UNIVERSIDAD DE LA RIOJA

Facultad

Facultad de Ciencia y Tecnología

Titulación

Grado en Ingeniería Informática

Título

Prototipo de aplicación móvil para consultar la sensibilidad antimicrobiana a partir de imágenes de antibiogramas

Autor/es

Juan Miguel Vieira Ramírez

Tutor/es

Eloy Javier Mata, Jónathan Heras

Departamento

Matemáticas y Computación

Curso académico

2015-2016

UNIVERSIDAD DE LA RIOJA

Facultad de Ciencia y Tecnología

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Prototipo de aplicación móvil para consultar la sensibilidad antimicrobiana a partir de imágenes de antibiogramas

Alumno:

Juan Miguel Vieira Ramírez

Tutores:

Eloy Javier Mata

Jónathan Heras

Logroño, Junio, 2016

Resumen

El principal objetivo de este proyecto es proporcionar una herramienta informática que facilite el proceso de análisis de la sensibilidad antimicrobiana ante diferentes concentraciones de antibióticos. Para conseguirlo, el primer paso ha consistido en extraer los datos del documento Excel que anualmente publica el EUCAST (European Committee on Antimicrobial Susceptibility Testing), donde se recogen los resultados de los test de susceptibilidad antimicrobiana, y almacenarlos en forma de documento XML bien estructurado.

Posteriormente, usando el documento XML como origen de datos, se ha desarrollado un servicio web que realiza los cálculos y proporciona el resultado de la sensibilidad de la bacteria analizada.

Finalmente, se ha creado un prototipo de aplicación móvil que muestra cómo consumir el servicio web y ver los resultados de una manera clara e intuitiva.

Summary

The main objective of this project is to provide a computing tool that facilitates the analysis of antimicrobial sensitivity to different concentrations of antibiotics. To achieve this goal, the first step was to extract data from an Excel document published annually by the EUCAST (European Committee on Antimicrobial Susceptibility Testing), where the results of the antimicrobial susceptibility test are collected, and to store it as an XML document well structured.

Afterwards, using the XML document as a data source, it was developed a web service that performs calculations and provides the results of the analyzed bacteria sensitivity.

Finally, it was created a prototype mobile application that shows how to consume the web service and allows to view the results in a clear and intuitive way.

Contenido

Introducción

Este trabajo surge de un proyecto de colaboración entre el "Grupo de investigación en informática" y el "Grupo de investigación de ecología molecular de la resistencia a antimicrobiano y seguridad alimentaria", ambos de la Universidad de La Rioja.

El objetivo de este proyecto es facilitar el proceso de análisis de la sensibilidad bacteriana ante diferentes concentraciones de antibióticos.

Este estudio de la sensibilidad bacteriana se realiza principalmente por técnicas de dilución o difusión. En el caso de las de difusión se obtienen resultados cualitativos (sensible, intermedio, resistente).

Una de las pruebas más utilizadas es el antibiograma realizado por el método de difusión en agar.

La interpretación de los resultados del antibiograma (sensible, intermedio o resistente) se realiza en función de los valores establecidos por diferentes comités, como el Clinical and Laboratory Standards Institute en Estados Unidos, el European Committee on Antimicrobial Susceptibility Testing en Europa y la Mesa Española de Normalización de la Sensibilidad y Resistencia a los Antimicrobianos.

Este análisis del antibiograma consta de tres partes:

1- La bacteria a investigar se inocula en una placa de agar y sobre su superficie se disponen discos de papel de filtro a los que previamente se les ha incorporado el agente antimicrobiano.
2- Se incuba la placa durante unas horas y se estudia el crecimiento bacteriano en ella.
3- Se mide los diámetros de la zona de inhibición (halos) que se forma alrededor de cada disco y se compara con las referencias oportunas publicadas por los organismos oficiales.

Este trabajo fin de grado se concentra en la última parte. Su objetivo principal es facilitar la consulta de las referencias publicadas por el European Committee on Antimicrobial Susceptibility Testing (EUCAST).

En la *Figura 1* se muestra un antibiograma donde cada disco ha creado la zona de inhibición que se medirá para determinar la sensibilidad de la bacteria cultivada al antibiótico del disco.

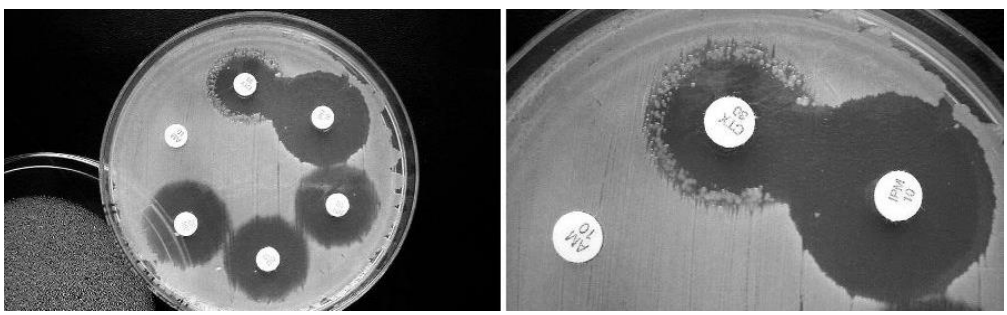


Figura 1. Antibiograma. Fuente: http://www.microinmuno.qb.fcen.uba.ar/SeminarioAntibioticos.htm

Metodología

He decidido realizar el proyecto utilizando metodología ágil por los siguientes motivos:

- Es un proyecto con algunos rasgos bien definidos pero otras opciones que pueden variar bastante durante la realización
- Quiero mantener una comunicación constante con el cliente al integrarlo en el equipo de trabajo para obtener un producto mejor adaptado a sus necesidades
- Considero importante que, si por alguna razón no se llega a completar todo lo esperado inicialmente, haya obtenido un producto funcional aunque sea de forma parcial
- Quiero aprender y profundizar en el uso de metodologías ágiles

Sprint 0. Planificación

Como es costumbre al emplear una metodología ágil, desarrollo un sprint inicial, el sprint 0, que recoge la definición y planificación del proyecto.

Temas

Para definir la funcionalidad del proyecto utilizo temas e historias de usuario.

Identifico tres temas bien definidos:

ID	T1
Nombre	Conversor Excel – XML
Descripción	Como usuario quiero que la aplicación extraiga los datos del fichero Excel y forme un fichero XML con ellos para usarlo como origen de datos.
Coste	4
Valor	7

ID	T2
Nombre	Servicio web
Descripción	Como usuario quiero que la aplicación utilice un fichero XML como origen de datos, admita peticiones REST y proporcione los datos de forma ordenada y coherente.
Coste	4
Valor	9

ID	T3
Nombre	Aplicación móvil
Descripción	Como usuario quiero que la aplicación, a través de una interfaz usable, permita utilizar el servicio web de forma intuitiva y clara.
Coste	8
Valor	10

Para las historias de usuario de los próximos sprints utilizaré la técnica MoSCoW para elegir la importancia. Para estos tres temas no lo veo necesario.

