. Carpetas de la aplicación.

Todas las clases que corresponden con el módulo **Modelo** se encuentran en la carpeta `app/java/com.example.jhon.notasmedicas/Modelo`.

Los archivos que corresponden al módulo **Vista** se encuentran en la carpeta `app/res/layout`.

Del módulo **VistaModelo** son las clases `ExpedienteMedico`, `NuevoPaciente`, `MainActivity` y `PacienteArrayAdapter` y las carpetas `VistasNotas` y `VistasSintomas` que están ubicadas en `app/java/com.example.jhon.notasmedicas`.

La carpeta `app/java/com.example.jhon.notasmedicas/trasporte` contiene todas las clases para la transmisión de datos hacia el servicio.

La carpeta `app/java/com.example.jhon.notasmedicas/Datos` contiene clases para guardar y recuperar los datos almacenados de manera local

La carpeta `app/java/com.example.jhon.notasmedicas/drawable` contiene imágenes que utiliza la aplicación.

La carpeta `app/java/com.example.jhon.notasmedicas/menú` contiene archivos xml, que se utilizan para construir menús en la aplicación.

Arquitectura del Servicio Web

Para la conexión con el servicio web se implementó una arquitectura de dos capas como se muestra en la Ilustración 6.

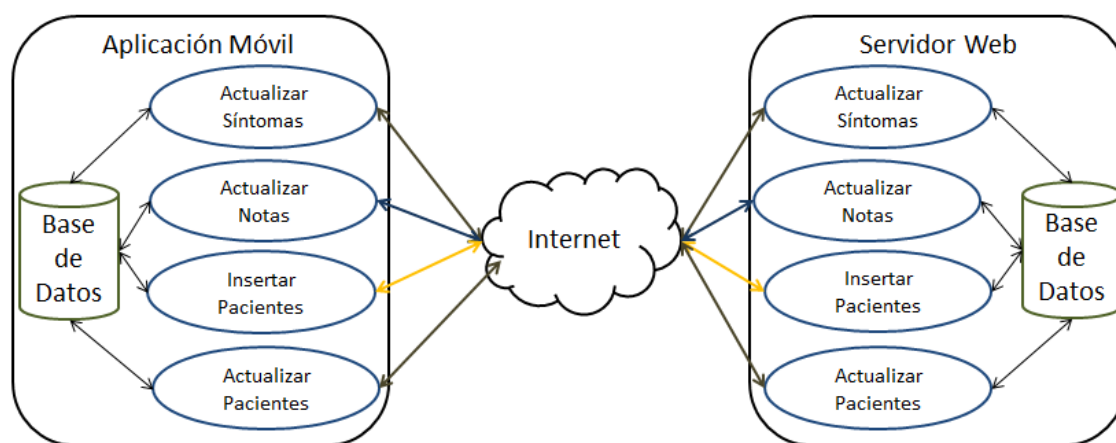


Ilustración 6. Modelo de dos capas.

La primera capa consta de la aplicación y la segunda de servicio web, que se comunican a través del internet.

Cada capa tiene su propia base de datos. El del servicio web guarda los síntomas y notas médicas de todos los pacientes registrados, y el de la aplicación solo guarda los síntomas y notas médicas de algunos pacientes.

Para la comunicación entre las dos capas se utiliza el protocolo HTTP.

6.1.5. Estructura de la base de datos

En el proyecto se tienen dos estructuras de datos, una se encuentra instalada en la aplicación y la otra estructura se encuentra instalada en el servicio web.

En la aplicación se utilizó la clase `sqlite` y la ruta que ofreció el objeto `Context` para instalar una base de datos entidad-relación, esquematizada en el Diagrama 5.

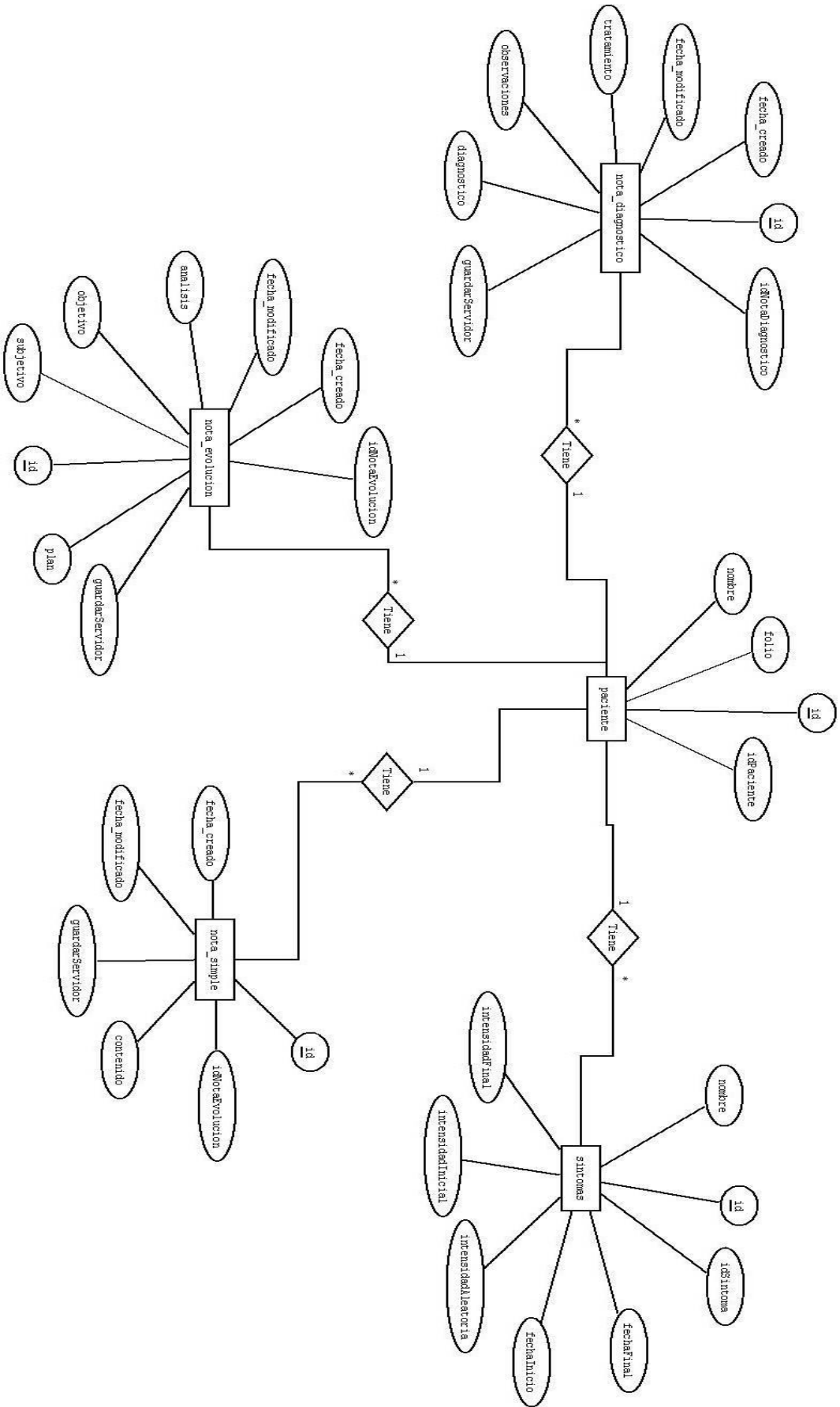
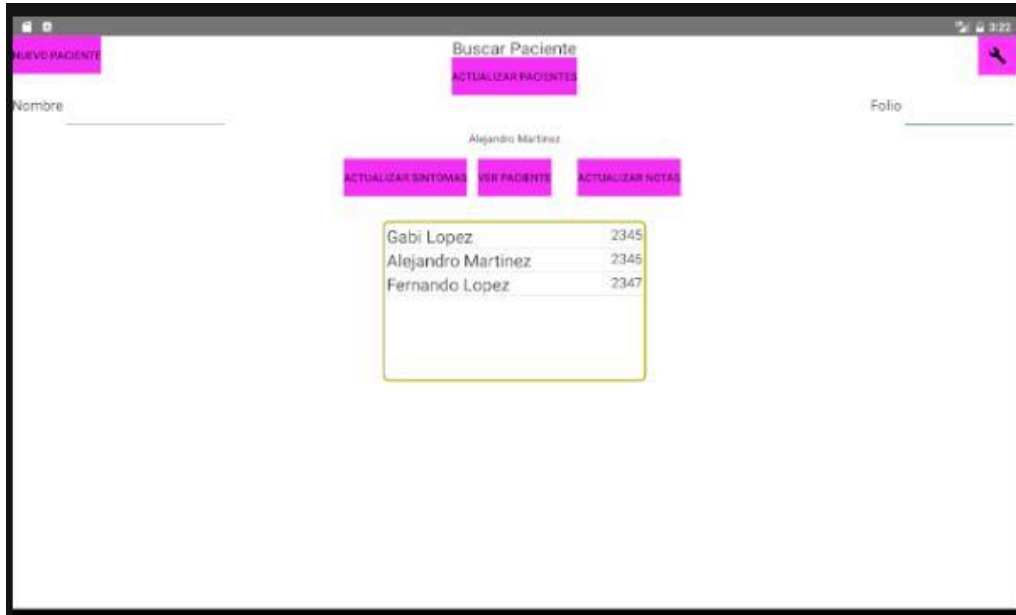