Historias de usuario

Para el tema 1, Conversor Excel – XML:

Para este tema trabajo con los siguientes recursos:

- Documento Excel obtenido de la web de EUCAST (http://www.eucast.org/fileadmin/src/media/PDFs/EUCAST_files/Breakpoint_tables/v_6.0_Breakpoint_table.xls)
- Entorno oXygen v16.1 para la generación del XML Schema y el XML final

Identifico inicialmente las siguientes **historias de usuario**:

ID	H1.1
Nombre	XML Schema
Descripción	Como desarrollador quiero que el documento XML final cumpla un esquema para su uso correcto y validación
Coste	3
Valor	8
Dependencias	-
Condiciones de Satisfacción	-Creación de un XML Schema correcto -Se puede usar el esquema para crear el XML de forma adecuada -El esquema corresponde de forma lógica a la información y estructura del documento Excel

ID	H1.2
Nombre	XML
Descripción	Como usuario quiero que el documento XML contenga, de forma lógica y clara, la información del documento Excel
Coste	8
Valor	10
Dependencias	H1.1
Condiciones de Satisfacción	-Creación de un XML con los datos -Se puede utilizar el fichero XML como origen de datos válido

Para el tema 2, Servicio Web

Cuento con los recursos siguientes:

- XML generado por la primera parte del proyecto como origen de datos
- Entorno NetBeans v8.1 con Java 7 (mejor compatibilidad) instalada para la creación del servicio
- Entorno oXygen v16.1 para las pruebas con el lenguaje de consultas XQUERY
- Servidor Centos 8 suministrado por la Universidad de La Rioja para desplegar el servicio

El tema está compuesto inicialmente por las **historias de usuario**:

ID	H2.1
Nombre	Consultas XQUERY
Descripción	Como desarrollador quiero conseguir las consultas necesarias en lenguaje XQUERY para la obtención de los datos del fichero XML
Coste	1
Valor	8
Dependencias	H1.2
Condiciones de Satisfacción	Obtener las consultas necesarias

ID	H2.2
Nombre	Servicio en Java
Descripción	Como usuario quiero tener un servicio web para aceptar peticiones, consultar el origen de datos XML y devolver los datos correspondientes a la petición
Coste	9
Valor	10
Dependencias	-H1.2 -H2.1
Condiciones de Satisfacción	Puedo hacer peticiones y obtener los datos correctos

ID	H2.3
Nombre	Despliegue del servicio
Descripción	Como usuario quiero poder acceder al servicio a través de un dominio determinado
Coste	6
Valor	8
Dependencias	H2.2
Condiciones de Satisfacción	Puedo acceder al servicio 24/7

Para el tema 3, Aplicación móvil

Tengo estos recursos para desarrollar este tema:

- Entorno Android Studio v2.1
- Móvil Sony Xperia L para las pruebas
- Servicio web para obtención de datos

Las **historias de usuario** iniciales:

ID	H3.1
Nombre	Diseño de interfaz
Descripción	Como usuario quiero tener una interfaz clara y usable
Coste	8
Valor	9
Dependencias	-
Condiciones de Satisfacción	-Puedo utilizar la aplicación de forma fácil -La interfaz no es una traba para la funcionalidad.

ID	H3.2
Nombre	Gestión de consultas
Descripción	Como usuario quiero crear, modificar y guardar consultas de la sensibilidad bacteriana
Coste	7
Valor	8
Dependencias	-H2.3
Condiciones de Satisfacción	Puedo gestionar consultas

ID	H3.3
Nombre	Resultados
Descripción	Como usuario quiero poder introducir la bacteria, los antibióticos y las mediciones y obtener los valores de sensibilidad
Coste	8
Valor	10
Dependencias	-H2.3
Condiciones de Satisfacción	-Obtengo datos correctos de forma rápida -Veo los resultados de forma clara -Puedo almacenar los resultados

ID	H3.4
Nombre	Base de datos
Descripción	Como usuario quiero poder almacenar los resultados y utilizar la aplicación de forma offline
Coste	8
Valor	9
Dependencias	-H2.3
Condiciones de Satisfacción	-Puedo almacenar consultas y resultados -Puedo obtener resultados sin conexión al servicio al estar los datos en el móvil

ID	H3.5
Nombre	Almacenamiento externo
Descripción	Como usuario quiero exportar las consultas y resultados a través de una cuenta de almacenamiento en la nube como Google Drive o similar
Coste	7
Valor	3
Dependencias	-H3.2 -H3.3 -H3.4
Condiciones de Satisfacción	-Puedo conectarme a una cuenta de servicio de almacenamiento y exportar los datos

Pila del producto

La pila inicial del producto prevé un que todos los requisitos de la aplicación se llevar

Tema	**ID**	**Nombre**	**Importancia**	**Dependencias**	**Coste (h)**	
Sprint 0		Planificación e investigación	-	-	40	
1	H1.1	XML Schema	8	-	20	70
	H1.2	XML	10	H1.1	50	
2	H2.1	Consultas XQUERY	8	H1.2	10	60
	H2.2	Servicio en Java	10	H1.2, H2.1	40	
	H2.3	Despliegue del servicio	9	H2.2	10	
3	H3.3	Resultados	10	H2.3	30	130
	H3.2	Gestión de consultas	9	H2.3	30	
	H3.1	Diseño de interfaz	8	-	30	
	H3.4	Base de datos	9	H2.3	20	
	H3.5	Almacenamiento externo	3	H3.2, H3.3, H3.4	20	
Total					**300**	

Tabla 1. Pila inicial del producto

En la *Tabla 1* se refleja la pila inicial del proyecto. El orden de las historias de usuario a realizar viene determinado por su importancia, coste y dependencias.

Aplicando la técnica MoSCoW separo por importancia:

- M (Must): H1.1, H1.2, H2.1, H2.2, H3.3, H3.2
- S (Should): H2.3, H3.1
- C (Could): H3.4, H3.5
- W (Won't): El plan inicial prevé el total desarrollo del proyecto.

Organización temporal

Los sprints serán inicialmente de dos semanas de duración, pero pueden cambiar según se desarrolle el proyecto. Si hay cambios se reflejarán al final de cada sprint y se reordenará la pila del producto.

Habrá un total previsto de 6 sprints, uno de ellos de tres semanas en lugar de dos.

Cabe mencionar que empecé a recopilar información sobre metodologías ágiles, a obtener los entornos necesarios y a organizar mis tareas para optimizar la realización del proyecto y crear el sprint 0. Todo esto llevó lo que un sprint normal, las dos primeras semanas.

Plan de pruebas

Realizaré pruebas al final de cada sprint. Las pruebas que podría realizar son:

- Pruebas unitarias: Pruebas para asegurar que el código es correcto y óptimo. Con Netbeans utilizaré JUnit y con Android Studio el sistema que incorpora Gradle.
- Pruebas de integración: Una vez pasados los test unitarios realizaré pruebas de integración. Para ello no utilizaré software específico si no que las haré yo.
- Pruebas con herramientas específicas como la de NetBeans para el funcionamiento de un servicio web.

Análisis

El proyecto comienza realizándolos antibiogramas y analizando el documento Excel *v_6.0_Breakpoint_table.xlsx* descargado desde la web de EUCAST. EUCAST es el comité europeo encargado de realizar los análisis y publicar los resultados de la susceptibilidad bacteriana.

El documento Excel con los resultados es actualizado de forma anual. En este proyecto ya estoy utilizando la versión de 2016.

El procedimiento es el siguiente:

1- Se aplican gotas de diferentes antibióticos y en diferentes concentraciones sobre cultivos bacterianos.
2- Después del tiempo necesario, se miden el diámetro generado por cada antibiótico.
3- Se accede al documento de EUCAST, se busca la familia de la bacteria y cada uno de los antibióticos probados.
4- Para cada uno de los medicamentos el documento tiene dos valores, S y R. Si el diámetro del antibiótico es **mayor o igual a S** significa que la bacteria es **sensible** al medicamento. Si el diámetro es **menor a R** significa que es **resistente**. Es posible que el valor esté entre [S, R) por lo que la sensibilidad de la bacteria es **intermedia**. Finalmente mencionar que algunos antibióticos no están testeados aún sobre la bacteria en concreto. En el *Anexo 2* se muestra un ejemplo.

Este procedimiento se hace un poco pesado al tener que navegar a través del documento Excel. Por tanto, este proyecto pretende agilizar el proceso y hacerlo más cómodo.

Contenido del documento Excel

El documento tiene información adicional en las primeras páginas como notas sobre los cambios con respecto al documento del año anterior, el índice del contenido y notas con ejemplos sobre la forma de usar el documento. En la *Figura 2* está descrito el contenido de forma más extensa de las páginas correspondientes a las familias bacterianas.

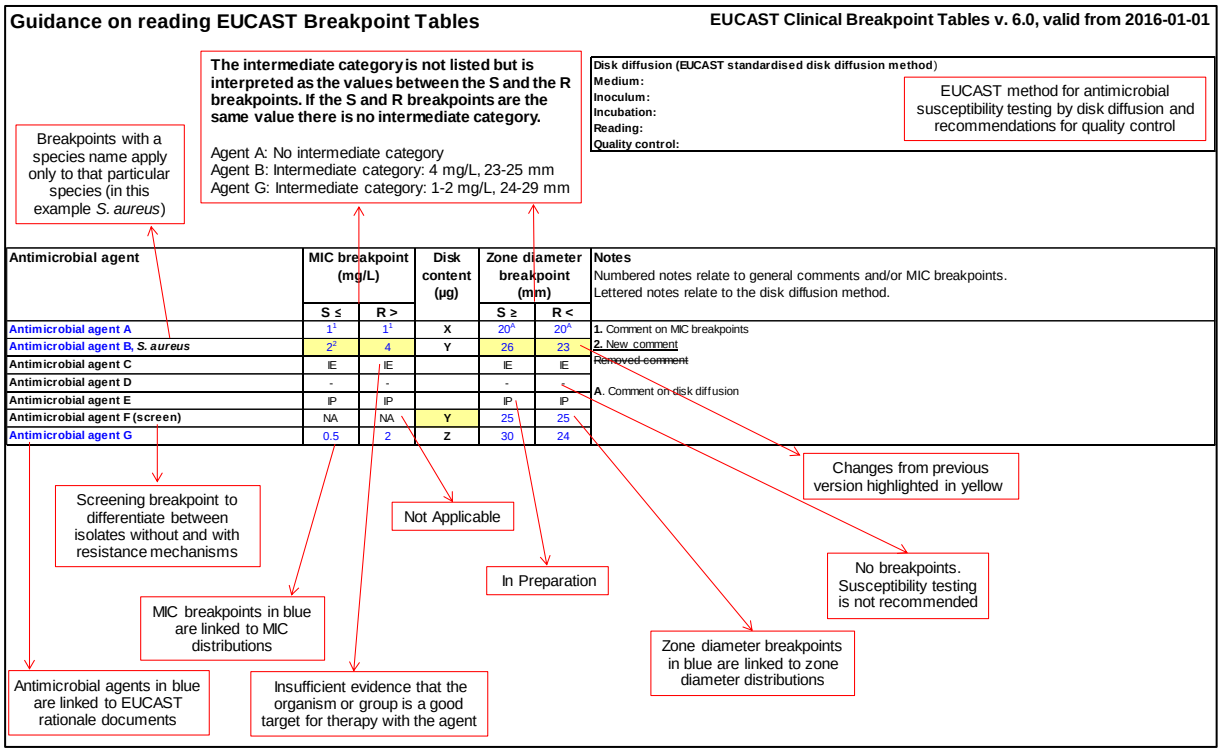
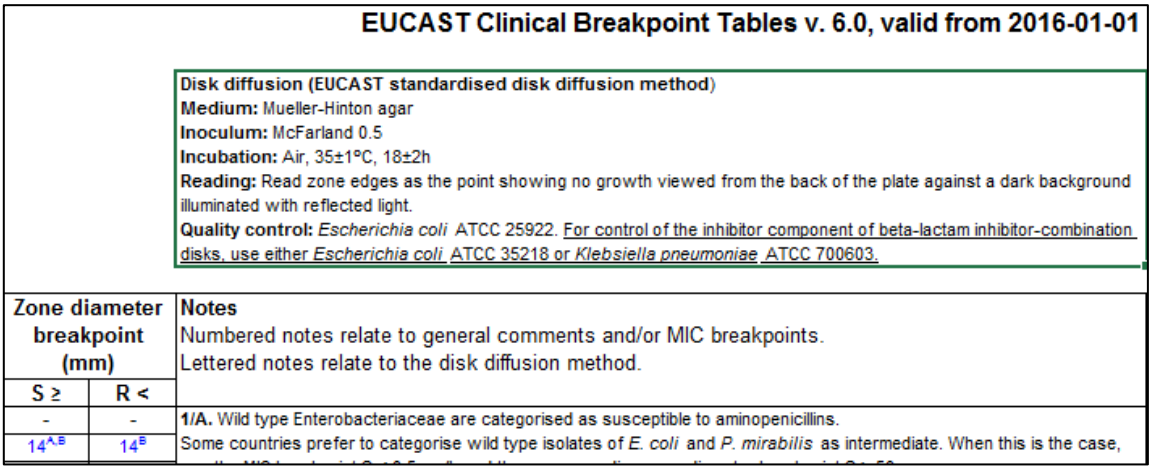


Figura 2. Descripción de las tablas de breakpoints

El documento recoge mucha información sobre la familia bacteriana. Lo que nos concierne principalmente es el nombre de cada familia de antibióticos, las notas que puedan tener, los antibióticos con sus medidas S y R (en mm, las S y R del apartado *MIC Breakpoint* corresponden a otro tipo de estudio y se dan en mg/L) y las notas y enlaces web que cada uno pueda poseer.

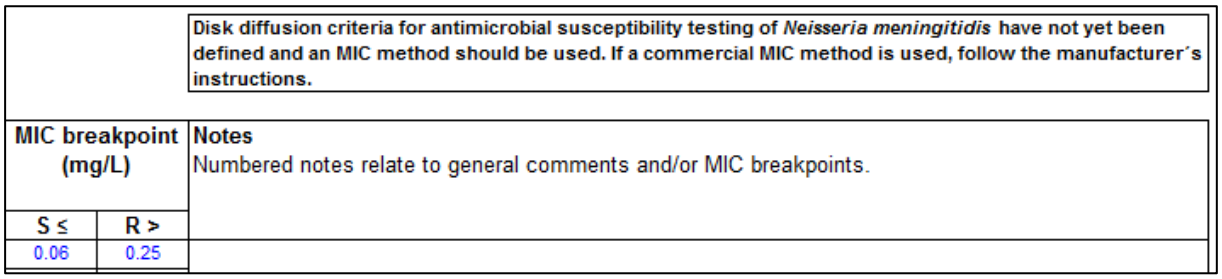
Casi todas las familias bacterianas tienen la misma estructura y son relevantes para nuestro estudio, pero algunas no tienen mediciones adecuadas.


EUCAST Clinical Breakpoint Tables v. 6.0, valid from 2016-01-01

Disk diffusion (EUCAST standardised disk diffusion method)
Medium: Mueller-Hinton agar
Inoculum: McFarland 0.5
Incubation: Air, 35±1°C, 18±2h
Reading: Read zone edges as the point showing no growth viewed from the back of the plate against a dark background illuminated with reflected light.
Quality control: *Escherichia coli* ATCC 25922. For control of the inhibitor component of beta-lactam inhibitor-combination disks, use either *Escherichia coli* ATCC 35218 or *Klebsiella pneumoniae* ATCC 700603.

Zone diameter breakpoint (mm)		Notes
S ≥	R <	Numbered notes relate to general comments and/or MIC breakpoints. Lettered notes relate to the disk diffusion method.
-	-	**1/A.** Wild type Enterobacteriaceae are categorised as susceptible to aminopenicillins.
14[A,B]	14[B]	Some countries prefer to categorise wild type isolates of *E. coli* and *P. mirabilis* as intermediate. When this is the case,

Figura 3. Familia bacteriana Enetobacteriacea



	Disk diffusion criteria for antimicrobial susceptibility testing of *Neisseria meningitidis* have not yet been defined and an MIC method should be used. If a commercial MIC method is used, follow the manufacturer´s instructions.
MIC breakpoint (mg/L)	**Notes** Numbered notes relate to general comments and/or MIC breakpoints.
S ≤ / **R >**	
0.06 / 0.25	

Figura 4. Familia bacteriana Neisseria meningitidis

En la *Figura 3* tenemos un ejemplo de familia bacteriana con la que sí podemos trabajar. Por el contrario, la familia bacteriana de la *Figura 4* no contiene las mediciones por diámetro (mediciones por difusión).

La penúltima página del documento contiene una recopilación de mediciones de antibióticos que se consultarán en caso de no encontrar la familia analizada o si una nota lo indica expresamente. Esta página se llama *PK/PD (non-species related) breakpoints*.

Finalmente, la última página del documento describe las dosis validadas de cada uno de los antibióticos empleados en los análisis.