Diagrama 5. Entidad-Relación de la base de datos de la aplicación.

6.1.6. Uso del Sistema

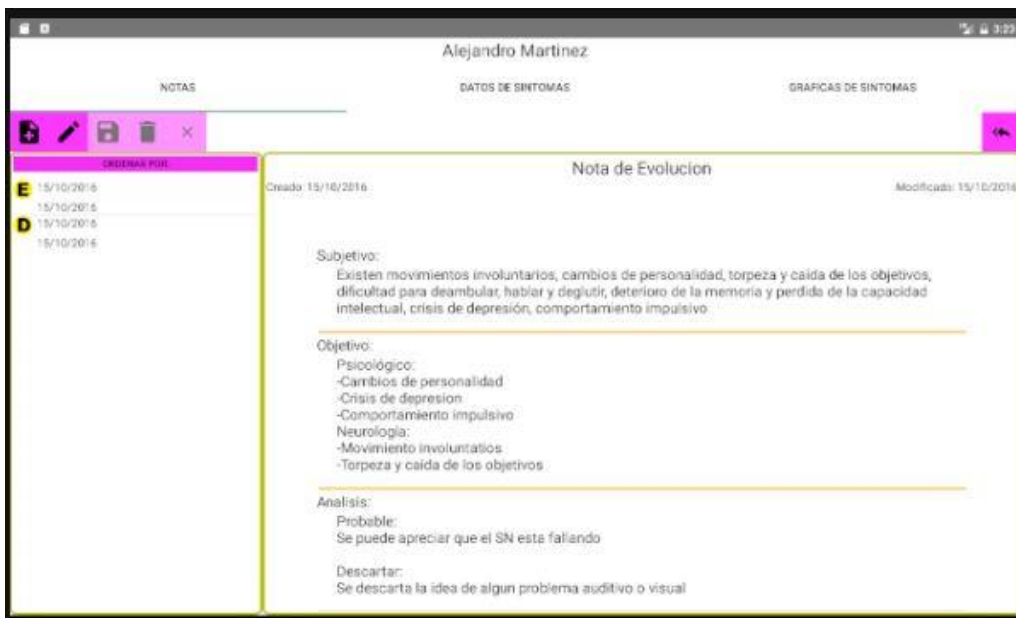
Iniciar la aplicación

Al iniciar la aplicación se muestra una pantalla como la Pantalla 4.



Pantalla 4. Inicio de la aplicación.

Cuando se selecciona un paciente y se presiona el botón Ver Paciente, enseguida mostrará la Pantalla 5.

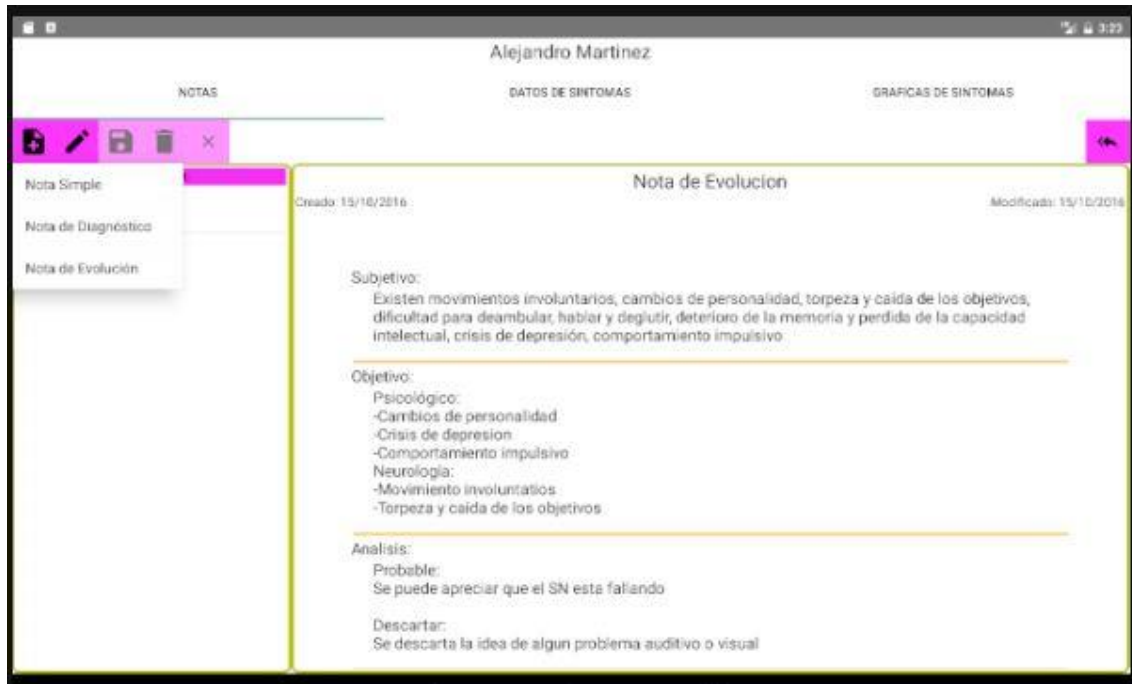


Pantalla 5. Notas médicas.

Crear nueva nota

Para crear una nueva nota solo se presiona el botón de nueva nota .

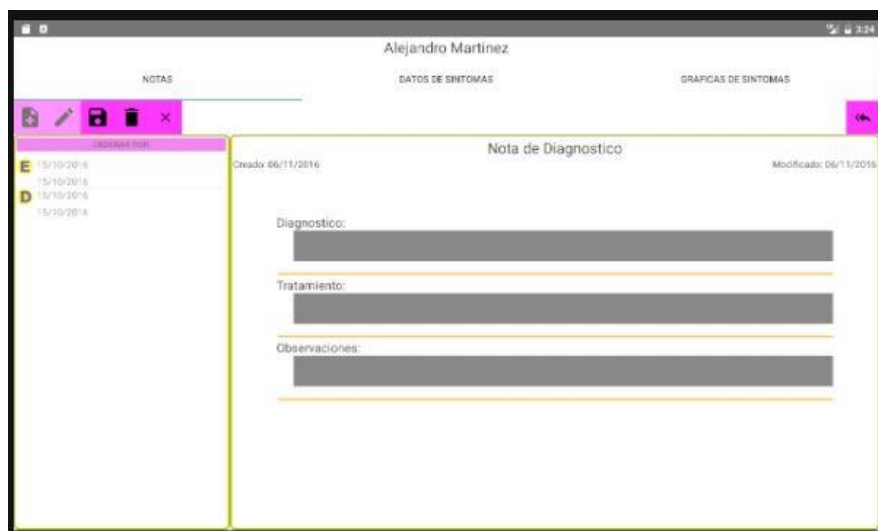
La aplicación muestra una lista de diferentes tipos de notas médicas como se muestra en la Pantalla 6.