Tratamiento de XML

Una vez analizado el documento Excel debo extraer su información. Para ello he investigado dos librerías en Java para el tratamiento de este tipo de documentos:

- JExcel: Actualizado en 2009. Ampliamente utilizado. No soporta documentos .xlsx
- Apache POI: API actualizada. Soporta documentos .xlsx y tiene mucha más documentación.

Lo más recomendable, en mi caso, es utilizar Apache POI ya que trabajo con documentos .xlsx (superiores a versiones de 2007). Utilizaré concretamente la versión **3.14**. En la parte de implementación mostraré cómo funciona en profundidad, pero, básicamente, esta librería permite recorrer el documento página por página y celda por celda para ir extrayendo el contenido, detectar el tamaño y estilo de la fuente e incluso si contiene o no superíndices.

XQuery - Sintaxis

La web http://www.w3schools.com/ tiene muchos tutoriales de programación realmente útiles. Me introduje en la sintaxis de XQuery siguiendo uno de esos tutoriales y haciendo pruebas sobre el documento XML.

Para que el proceso me resultara más claro, relacioné mentalmente esta sintaxis con las ya conocidas: consultas en SQL e iteraciones de colecciones en Java.

Esto es una consulta básica:

```
doc("books.xml")/bookstore/book[price>30]/title
```

“Devuelve todos los títulos de los libros cuyo precio es mayor a 30”

La misma consulta se puede hacer usando la sintaxis FLWOR (“flower”). Ver *Figura 5*.

- **For** - selects a sequence of nodes
- **Let** - binds a sequence to a variable
- **Where** - filters the nodes
- **Order by** - sorts the nodes
- **Return** - what to return (gets evaluated once for every node)

Figura 5. Sintaxis FLWOR de XQuery

```
for $x in doc("books.xml")/bookstore/book

where $x/price>30

return $x/title
```

Yo preferí usar la sintaxis compacta siempre que pude.

XQuery - Java

Las librerías que mejor funcionan en Java para tratar XQuery según múltiples usuarios son las de Oracle. Para poder utilizarlas se recomienda instalar una base de datos de Oracle y enlazar las dependencias. Opté por buscar las librerías sueltas e incorporarlas al proyecto.

Conseguí la versión 4.3.0. Esta versión incorpora gran cantidad de librerías. Para mi proyecto solo necesité las ilustradas en la *Figura 6*.

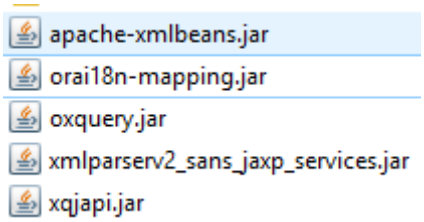


Figura 6. Librerías básicas de Oracle para XQuery

Servicio web

Para implementar el servicio web sopesé hacerlo en PHP o en Java. Decidí hacerlo en Java al tener más conocimiento de este lenguaje y poder tener algo más de soltura en caso de complicaciones. Como proyecto personal quiero recrear el mismo servicio en PHP próximamente.

Existen muchas formas de implementar un servicio web en Java. Al analizar las diferentes opciones he optado por implementar un servicio RESTful ya que su desarrollo es relativamente sencillo pero con potente funcionalidad. Además es rápido y ligero gracias a que no tiene estados.

El servicio tiene la estructura de una API REST. Intercambia mensajes HTTP para su funcionamiento. Recibe concretamente dos tipos de peticiones:

- GET: Envía información del servicio
- POST: Recibe parámetros y devuelve un mensaje JSON con los resultados

El resto de peticiones (PUT, DELETE…) no están implementadas. Pensé en implementar PUT para cambiar el documento XML, pero prefiero hacerlo por ftp.

Siempre se utilizan los mensajes HTTP adecuados (200 OK, 404 NOT FOUND…).

El IDE NetBeans incorpora funcionalidades para ayudar con la construcción y testeo de este tipo de servicios web, hecho que reforzó mi decisión de emplearlo.

Aplicación móvil

Encontré diversas formas de consumir un servicio RESTful. En Android hay que tener una serie de precauciones ya que hay problemas de compatibilidad con algunas API.

Yo había trabajado en la empresa de prácticas con la API de Apache para conexiones y la *loopj* de Android, pero éstas presentaban muchos conflictos. Sin ir más lejos, la de Apache dejó de funcionar para la API 23 de Android (Android 6.0+) y he elegido desarrollar desde la versión 4.2 hasta la 6.0 y superiores, por lo que no pude utilizarla.

Las búsquedas me llevaron a elegir *Volley* de Google. Esta API permite conexiones bastante rápidas con servicios REST. No está recomendada para comunicarse con mensajes muy extensos ya que se basa en una funcionalidad de cola y caché, almacenando las peticiones realizadas para, en caso de utilidad, recuperarlas en lugar de realizar una nueva conexión.

Como este proyecto no utiliza una gran cantidad de datos, esta librería es más que suficiente para cubrir todas las necesidades.

Otros factores importantes a elegir a la hora de desarrollar una aplicación móvil son elegir las versiones a las que está destinada y los dispositivos que soporta.

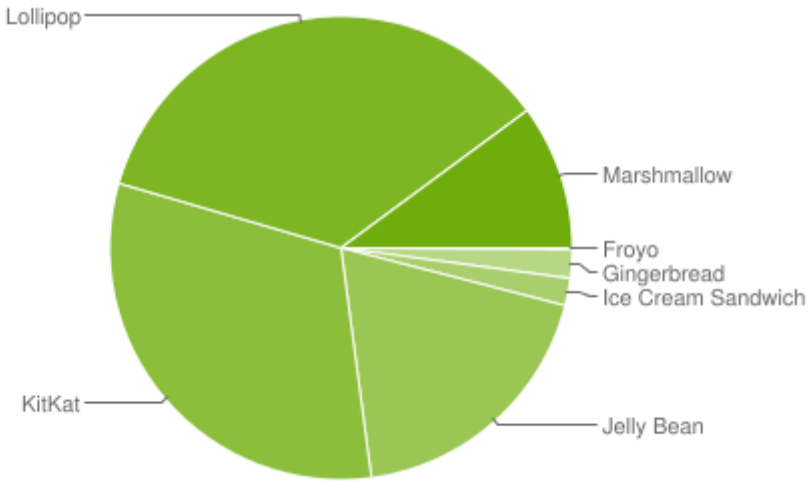


Figura 7. Porcentaje de uso de cada versión de Android. Fuente: https://developer.android.com/about/dashboards/index.html?hl=es

Basándome en los datos de la *Figura 7* he decidido implementar mi aplicación para versiones de Android 4.2 o superiores, cubriendo así el 89.2% del mercado.

En cuanto a los dispositivos que soporta la app, podrá funcionar tanto en móviles como en tablets y adaptarse a los tamaños de forma dinámica.

Diseño

El primer paso empleando la información recopilada durante el análisis es desarrollar el XML Schema que se empleará para validar el documento XML generado por el programa.

El segundo paso es crear una estructura de clases Java que corresponda al esquema y almacene de forma temporal los datos para construir el documento XML.

El paso siguiente es diseñar la interfaz del servicio web. Finalmente diseñaré la interfaz de la aplicación móvil.

XML Schema

Este esquema refleja directamente al contenido del documento Excel y tiene la siguiente estructura:

Un elemento raíz llamado **BreakpointEUCAST** que tiene el atributo obligatorio **year** y luego una secuencia de elementos **Family** (mínimo uno) y, de forma opcional, un elemento **Dosages**.

A su vez, **Family** y **Dosages** son elementos de tipo compuesto que contienen atributos y elementos.

También he creado un tipo simple **Note** para reutilizarlo ya que aparece repetidas veces en diferentes partes del esquema. En la *Figura 8* está la estructura general del esquema. El *Anexo 1* contiene la estructura completa.

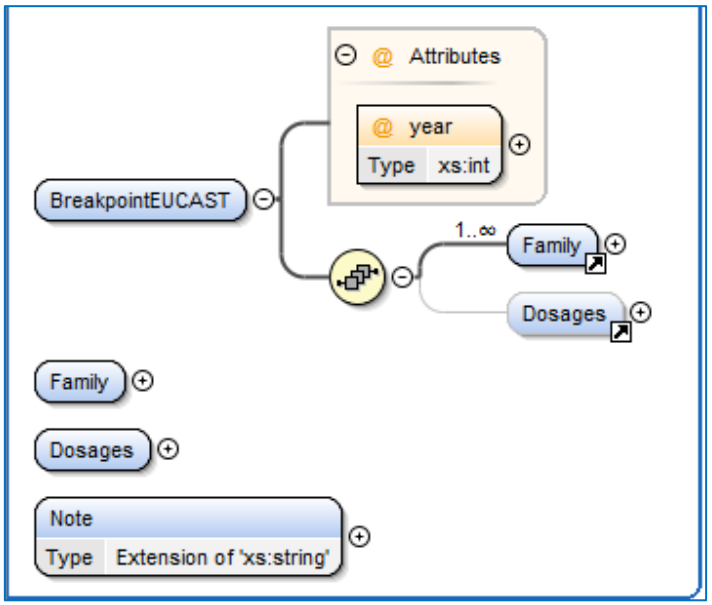


Figura 8. Resumen de la estructura del esquema

Paso a analizar en más detalle cada elemento. El primero es *Family*. Este nodo corresponde a una página del Excel, es decir, a una familia bacteriana (*Figura 9*).

Como atributos tiene *defined* que determina si es o no un tipo de familia que podemos analizar, y *name* que almacena su nombre.

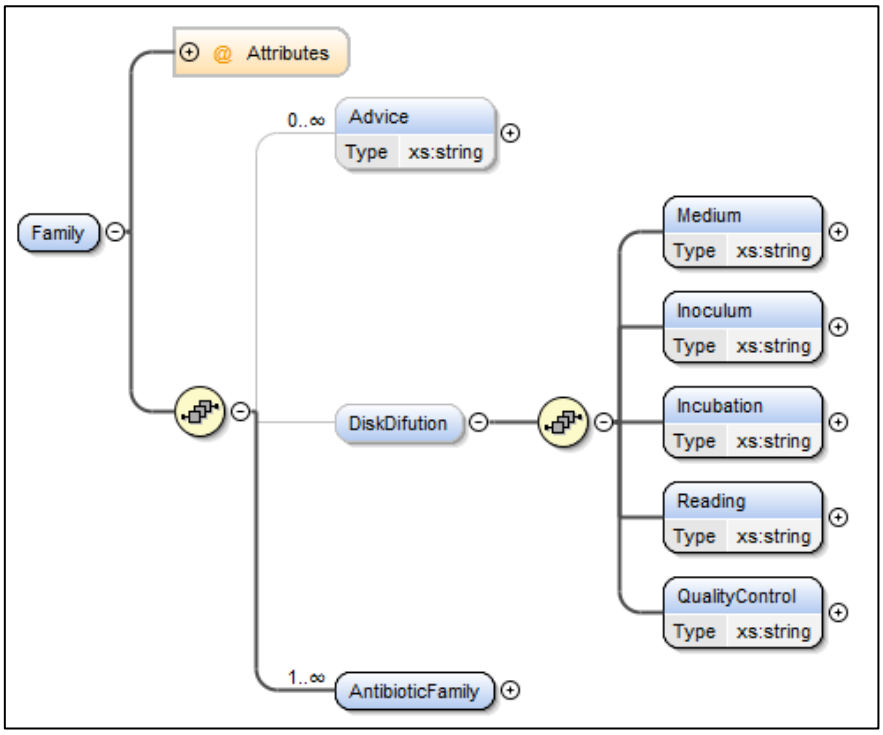


Figura 9. Estructura del nodo Family

Algunas familias tienen "avisos" que puede ser interesante almacenar. El *Anexo 3* recoge un ejemplo. Las familias que sí podemos analizar (*defined*) contienen un apartado con información referente a las medidas y cantidades empleadas para su análisis llamado *Disk Difution* (*Figura 10*).

Disk diffusion (EUCAST standardised disk diffusion method)
Medium: Mueller-Hinton agar
Inoculum: McFarland 0.5
Incubation: Air, 35±1°C, 18±2h
Reading: Read zone edges from the back of the plate against a dark background illuminated with reflected light (see below for specific instructions).
Quality control: *Escherichia coli* ATCC 25922

Figura 10. Disk Difution

El último hijo de Family es AntibioticFamily (*Figura* 11) que representa "una o varias familias de antibióticos". Este tiene su atributo name y un nodo Notes que puede aparecer o no y contiene todas las notas de esta familia de antibióticos (ejemplo en el *Anexo 4*).

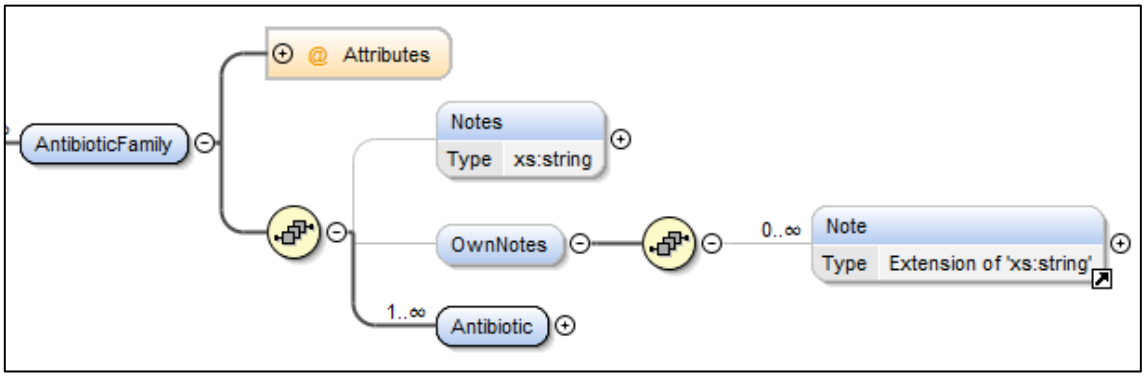


Figura 11. AntibioticFamily

Contiene también el nodo *OwnNotes* que es una secuencia de elementos *Note* (puede no estar) que representan las posibles notas indicadas en forma de superíndice pertenecientes a la familia antibiótica (en la *Figura 12* la familia *Penicillins* tiene asignada la nota *1* por superíndice).

Su último nodo es una secuencia, con mínimo de 1, de elementos *Antibiotic*. La *Figura 12* muestra la familia de antibióticos *Penicillins* con su lista de antibióticos.

Penicillins[1]	MIC breakpoint (mg/L)		Disk content (µg)	Zone diameter breakpoint (mm)	
	S ≤	R >		S ≥	R <
Benzylpenicillin	-	-		-	-
Ampicillin	8[1]	8	10	14[A,B]	14[B]
Ampicillin-sulbactam	8[1,2]	8[2]	10-10	14[A,B]	14[B]
Amoxicillin	8[1]	8	-	Note[C]	Note[C]

Figura 12. Penicillinis con algunos de sus antibióticos

El nodo *Antibiotic*, como vemos en la *Figura 13*, tiene el atributo *name*, un posible *Link*, una secuencia de *Notes* que cumple la misma función que las *OwnNotes* de *AntibioticFamily* y puede no aparecer, un nodo *MICBreakpoint* con mediciones *S* y *R*, nodo *DiskContent* con valor simple y opcional y uno o ningún nodo *ZoneDiameterBreakpoint* con mediciones *S* y *R*. Este último nodo es el que más nos interesa ya que es el que contiene la información para determinar la sensibilidad de la bacteria ante este antibiótico.