Pantalla 6. Nueva nota.

Seleccione un tipo de nota.

En seguida la aplicación creara una nota vacía donde los cuadros editables estarán marcados con diferente color, como se muestra en la Pantalla 7.



Pantalla 7. Nueva nota simple diagnóstico.

También cuando se crea la nota se habilitan los botones de guardar borrar y cancelar, que son las acciones que se puede hacer con la nota creada.

Para guardar la nota solo presione el botón de Guardar



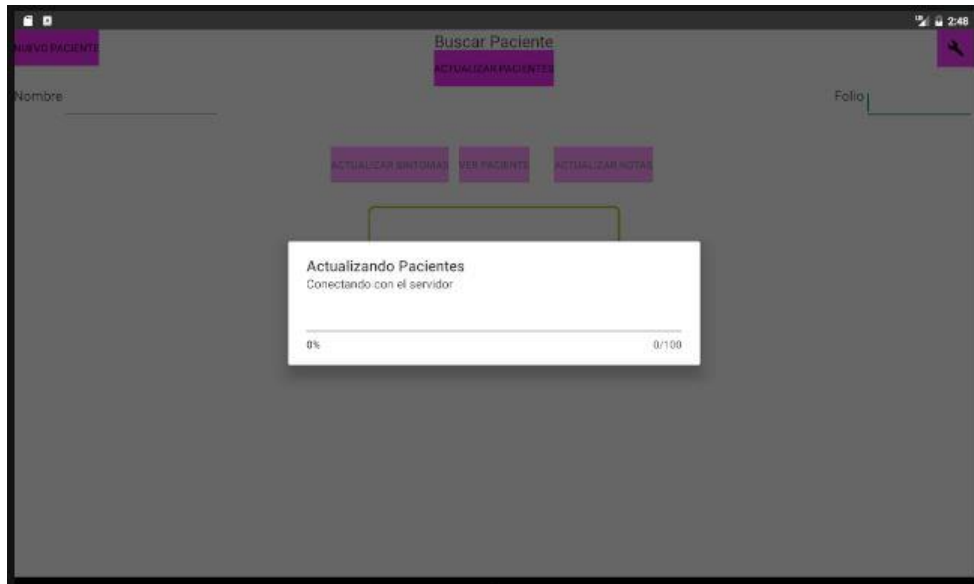
Pantalla 8. Guardar Nota.

Datos como la creado y modificado, se llenan automáticamente.

Actualizar Pacientes

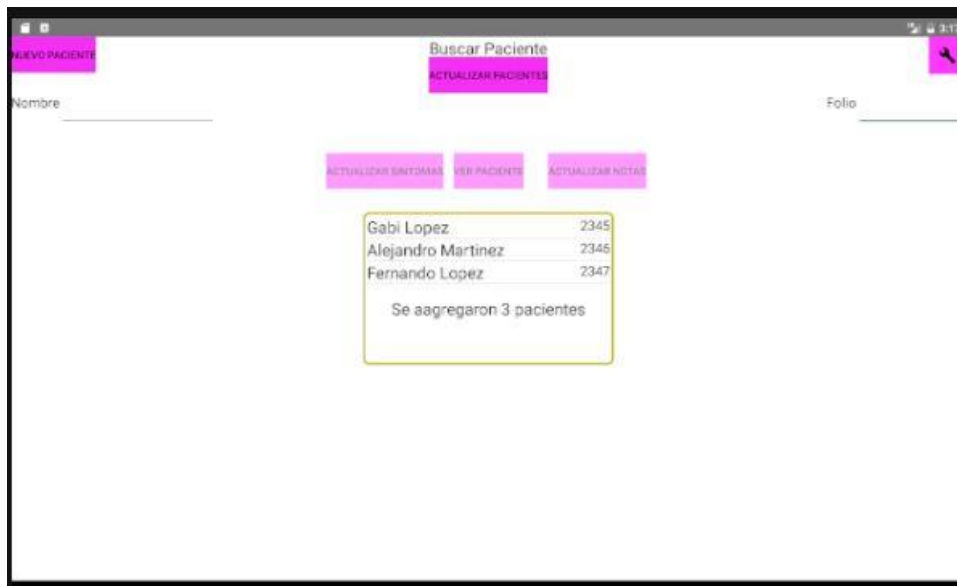
Para actualizar los pacientes con el servicio web solo presione el botón de Actualizar Pacientes, ubicado en la Pantalla 4. Inicio de la aplicación.

Aparecerá una notificación como se muestra en la Pantalla 9.



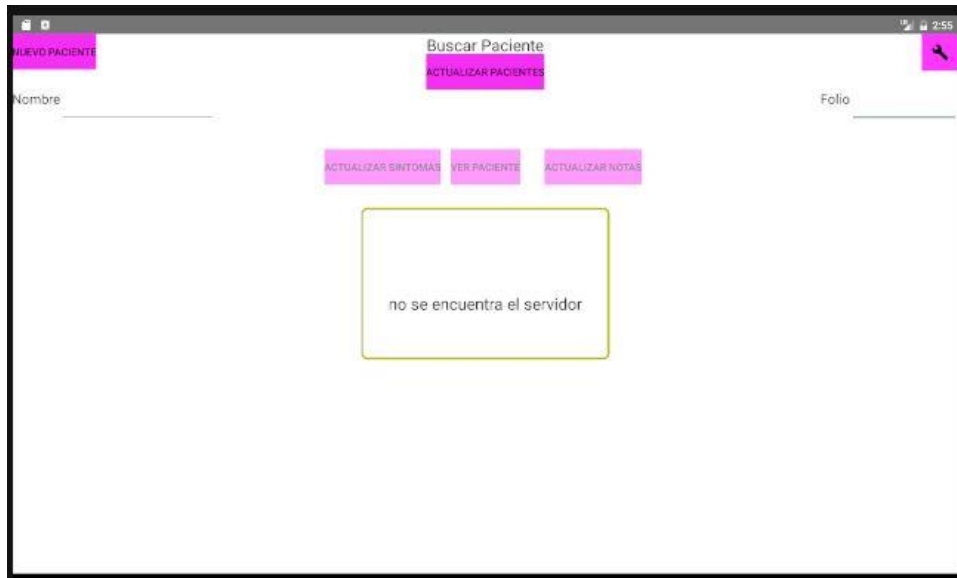
Pantalla 9. Conectando con el servicio web.

Al final aparecerá un mensaje de cuantos pacientes fueron agregados, como se muestra en la Pantalla 10.



Pantalla 10. Éxito al actualizar pacientes.

En caso de que haya un error con la conexión mostrara un mensaje como se muestra en la Pantalla 11.



Pantalla 11. Error de conexión.

6.2. Hardware y Software

6.2.1. Tecnología para el desarrollo de la aplicación

En el desarrollo de la aplicación, se utilizó Android Studio con la versión 2.1.2.

Los requisitos mínimos que requiere Android Studio son:

- 2GB de memoria RAM.
- 2GB de disco duro.
- Java SE Development Kit (JDK) 8.

Para probar la aplicación se utilizó el emulador de Android Studio, que requiere de una tarjeta gráfica Intel de 64-bits.

En el desarrollo del servicio web se utilizó Netbeans versión 8.1 y Apache Tomcat 8.0 que se incluye en la instalación de Netbeans.