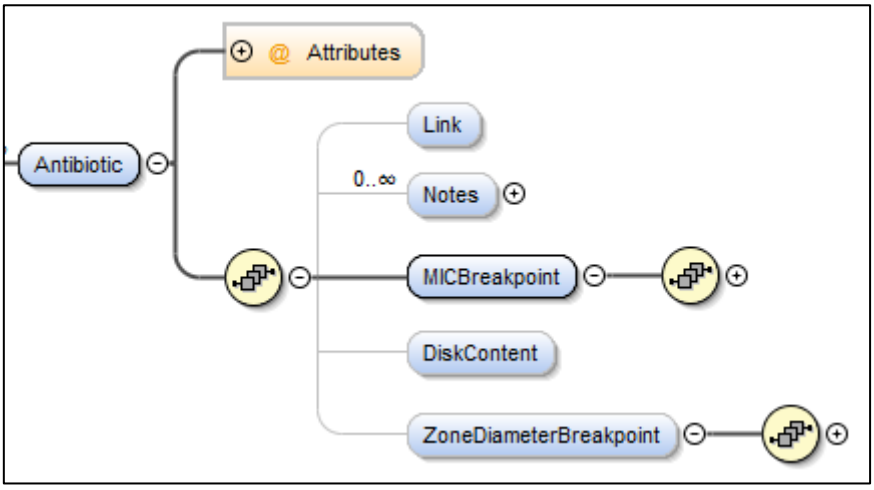


Figura 13. Nodo *Antibiotic*

Concretamente, la información de dicha sensibilidad es almacenada en los nodos *S* y *R* que tienen la estructura recogida en la *Figura 14*.

Cada uno de estos nodos tiene un *Value* que es el valor que vamos a consultar para compararlo con el diámetro medido, un posible *Link* y una posible lista de elementos *Note*.

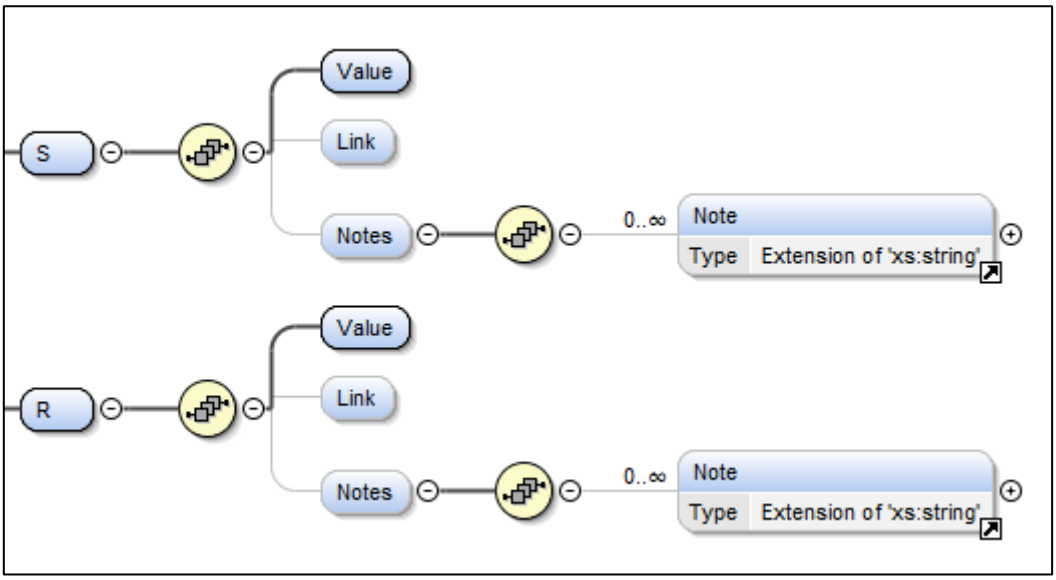


Figura 14. Nodos *S* y *R*

Representación en Java

El diagrama de clases equivalente al esquema anterior está recogido de forma resumida en la *Figura 15* y de forma completa en el *Anexo 7*.

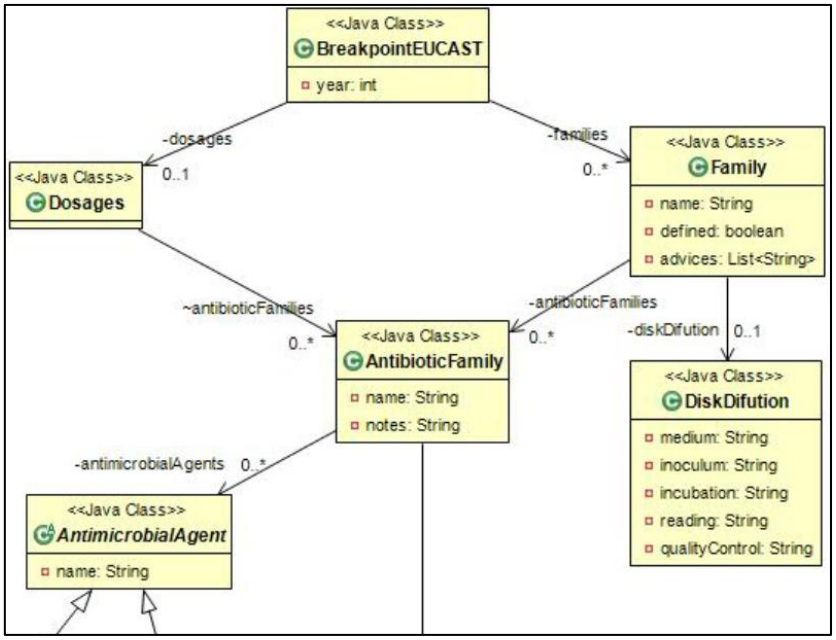


Figura 15. Recorte del diagrama de clases

Servicio web

La fase de diseño de esta tarea reside en su interfaz. (Entendemos interfaz como el medio por el cual se comunica el servicio con el exterior, su “superficie de contacto”).

El servicio tiene tres elementos

- /EUCAST
- /EUCAST/EUCAST_breakpoints/analyzer
- /EUCAST/EUCAST_breakpoints/analyzer/families

En */EUCAST* se obtiene una página HTML básica con información sobre el servicio.

Por su parte, en */EUCAST/EUCAST_breakpoints/analyzer* está alojado el servicio y recibe peticiones GET y POST.

Finalmente, en */EUCAST/EUCAST_breakpoints/analyzer/families* se aloja otro método GET que permite obtener todas las familias bacterianas y antibióticas.

Utilicé las peticiones GET para comprobar si el servicio estaba operativo.

Las peticiones POST que son las que realmente obtienen resultados del servicio se procesan en el método *checkSensibility()*.

El método *checkSensibility()* (*Figura* 16) recibe los parámetros de *family, antibiotic* y *diameter* que utiliza para realizar las consultas XQuery.

```
//el método devolverá un json bien formado si todo es correcto
//si hay errores de sintaxis o falta de parámetros devolverá el error html correspondiente
//si no encuentra los parámetros suministrados devolverá un json {"resistance":"not found"}
//el cliente deberá controlar este caso (se ha elegido pasar el error por json para agilizar
//las operaciones del cliente)
@POST
@Consumes("application/x-www-form-urlencoded")
@Produces(MediaType.APPLICATION_JSON)
public Response checkSensibility(@FormParam("family") String familiaBacteriana,
        @FormParam("antibiotic") String antibiotico,
        @FormParam("diameter") float diametroHalo){
```

Figura 16. Cabecera del método *chekSensibility()*

Puede devolver:

- 200 OK con el JSON correspondiente a la respuesta de la consulta
- 200 OK con el JSON *{"sensibility":"not found"}* si la consulta no obtiene resultados
- 400 ERROR con el mensaje *Incorrect parameteres* si falta alguno de los parámetros de entrada
- 404 ERROR con el mensaje *Source file not found* si no encuentra el documento XML

El JSON correcto de la consulta tiene la estructura: **{"antibioticFamily", "antimicrobial", "sensibility"}.** En el *Anexo 6* hay un ejemplo de una respuesta JSON completa.

Por su parte, el método *getFamilies()* (*Figura* 17) no recibe parámetros y devuelve siempre 200 OK con un JSON bien formado.

```
@GET
@Path("families")
@Produces(MediaType.APPLICATION_JSON)
public String getFamilies() {
```

Figura 17. Cabecera del método *getFamilies()*

Si todo funciona correctamente devolverá el JSON *{"families":[…], "antibioticFamilies":[…]}*. Si no, devolverá {"families":"error"}. Este método tiene una respuesta más sencilla que el anterior ya que el único error posible es que no encuentre el documento XML en el servidor para hacer las consultas.

Aplicación móvil

Uso Android Studio v2.1.2 para empezar la app. En el primer paso tan solo necesito probar el envío y recepción de datos.