Los requisitos mínimos que requiere Netbeans:

- Procesador Intel i5 o equivalente.
- Memoria RAM de 2GB para 32 bits o 4GB para 64 bits.
- 1.5GB de espacio libre en disco duro.
- Java SE Development Kit (JDK) 8

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

La aplicación funciona para Android 3.1 o superior.

Pero para tener todas las fusiones se necesita Android 5.1 o superior.

7. Resultados

En el desarrollo se completaron todos los módulos de la aplicación, a continuación, se mencionan los módulos que se implementaron:

- Gestionar Notas Médicas.
- Ver y graficar síntomas.
- Graficar síntomas.
- Plantillas de Notas Médicas.
- Construir la base de datos para almacenar los pacientes y las notas médicas.
- Diseñar y construir el Servicio de notas médicas.

Aparte se construyó el modulo **Personalizar Colores**, para que la aplicación sea más atractiva visualmente.

8. Análisis y discusión de resultados

Todos los módulos propuestos funcionan correctamente a partir de las versión 3.2 de Android. Sin embargo, el módulo de personalizar colores solo funciona correctamente a partir de la versión 5.1.

9. Conclusiones

Es importante mencionar que este proyecto terminal es mi primer acercamiento a la programación para Android y se requiere tiempo aprender una nueva arquitectura y herramientas.

El objetivo generar que es Diseñar e implementar una aplicación móvil en Android para gestionar notas médicas, si se pudo lograr. La aplicación gestiona correctamente las notas médicas para cada uno de los pacientes registrados.

En el objetivo específico Analizar, diseñar y construir el módulo Gestionar paciente. Solo se logró filtrar los pacientes por nombre o por folio.

Todos los módulos se implementaron de manera exitosa.

En el módulo **Plantillas de notas médicas**, se implementaron nota simple, nota de diagnóstico y nota de evolución.

En el módulo **Ver y filtrar síntomas**, se agregó la características de ordenar ascendente y descendente, y ordenar los síntomas por cada una de sus características (Nombre, intensidad Inicial, Intensidad final, etc.).

En el módulo **Graficar síntomas**, se construyeron dos gráficas diferentes, una de dispersión y otra de líneas. Cada grafica tiene un algoritmo que sin importar cuántos síntomas o cuántos diferentes tipos de síntomas se van a graficar, el algoritmo se tarda $O(n)$ donde n es el número de síntomas a graficar. Gracias a este algoritmo se agregó la característica en la gráfica de dispersión donde el médico selecciona que tipo de síntomas quiere graficar.

10. Perspectivas del Proyecto

En la aplicación como ya se había mencionado se pude arreglar los errores del módulo **Personalizar Colores**. Se puede extender con nuevos tipos de notas médicas. Se puede añadir nuevos módulos tales como: Vacunas del paciente, Citas del Paciente, etc. Se puede añadir nuevas conexiones con otros servicios web para almacenar o compartir información del Paciente.

En el servicio web se puede extender la base de datos para añadir nuevas notas médicas. Se puede crear una página web, para que el médico no esté obligado a instalar la aplicación móvil.

11. Bibliografía

- [1] F. Maldonado, "Prototipo de una aplicación móvil para la gestión de síntomas de un paciente", Proyecto Terminal, División de CBI, Universidad Autonomía Metropolitana Azcapotzalco, Mexico, 2014.
- [2] C. J. García, Sistema para identificar la ubicación e intensidad de los síntomas de un paciente a partir de notas médicas, Proyecto Terminal, División de CBI, Universidad Autonomía Metropolitana Azcapotzalco, México, 2015.
- [3] P. Correa, Sistema para la identificación de fechas de los síntomas de un paciente a partir de notas médicas, Proyecto Terminal, División de CBI, Universidad Autonomía Metropolitana Azcapotzalco, México, 2015.
- [4] Play.google.com, 2016. [En línea]. Available: <https://play.google.com/store/apps/details?id=com.socialnmobile.dictapps.notepad.color.note>.
- [5] Managinglife.com, «ManagingLife | Helping You Through,» 2016. [En línea]. Available: <http://www.managinglife.com/>.
- [6] O. SL, «Ofimedic Software Médico,» Ofimedic.com, 2016. [En línea]. Available: <https://www.ofimedic.com/>.
- [7] Es.wikipedia.org, «Hypertext Transfer Protocol,» [En línea]. Available: https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol.
- [8] Es.wikipedia.org, «Identificador de recursos uniforme,» [En línea]. Available: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme.
- [9] A. Overview, «Android Studio Overview | Android Developers,» Developer.android.com, 2016. [En línea]. Available: <http://developer.android.com/intl/es/reference/packages.html>.
- [10] L.González, «Notas de evolucion en 5 minutos -Sapiens Medicus,» Sapiens Medicus, 2014. [En línea]. Available: <http://sapiensmedicus.org/blog/2014/12/02/notas-de-evolucion-en-cinco-minutos>.

- [11] «Welcome to NetBeans,» Netbeans.org, 2016. [En línea]. Available: <https://netbeans.org/>.
- [12] «RestTemplate (Spring Framework 4.3.3.RELEASE API),» Docs.spring.io, [En línea]. Available: <http://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html>.
- [13] «PhilJay/MPAndroidChart,» GitHub, 2016. [En línea]. Available: <https://github.com/PhilJay/MPAndroidChart>.

Apéndice Javadoc de la Aplicación móvil

Packages

com.example.jhon.notasmedicas
com.example.jhon.notasmedicas.Datos
com.example.jhon.notasmedicas.Modelo
com.example.jhon.notasmedicas.VistasNotas
com.example.jhon.notasmedicas.VistasSintomas

1. Package com.example.jhon.notasmedicas

Class	Description
ExpedienteMedico	Se dio ese nombre a la Clase porque contiene una TabHost, la cual se puede ir añadiendo nuevas pestañas y se puede ir añadiendo información al paciente (igual que los expedientes médicos de un paciente).
MainActivity	Esta clase es la que se muestra al Médico cuando se abre la aplicación.
NuevoPaciente	El objetivo es registrar un nuevo paciente en el servidor web y en la aplicación

1.1. Class ExpedienteMedico

- java.lang.Object
- android.content.Context
 - android.content.ContextWrapper
 - android.view.ContextThemeWrapper
 - android.app.Activity
 - android.support.v4.app.FragmentActivity
 - com.example.jhon.notasmedicas.ExpedienteMedico

Constructor and Description

ExpedienteMedico()

Modifier and Type	Method and Description
protected void	onCreate(android.os.Bundle savedInstanceState)

1.2. Class MainActivity

- java.lang.Object
- android.content.Context
 - android.content.ContextWrapper
 - android.view.ContextThemeWrapper
 - android.app.Activity
 - com.example.jhon.notasmedicas.MainActivity

Constructor and Description

MainActivity()

Modifier and Type	Method and Description
protected void	onCreate(android.os.Bundle savedInstanceState)
protected void	onResume()
protected void	onStart()

1.3. Class NuevoPaciente

- java.lang.Object
- android.content.Context
 - android.content.ContextWrapper
 - android.view.ContextThemeWrapper
 - android.app.Activity
 - com.example.jhon.notasmedicas.NuevoPaciente

Constructor and Description

NuevoPaciente()

Modifier and Type	Method and Description
protected void	onCreate(android.os.Bundle savedInstanceState)

2. Package com.example.jhon.notasmedicas.Datos

Class	Description
BaseDeDatos	
EliminarDatos	
GuardarDatos	
Nombre_Columnas	Contiene los nombre de las tablas y columnas de la base de datos

2.1. Class BaseDeDatos

- java.lang.Object
- android.database.sqlite.SQLiteOpenHelper
 - com.example.jhon.notasmedicas.Datos.BaseDeDatos

Constructor and Description

BaseDeDatos()

Modifier and Type	Method and Description
static BaseDeDatos	getBaseDeDatos()
void	onCreate(android.database.sqlite.SQLiteDatabase db)
void	onUpgrade(android.database.sqlite.SQLiteDatabase db)

	db, int oldVersion, int newVersion)
--	-------------------------------------

2.2. Class EliminarDatos

- java.lang.Object
- com.example.jhon.notasmedicas.Datos.EliminarDatos

Constructor and Description

EliminarDatos()

Modifier and Type	Method and Description
static EliminarDatos	getEliminarDatos()
boolean	nota(NotaDiagnostico nd)
boolean	nota(NotaEvolucion ne)
boolean	nota(NotaSimple ns)

2.3. Class GuardarDatos

- java.lang.Object
- com.example.jhon.notasmedicas.Datos.GuardarDatos

Constructor and Description

GuaradarDatos()

Modifier and Type	Method and Description
void	actualizarNota(Nota nota)
static GuardarDatos	getGurdarDatos()
boolean	nota(int folio, NotaDiagnostico nd)
boolean	nota(int folio, NotaEvolucion ne)
boolean	nota(int folio, NotaSimple ns)
boolean	paciente(java.lang.String nombre, int folio, int id)
boolean	sintoma(int folio, Sintoma sintoma)

2.4. Class Nombre_Columnas

Modifier and Type	Method and Description
java.lang.String	nd_diagnostico
java.lang.String	nd_fecha_creado

java.lang.String	nd_fecha_modificado
java.lang.String	nd_guardarServidor
java.lang.String	nd_id
java.lang.String	nd_idNotaDiagnostico
java.lang.String	nd_observaciones
java.lang.String	nd_tratamiento
java.lang.String	ne_analisis
java.lang.String	ne_fecha_creado
java.lang.String	ne_fecha_modificado
java.lang.String	ne_guardarServidor
java.lang.String	ne_id
java.lang.String	ne_idNotaEvolucion
java.lang.String	ne_objetivo
java.lang.String	ne_plan
java.lang.String	ne_subjetivo
java.lang.String	ns_contenido
java.lang.String	ns_fecha_creado
java.lang.String	ns_fecha_modificado
java.lang.String	ns_guardarServidor
java.lang.String	ns_id
java.lang.String	ns_idNotaSimple
java.lang.String	p_folio
java.lang.String	p_id
java.lang.String	p_idPaciente
java.lang.String	p_nombre
java.lang.String	p_nota_diagnostico
java.lang.String	p_nota_evolucion
java.lang.String	p_nota_simple
java.lang.String	paciente_sintomas
java.lang.String	pnd_id
java.lang.String	pnd_nota_diagnostico
java.lang.String	pnd_paciente
java.lang.String	pne_id
java.lang.String	pne_nota_evolucion
java.lang.String	pne_paciente
java.lang.String	pns_id
java.lang.String	pns_nota_simple
java.lang.String	pns_paciente
java.lang.String	ps_id
java.lang.String	ps_paciente
java.lang.String	ps_sintoma
java.lang.String	s_fechaFinal
java.lang.String	s_fechaInicio
java.lang.String	s_id
java.lang.String	s_idSintoma
java.lang.String	s_intensidadAleatoria

java.lang.String	s_intensidadFinal
java.lang.String	s_intensidadInicial
java.lang.String	s_nombre
java.lang.String	T_nota_diagnostico
java.lang.String	T_nota_evolucion
java.lang.String	T_nota_simple
java.lang.String	T_paciente
java.lang.String	T_sintoma

3. Package com.example.jhon.notasmedicas.Modelo

Class	Description
Nota	Contiene la información mínima y los métodos mínimos que necesita una nota.
NotaDiagnostico	Contiene información de una nota de diagnóstico y dos métodos constructores.
NotaEvolucion	Contiene información de una nota de evolución y dos métodos constructores.
Notas	Contiene todas las notas del paciente, tiene como atributos unos índices para saber cómo ordenar las notas.
NotaSimple	Contiene información de una nota simple y dos métodos constructores.
Paciente	Contiene al paciente que se ésta consultando e información del paciente, su nombre, folio, notas médicas y síntomas.
Pacientes	Contiene todos los pacientes registrados en el sistema.
Sintoma	Contiene toda la información de síntoma.
Sintomas	Contiene todos los síntomas de paciente, los cuales están separados por el nombre del síntoma, en esta clase contiene un método que se puede conocer los nombres de los síntomas y otro que recupera los síntomas de acuerdo al nombre.

3.1. Class Nota

Constructor and Description

`Nota(java.util.Date creado, java.util.Date modificado, int imagen, boolean guardarServidor)`

Modifier and Type	Method and Description
abstract void	<code>editar(android.widget.LinearLayout view, android.content.Context context, android.content.res.Resources resources)</code>
java.lang.String	<code>getCreado()</code>
java.util.Date	<code>getDateCreado()</code>
java.util.Date	<code>getDateModificado()</code>
long	<code>getIdLocal()</code>
int	<code>getIdServidor()</code>
int	<code>getImagen()</code>
java.lang.String	<code>getModificado()</code>
abstract java.lang.String	<code>getTituloNota()</code>
abstract void	<code>guardar(android.widget.LinearLayout view)</code>

void	guardarServidor (boolean b)
boolean	isGuardarServidor ()
abstract void	mostrar (android.widget.LinearLayout view, android.content.Context context)
protected void	ponerEstilosYmostrar (android.widget.TextView editText, android.widget.LinearLayout view, android.content.Context context, java.lang.String cual)
void	setIdLocal (long idLocal)
void	setIdServidor (int id)
void	setModificado (java.util.Date modificado)

3.2. Class NotaDiagnostico

- com.example.jhon.notasmedicas.Modelo.Nota
 - com.example.jhon.notasmedicas.Modelo.NotaDiagnostico

Constructor and Description

NotaDiagnostico(java.util.Date creado, java.util.Date modificado)
NotaDiagnostico(java.lang.String diagnostico, java.lang.String tratamiento, java.lang.String observaciones, java.util.Date creado, java.util.Date modificado, boolean guardarServidor)

Modifier and Type	Method and Description
void	editar (android.widget.LinearLayout view, android.content.Context context, android.content.res.Resources resources)
java.lang.String	getDiagnostico ()
java.lang.String	getObservaciones ()
java.lang.String	getTituloNota ()
java.lang.String	getTratamiento ()
void	guardar (android.widget.LinearLayout view)
void	mostrar (android.widget.LinearLayout view, android.content.Context context)

3.3. Class NotaEvolucion

- com.example.jhon.notasmedicas.Modelo.Nota
 - com.example.jhon.notasmedicas.Modelo.NotaEvolucion

Constructor and Description

NotaEvolucion(java.util.Date creado, java.util.Date modificado)
NotaEvolucion(java.lang.String subjetivo, java.lang.String objetivo, java.lang.String analisis, java.lang.String plan, java.util.Date creado, java.util.Date modificado,

boolean guardarServidor)

Modifier and Type	Method and Description
void	editar(android.widget.LinearLayout view , android.content.Context context , android.content.res.Resources resources)
java.lang.String	getAnalisis()
java.lang.String	getObjetivo()
java.lang.String	getPlan()
java.lang.String	getSubjetivo()
java.lang.String	getTituloNota()
void	guardar(android.widget.LinearLayout view)
void	mostrar(android.widget.LinearLayout view , android.content.Context context)

3.4. Class Notas

Constructor and Description

Notas(java.util.ArrayList<Nota> notas)

Modifier and Type	Method and Description
boolean	addNota(Nota nota)
Nota	getNota(int i)
Nota	getNotaCreado(int i)
Nota	getNotaModificado(int i)
Nota	getNotaTipo(int i)
java.util.List<Nota>	getTitulos()
java.util.List<Nota>	getTitulosCreado()
java.util.List<Nota>	getTitulosModificado()
java.util.List<Nota>	getTitulosTipo()
void	ordenarCreado()
void	ordenarModificado()
void	ordenarTipo()
boolean	removeNota(Nota nota)