Actualmente la mejor forma de crear una aplicación para Android es utilizando *Material Design.* Uno de los problemas más importante de Android ha sido siempre la diferencia de resoluciones de los dispositivos que lo soportan dificultando así un diseño global común. El otro gran problema es que no existían unas guías de diseño claras por lo que las aplicaciones podían ser completamente diferentes entre ellas, empeorando el *look-and-feel* de los usuarios al tener que aprender por completo cada interfaz en lugar de reutilizar conocimientos conseguidos al utilizar otras aplicaciones.

Por ejemplo, si por norma colocamos el menú con opciones arriba a la izquierda los usuarios sabrán dónde buscar las opciones en todas las aplicaciones y no tendrán que descubrirlo cada vez.

Para solucionar todo esto, Android ha desarrollado *Material Design* que es un conjunto de patrones y guías de diseño, tanto para Android como para web, que permite el desarrollo de aplicaciones de una forma mucho más ordenada y coherente además de incorporar elementos visualmente llamativos que ayuden al usuario. Material Design se ha convertido rápidamente en un estándar (sobre todo en Android) por lo que he decidido aplicarlo a mi aplicación.

Uno de los elementos clave en *Material Design* es la selección de colores. Es recomendable usar fondos muy claros y colores que destaquen. Para elegir la paleta básica de colores de mi aplicación utilicé la web http://www.materialpalette.com/. La web permite elegir dos colores y te genera la paleta recomendada como vemos en la *Figura 18*.

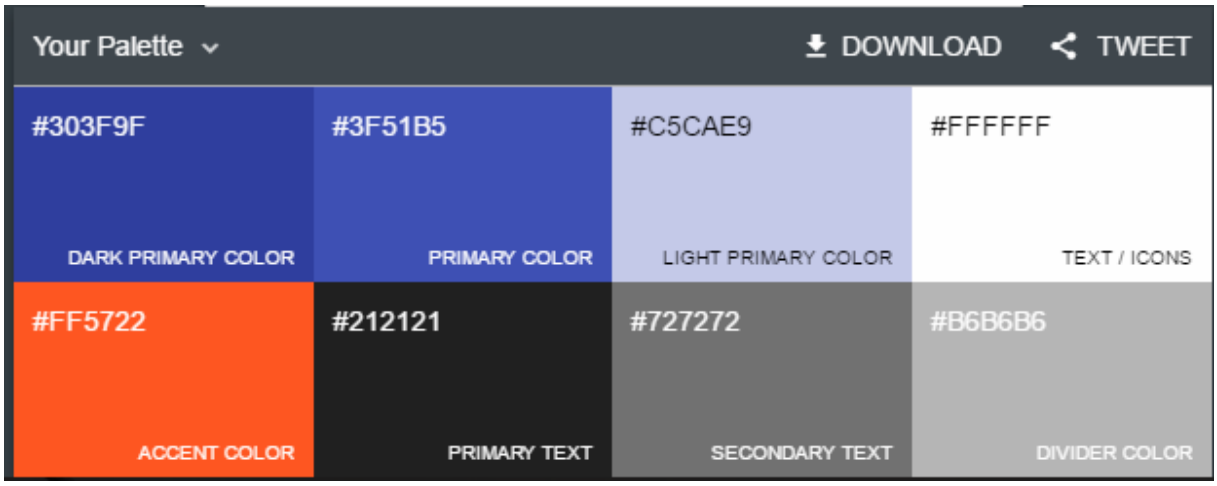


Figura 18. Paleta de colores básica de la aplicación

Creo una única Activity llamada *Principal* con tres EditText para los parámetros y un botón para realizar la conexión. También hay un TextView que mostrará la resistencia de la bacteria al medicamento. Ver *Figura 19*. *El prototipo está en castellano pero el resultado final es en inglés.

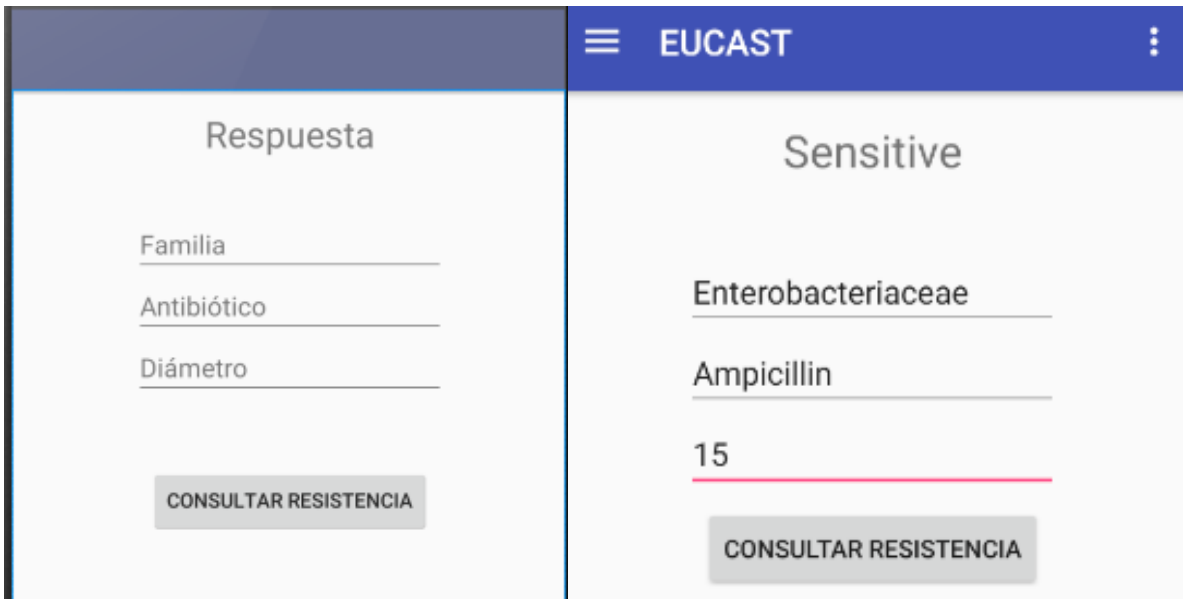


Figura 19. Prototipo inicial de la ventana de solicitud de prueba de sensibilidad

Para poder utilizar correctamente los datos recibidos del servidor debo utilizar el mismo sistema de clases empleado para generar dichos datos (*Figura 20*).

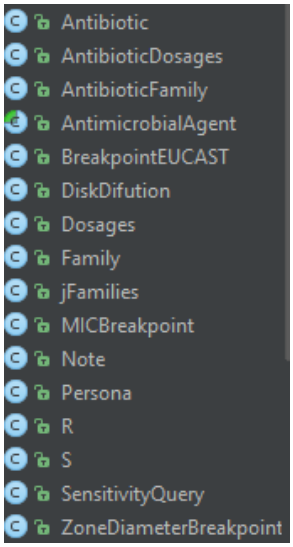


Figura 20. Clases correspondientes al modelo de la aplicación

A parte de las clases ya existentes, creo la clase *SensivilityQuery* (*Figura* 21) que refleja la respuesta del servidor para poder convertirla de forma directa de JSON a objeto *SensivilityQuery* usando Gson.

```
public class SensitivityQuery implements Parcelable {
    private String sensitivity;
    private String family;
    private String antibioticName;
    private String diameter;
    private AntibioticFamily antibioticFamily;
    private Antibiotic antibiotic;
```

Figura 21. Cabecera y atributos de *SensitivityQuery*

También tengo que importar las librerías para procesamiento de XML y JSON, así como la de Volley para las conexiones.

Creo la clase *ConnectionHandler* que se encarga de las conexiones. Esta clase tiene un método estático por cada conexión que quiera hacer. Los métodos que desarrollo inicialmente son *checkSensibility()* para comprobar la sensibilidad bacteriana y *getFamilies()* para obtener todas las familias tanto bacterianas como antibióticas.

Una vez realizada una consulta de sensibilidad se almacena en el móvil (si no hay error en la consulta) y se muestra de forma que el usuario pueda acceder fácilmente a los datos.