3.5. Class NotaSimple

- com.example.jhon.notasmedicas.Modelo.Nota
 - com.example.jhon.notasmedicas.Modelo.NotaSimple

Constructor and Description

NotaSimple(java.util.Date creado, java.util.Date modificado)

NotaSimple(java.lang.String contenido, java.util.Date creado, java.util.Date modificado, boolean guardarSevidor)

Modifier and Type	Method and Description
void	<code>editar(android.widget.LinearLayout view, android.content.Context context, android.content.res.Resources resources)</code>
java.lang.String	<code>getContenido()</code>
java.lang.String	<code>getTituloNota()</code>
void	<code>guardar(android.widget.LinearLayout view)</code>
void	<code>mostrar(android.widget.LinearLayout view, android.content.Context context)</code>

3.6. Class Paciente

Constructor and Description

`Paciente(java.lang.String nombre, java.lang.String folio)`

Modifier and Type	Method and Description
java.lang.String	<code>getFolio()</code>
int	<code>getId()</code>
java.lang.String	<code>getNombre()</code>
Notas	<code>getNotas()</code>
static Paciente	<code>getPaciente()</code>
Sintomas	<code>getSintomas()</code>
static Paciente	<code>nuevoPaciente(java.lang.String nombre, java.lang.String folio)</code>

3.7. Class Pacientes

Constructor and Description

`Pacientes()`

Modifier and Type	Method and Description
boolean	<code>addPaciente(java.lang.String nombre, java.lang.String folio)</code>
java.util.ArrayList<Paciente>	<code>getPacientes()</code>

3.8. Class Sintoma

Constructor and Description

`Sintoma(int idServidor, int intensidadInicial, int intensidadFinal, boolean intensidadAleatoria, java.util.Date fechaInicio, java.util.Date fechaFinal, java.lang.String nombre)`

Modifier and Type	Method and Description
long	<code>getDuracion()</code>
java.lang.String	<code>getDuracionS()</code>
java.util.Date	<code>getFecha()</code>
java.util.Date	<code>getFechaFinal()</code>
java.lang.String	<code>getFechaFinalS()</code>

java.lang.String	getFechaInicialS()
int	getIdServidor()
java.lang.String	getIntencidadFinalS()
java.lang.String	getIntencidadInicialS()
float	getIntensidad()
int	getIntensidadFinal()
int	getIntensidadIncial()
java.lang.String	getNombre()
boolean	isIntensidadAleatoria()
boolean	isIntensidadContinua()
boolean	isIntensidadEnAumento()

3.9. Class Sintomas

Constructor and Description

Sintomas()

Modifier and Type	Method and Description
boolean	addSintoma(java.lang.String nombre, Sintoma sintoma)
java.util.ArrayList<java.lang.String>	getNombresSintomas()
java.util.ArrayList<Sintoma>	getSintomas(java.lang.String nombre)

4. Package com.example.jhon.notasmedicas.VistasNotas

Class	Description
BarraDeMenuNotas	Tiene como objetivo gestionar las notas médicas (Añadir, editar, eliminar y cancelar).
MenuNotas	Tiene como objetivo mostrar una lista de todas las notas y ordenar la lista.
NotaContenido	Tiene como objetivo mostrar información de una de las notas y poder editarlas.
NotasFragment	Tiene como objetivo poder mostrar una lista de notas y el contenido de una nota al mismo tiempo.

4.1. Class BarraDeMenuNotas

- java.lang.Object
 - android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasNotas.BarraDeMenuNotas

Constructor and Description

BarraDeMenuNotas()

Modifier and Type	Method and Description
void	deshabilitarBotones()
void	onCreate(android.os.Bundle savedInstanceState)
android.view.View	onCreateView(android.view.LayoutInflater

	<code>inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)</code>
<code>void</code>	<code>onPause()</code>
<code>void</code>	<code>deshabilitarBotones()</code>
<code>void</code>	<code>onCreate(android.os.Bundle savedInstanceState)</code>

4.2. Class MenuNotas

- `java.lang.Object`
- `android.support.v4.app.Fragment`
 - `com.example.jhon.notasmedicas.VistasNotas.MenuNotas`

Constructor and Description

`MenuNotas()`

Modifier and Type	Method and Description
<code>boolean</code>	<code>editarNota()</code>
<code>void</code>	<code>eliminarNota()</code>
<code>void</code>	<code>filtrarNotas(int cual)</code>
<code>boolean</code>	<code>guardarNota()</code>
<code>void</code>	<code>habilitarOrdenar()</code>
<code>void</code>	<code>inhabilitarOrdenar()</code>
<code>void</code>	<code>nuevaNota(int cual)</code>
<code>void</code>	<code>onCreate(android.os.Bundle savedInstanceState)</code>
<code>android.view.View</code>	<code>onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)</code>
<code>void</code>	<code>setArguments(NotaContenido notaContenido, BarraDeMenuNotas barraDeMenuNotas)</code>
<code>void</code>	<code>verNota(int posicion)</code>

4.3. Class Contenido

- `java.lang.Object`
- `android.support.v4.app.Fragment`
 - `com.example.jhon.notasmedicas.VistasNotas.NotaContenido`

Constructor and Description

`NotaContenido()`

Modifier and Type	Method and Description
void	cancelarNota()
void	editarNota()
void	eliminarNota()
void	guardar()
void	nuevaNota(Nota n)
android.view.View	onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)
void	verNota(Nota nota)

4.4. Class NotasFragment

- java.lang.Object
- android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasNotas.NotasFragment

Constructor and Description

NotasFragment()

Modifier and Type	Method and Description
void	onCreate(android.os.Bundle savedInstanceState)
android.view.View	onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)
void	setArguments(NotaContenido notaContenido, MenuNotas menuNotas)

5. Package com.example.jhon.notasmedicas.VistasSintomas

Class	Description
DatosSintomas	Tiene como objetivo poder mostrar una lista y al mismo tiempo unos botones al mismo tiempo.
GraficarComparacionSintomas	Tiene como objetivo crear una gráfica de dispersión.
GraficaSintomas	A simple Fragment subclass.
GraficaSintomaSintomas	Tiene como objetivo crear una gráfica de líneas.
GraficasSintomasFragment	Tiene como objetivo poder mostrar una lista y una gráfica al mismo tiempo.
TablaSintomas	Tiene como objetivo mostrar una lista de los síntomas y ordenar la lista.

5.1. Class DatosSintomas

- java.lang.Object

- android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasSintomas.DatosSintomas

Constructor and Description

DatosSintoma()

Modifier and Type	Method and Description
void	onCreate(android.os.Bundle savedInstanceState)
android.view.View	onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)

5.2. Class GraficarComparacionSintomas

- java.lang.Object
 - android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasSintomas.GraficaComparacionSintomas

Constructor and Description

GraficarComparacionSintomas ()

Modifier and Type	Method and Description
protected com.github.mikephil.charting.data.ScatterData	crearDatosComparacion(java.util.ArrayList quienParticipa)
void	onCreate(android.os.Bundle savedInstanceState)
android.view.View	onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)

5.3. Class GraficaSintomas

- java.lang.Object
 - android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasSintomas.GraficaSintomas

Constructor and Description

GraficarSintomas ()

Modifier and Type	Method and Description
void	cambiarGraficas()
protected com.github.mikephil.charting.data.ScatterData	crearDatosComparacion (java.util.ArrayList quienParticipa)
com.github.mikephil.charting.data.LineData	crearDatosGrafica (java.util.ArrayList quienParticipa)
com.github.mikephil.charting.data.LineData	crearDatosSintoma (java.util.ArrayList<Sintoma> sintoma)
protected com.github.mikephil.charting.data.ScatterData	generateScatterData()
void	inicializar (java.util.ArrayList<java.util.ArrayList<Sintoma>> sintomas, java.util.ArrayList<java.lang.String> sintomasNombres)
void	mostrar (java.util.ArrayList quienParticipa)
void	mostrar (com.github.mikephil.charting.data.ScatterData scatterData, com.github.mikephil.charting.data.LineData lineData, java.lang.String titulo)
android.view.View	onCreateView (android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)
void	ordenarSintomas (int izq, int der, java.util.ArrayList ordenar)
void	setArguments (GraficaSintomas elOtro, GraficasSintomasFragment siFragment)

5.4. Class GraficaSintomaSintomas

- java.lang.Object
- android.support.v4.app.Fragment
 - com.example.jhon.notasmedicas.VistasSintomas.GraficaSintomaSintomas

Constructor and Description

GraficaSintomaSintomas ()

Modifier and Type	Method and Description
void	mostrarGrafica (java.lang.String sintoma)
void	onCreate (android.os.Bundle savedInstanceState)
android.view.View	onCreateView (android.view.LayoutInflater inflater, android.view.ViewGroup container,

	<code>android.os.Bundle savedInstanceState)</code>
--	--

5.5. Class GraficaSintomasFragment

- `java.lang.Object`
- `android.support.v4.app.Fragment`
 - `com.example.jhon.notasmedicas.VistasSintomas.GraficasSintomasFragment`

Constructor and Description

GraficasSintomasFragment ()

Modifier and Type	Method and Description
<code>void</code>	<code>onCreate(android.os.Bundle savedInstanceState)</code>
<code>android.view.View</code>	<code>onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)</code>

5.6. Class TablaSintomas

- `java.lang.Object`
- `android.support.v4.app.Fragment`
 - `com.example.jhon.notasmedicas.VistasSintomas.TablaSintomas`

Constructor and Description

TablaSintomas ()

Modifier and Type	Method and Description
<code>void</code>	<code>cambioSeleccion(java.util.ArrayList cuales)</code>
<code>void</code>	<code>onCreate(android.os.Bundle savedInstanceState)</code>
<code>android.view.View</code>	<code>onCreateView(android.view.LayoutInflater inflater, android.view.ViewGroup container, android.os.Bundle savedInstanceState)</code>

Apéndice Javadoc del servicio web

Packages

DAO, Servicios, Transporte, VO.

1. Package DAO

Class	Description
DAONotas	Es una clase DAO que tiene como objetivo almacenar y recuperar las notas de la base de datos.
DAOPacientes	Es una clase DAO que tiene como objetivo almacenar y recuperar los pacientes de la base de datos.
DAOSintomas	Es una clase DAO que tiene como objetivo almacenar y recuperar los síntomas de la base de datos.
NewHibernateUtil	Hibernate Utility class with a convenient method to get Session Factory object

1.1. Class DAONotas

Modifier and Type	Method and Description
boolean	actualizarNota(NotaDiagnostico nd)
boolean	actualizarNota(NotaEvolucion ne)
boolean	actualizarNota(NotaSimple ns)
NotaDiagnostico	buscarNotaDiagnostico(int id)
NotaEvolucion	buscarNotaEvolucion(int id)
NotaSimple	buscarNotaSimple(int id)
static DAONotas	getDAONotas()
java.util.List< NotaDiagnostico >	getNotasDiagnostico(int folio)
java.util.List< NotaEvolucion >	getNotasEvolucion(int folio)
java.util.List< NotaSimple >	getNotasSimples(int folio)
boolean	ingresarNota(NotaDiagnostico nd, Paciente paciente)
boolean	ingresarNota(NotaEvolucion ne, Paciente paciente)
boolean	ingresarNota(NotaSimple ns, Paciente paciente)

1.2. Class DAOPacientes

Modifier and Type	Method and Description
Paciente	buscaPaciente(int folio)
boolean	existeFolio(int folio)
static DAOPaciente	getDAOPaciente()
boolean	insertarPaciente(Paciente p)
java.util.List< NotaSimple >	todosPacientes()

1.3. Class DAOSintomas

Modifier and Type	Method and Description
boolean	acualizarSintoma(Sintoma sintoma)
java.util.List< Sintoma >	buscarSintomas(int folio)
static DAOSintomas	getDAOSintomas()
Boolean	insertarSintoma(Paciente paciente, Sintoma sintoma)

1.4. Class NewHibernateUtil

Modifier and Type	Method and Description
Static	getSessionFactory()
org.hibernate.SessionFactory	
Static void	shutdown()

2. Package Servicios

Class	Description
ActualizarNotasResource	REST Web Service. Tiene como objetivo ofrecer el servicio web para actualizar notas.
ActualizarPacientesResource	REST Web Service. Tiene como objetivo ofrecer el servicio web para actualizar los pacientes
BuscarNotasResource	REST Web Service. Tiene como objetivo ofrecer el servicio web para buscar notas.
BuscarSintomasResource	REST Web Service. Tiene como objetivo llevar información de un síntoma.
RegistrarPacienteResource	REST Web Service. Tiene como objetivo ofrecer el servicio web para registrar un paciente.

2.1. Class ActualizarNotasResource

Modifier and Type	Method and Description
java.lang.String	post(java.lang.String p)

2.2. Class ActualizarPacientesResource

Modifier and Type	Method and Description
java.lang.String	post(java.lang.String p)

2.3. Class BuscarNotasResource

Modifier and Type	Method and Description
java.lang.String	post(java.lang.String folio)

2.4. Class BuscarSintomasResource

Modifier and Type	Method and Description
java.lang.String	post(java.lang.String folio)

2.5. Class RegistrarPacienteResource

Modifier and Type	Method and Description
java.lang.String	post(java.lang.String p)

3. Package Transporte

Class	Description
TNotaDiagnostico	Tiene como objetivo llevar información de una nota de diagnóstico.
TNotaEvolucion	Tiene como objetivo llevar información de una nota de evolución.
TNotaSimple	Tiene como objetivo llevar información de una nota simple.
TPaciente	Tiene como objetivo llevar información del paciente, y si es necesario también llevara las notas y/o los síntomas.
TPacientes	Tiene como objetivo transportar un grupo de pacientes.
TSintoma	Tiene como objetivo llevar información de un síntoma.

3.1. Class TNotaDiagnostico

Constructor and Description

TNotaDiagnostico()

TNotaDiagnostico(org.json.JSONObject nota)

TNotaDiagnostico(NotaDiagnostico nd)

Modifier and Type	Method and Description
java.lang.String	getDiagnostico()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaDiagnostico()
java.lang.String	getObservaciones()
java.lang.String	getTratamiento()
void	setDiagnostico(java.lang.String diagnostico)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)
void	setIdNotaDiagnostico(java.lang.Integer idNotaDiagnostico)
void	setObservaciones(java.lang.String observaciones)
void	setTratamiento(java.lang.String tratamiento)

3.2. Class TNotaEvolucion

Constructor and Description

TNotaEvolucion()

TNotaEvolucion(org.json.JSONObject nota)

TNotaEvolucion(NotaEvolucion ne)

Modifier and Type	Method and Description
java.lang.String	getAnalisis()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaEvolucion()
java.lang.String	getObjetivo()
java.lang.String	getPlan()
java.lang.String	getSubjetivo()
void	setAnalisis(java.lang.String analisis)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)
void	setIdNotaEvolucion(java.lang.Integer idNotaEvolucion)
void	setObjetivo(java.lang.String objetivo)
void	setPlan(java.lang.String plan)
void	setSubjetivo(java.lang.String subjetivo)

3.3. Class TNotaSimple

Constructor and Description

TNotaSimple()

TNotaSimple(org.json.JSONObject nota)

TNotaSimple(**NotaSimple** ns)

Modifier and Type	Method and Description
java.lang.String	getContenido()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaSimple()
void	setContenido(java.lang.String contenido)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)
void	setIdNotaSimple(java.lang.Integer idNotaSimple)

3.4. Class TPaciente

Constructor and Description

TPaciente()

TPaciente(org.json.JSONObject paciente)

TPaciente(**Paciente** paciente)

Modifier and Type	Method and Description
int	getFolio()
java.lang.Integer	getIdPaciente()
java.lang.String	getMensaje()
java.lang.String	getNombre()
java.util.List<TNotaDiagnostico>	getNotasDiagnostico()
java.util.List<TNotaEvolucion>	getNotasEvolucion()
java.util.List<TNotaSimple>	getNotasSimples()
java.util.List<TSintoma>	getSintomas()
void	setFolio(int folio)
void	setIdPaciente(java.lang.Integer idPaciente)
void	setMensaje(java.lang.String mensaje)
void	setNombre(java.lang.String nombre)
void	setNotasDiagnostico(java.util.List< TNotaDiagnostico > notasDiagnostico)
void	setNotasEvolucion(java.util.List< TNotaEvolucion > notasEvolucion)
void	setNotasSimples(java.util.List< TNotaSimple > notasSimples)
void	setSintomas(java.util.List< TSintoma > sintomas)

3.5. Class TPacientes

Constructor and Description

TPacientes()

Modifier and Type	Method and Description
boolean	addPaciente(TPaciente paciente)
java.util.List<TPaciente>	getPacientes()

3.6. Class TSintoma

Constructor and Description

TSintoma()

TSintoma(**sintoma** sintoma)

Modifier and Type	Method and Description
java.lang.String	getFechaFinal()
java.lang.String	getFechaInicio()
java.lang.Integer	getIdSintoma()
int	getIntensidadFinal()
int	getIntensidadInicial()
java.lang.String	getNombre()
boolean	isIntensidadAleatoria()
void	setFechaFinal(java.lang.String fechaFinal)
void	setFechaInicio(java.lang.String fechaInicio)
void	setIdSintoma(java.lang.Integer idSintoma)
void	setIntensidadAleatoria(boolean intensidadAleatoria)
void	setIntensidadFinal(int intensidadFinal)
void	setIntensidadInicial(int intensidadInicial)
void	setNombre(java.lang.String nombre)

4. Package Transporte

Class	Description
NotaDiagnostico	NotaDiagnostico generated by hbm2java Es una clase VO que utiliza DAONotas y tiene como objetivo tener la información de una nota de diagnóstico.
NotaEvolucion	NotaEvolucion generated by hbm2java Es una clase VO que utiliza DAONotas y tiene como objetivo tener la información de una nota de evolución.
NotaSimple	NotaSimple generated by hbm2java Es una clase VO que utiliza DAONotas y tiene como objetivo tener la información de una nota simple.
Paciente	Paciente generated by hbm2java Es una clase VO que utiliza DAOPacientes y tiene como objetivo tener la información de un paciente.
Sintoma	Sintoma generated by hbm2java Es una clase VO que utiliza DAOSintomas y tiene como objetivo tener la información de un síntoma.

4.1. Class NotaDiagnostico

Constructor and Description

NotaDiagnostico()

TPaciente(**Paciente** paciente)

Modifier and Type	Method and Description
java.lang.String	getDiagnostico()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaDiagnostico()
java.lang.String	getObservaciones()
java.util.Set< PacienteNotaDiagnostico >	getPacienteNotaDiagnosticos()
java.lang.String	getTratamiento()
void	setDiagnostico(java.lang.String diagnostico)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)
void	setIdNotaDiagnostico(java.lang.Integer idNotaDiagnostico)
void	setObservaciones(java.lang.String observaciones)
void	setPacienteNotaDiagnosticos(java.util.Set< PacienteNotaDiagnostico > pacienteNotaDiagnosticos)
void	setTratamiento(java.lang.String tratamiento)
java.lang.String	getDiagnostico()
java.lang.String	getFechaCreado()

4.2. Class NotaEvolucion

Constructor and Description

NotaEvolucion()

NotaEvolucion(**TNotaEvolucion** ne)

Modifier and Type	Method and Description
java.lang.String	getAnalisis()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaEvolucion()
java.lang.String	getObjetivo()
java.util.Set< PacienteNotaEvolucion >	getPacienteNotaEvoluciones()
java.lang.String	getPlan()
java.lang.String	getSubjetivo()
void	setAnalisis(java.lang.String analisis)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)

void	setIdNotaEvolucion(java.lang.Integer idNotaEvolucion)
void	setObjetivo(java.lang.String objetivo)
void	setPacienteNotaEvoluciones(java.util.Set<PacienteNotaEvolucion> pacienteNotaEvoluciones)
void	setPlan(java.lang.String plan)
void	setSubjetivo(java.lang.String subjetivo)

4.3. Class NotaSimple

Constructor and Description

NotaSimple()

NotaSimple(TNotaSimple ns)

Modifier and Type	Method and Description
java.lang.String	getContenido()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaSimple()
java.util.Set<PacienteNotaSimple>	getPacienteNotaSimples()
void	setContenido(java.lang.String contenido)
void	setFechaCreado(java.lang.String fechaCreado)
void	setFechaModificado(java.lang.String fechaModificado)
void	setIdNotaSimple(java.lang.Integer idNotaSimple)
void	setPacienteNotaSimples(java.util.Set<PacienteNotaSimple> pacienteNotaSimples)
java.lang.String	getContenido()
java.lang.String	getFechaCreado()
java.lang.String	getFechaModificado()
java.lang.Integer	getIdNotaSimple()
java.util.Set<PacienteNotaSimple>	getPacienteNotaSimples()

4.4. Class Paciente

Constructor and Description

Paciente()

Paciente(TPaciente paciente)

Modifier and Type	Method and Description
int	getFolio()
java.lang.Integer	getIdPaciente()
java.lang.String	getNombre()

java.util.Set<PacienteNotaDiagnostico>	getPacienteNotaDiagnosticos()
java.util.Set<PacienteNotaEvolucion>	getPacienteNotaEvoluciones()
java.util.Set<PacienteNotaSimple>	getPacienteNotaSimples()
java.util.Set<PacienteSintomas>	getPacienteSintomasas()
void	setFolio(int folio) int
void	setIdPaciente(java.lang.Integer idPaciente)
void	setNombre(java.lang.String nombre)
void	setPacienteNotaDiagnosticos(java.util.Set<PacienteNotaDiagnostico> pacienteNotaDiagnosticos)
void	setPacienteNotaEvoluciones(java.util.Set<PacienteNotaEvolucion> pacienteNotaEvoluciones)
void	setPacienteNotaSimples (java.util.Set<PacienteNotaSimple> pacienteNotaSimples)
void	setPacienteSintomasas(java.util.Set<PacienteSintomas> pacienteSintomasas)
int	getFolio()
java.lang.Integer	getIdPaciente()

4.5. Class Sintoma

Constructor and Description

Sintoma()

Modifier and Type	Method and Description
java.lang.String	getFechaFinal()
java.lang.String	getFechaInicio()
java.lang.Integer	getIdSintoma()
int	getIntensidadFinal()
int	getIntensidadInicial()
java.lang.String	getNombre()
java.util.Set<PacienteSintomas>	getPacienteSintomasas()
boolean	isActualizar()
boolean	isIntensidadAleatoria()
void	setActualizar(boolean actualizar)
void	setFechaFinal(java.lang.String fechaFinal)
void	setFechaInicio(java.lang.String fechaInicio)
void	setIdSintoma(java.lang.Integer idSintoma)
void	setIntensidadAleatoria(boolean intensidadAleatoria)
void	setIntensidadFinal(int intensidadFinal)
void	setIntensidadInicial(int intensidadInicial)