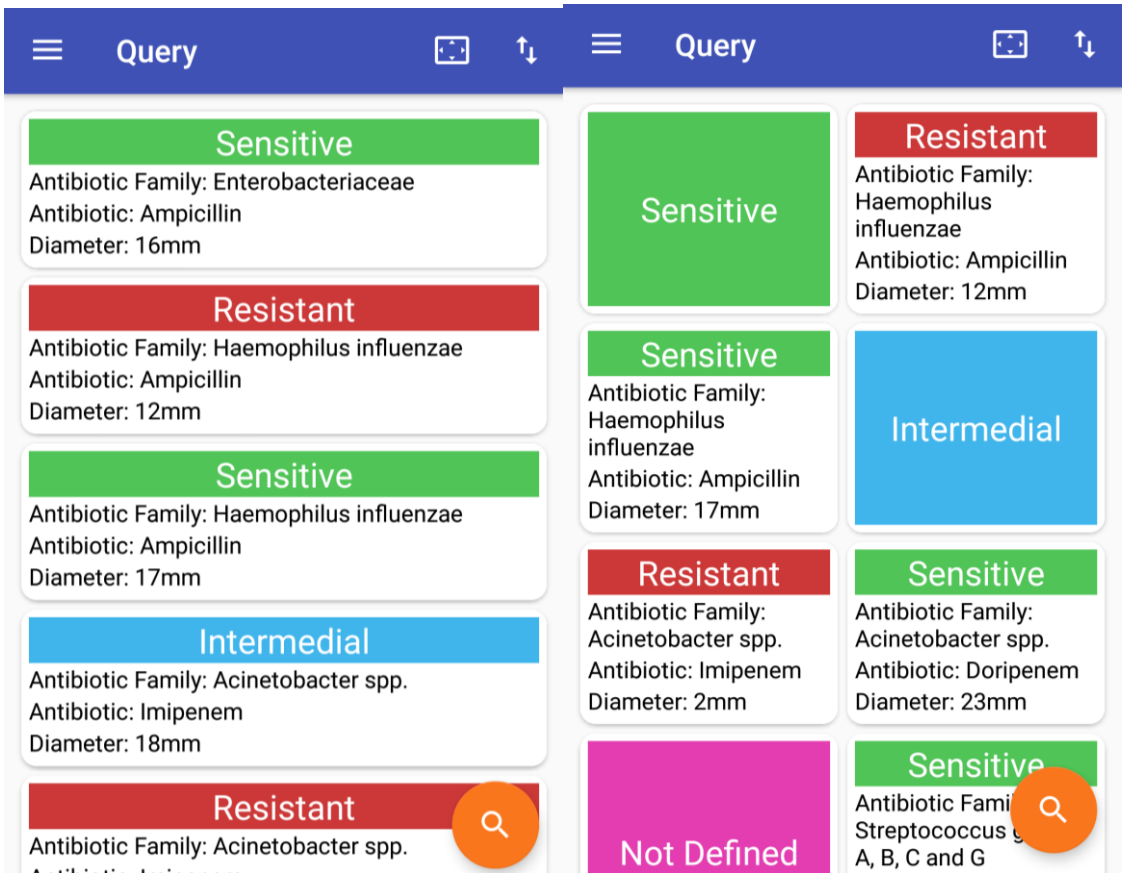


Figura 22. Lista con las consultas realizadas

En la *Figura 22* se aprecia el código de colores que sigue la aplicación según el resultado de la consulta sea Sensitive (verde), Resistant (rojo), o Intermedial (azul). Hay un cuarto color (magenta) para las consultas Not Defined que se dan cuando no hay valores de S y R definidos para el antibiótico en cuestión. La vista se puede cambiar de "lista a rejilla" o viceversa para mostrar la información a gusto del usuario.

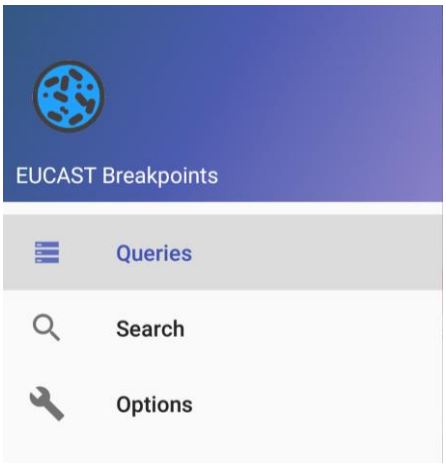


Figura 23. Menú lateral de la aplicación móvil

Para navegar por las diferentes pantallas principales de la aplicación creé un menú lateral con el aspecto de la *Figura 23*. Este tipo de menú también es un estándar hoy en día y los usuarios saben que lo pueden encontrar deslizando desde la parte izquierda de la pantalla o pulsando en el botón de hamburguesa de la esquina superior izquierda.

La base de datos de la aplicación es simple al ser una demo. No almacena todos los datos enviados por el servidor pero sí los más importantes de captar de manera visual como los valores de S y R, las notas y los links.

La base de datos tiene el esquema reflejado en la *Figura 24*.

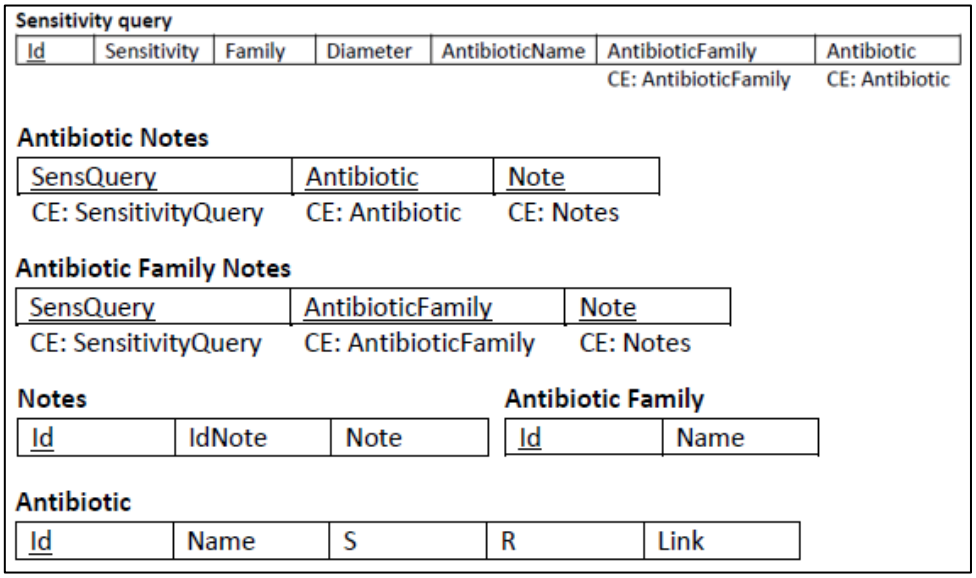


Figura 24. Base de datos de la aplicación

El funcionamiento de la base de datos es el siguiente:

1- Se realiza una consulta. Si es correcta se inserta en la tabla *SENSIBILITY_QUERY*. Esta tabla almacena la sensibilidad, familia bacteriana, antibiótico y diámetro introducidos en la consulta y, además, la familia del antibiótico introducido así como las notas que pueda tener.

2- Es posible que tanto el antibiótico como la familia del mismo se repitan durante las consultas así que las almaceno en su propia tabla y, en caso de repetirse, se devuelve el índice de la versión ya existente. De esta forma evito un crecimiento exagerado de la base de datos.
3- Todas las notas que puedan tener los elementos de la consulta se almacenan en la tabla *Notes*. En este caso también se controlan repetidos.
4- Finalmente almaceno en las tablas *ANTIBIOTIC_FAMILY_NOTES* Y *ANTIBIOTIC_NOTES* tres valores: índice de la consulta, índice del antibiótico o familia de antibiótico y el índice de la nota.

Implementación

Para crear una aplicación en Android se recomienda utilizar Android Studio, un IDE que utiliza Java para la creación de la lógica de la aplicación, XML para las vistas y Gradle para la compilación. Se puede desarrollar para Android utilizando C# a través de Xamarin, pero no es el lenguaje nativo de Android por lo que a veces surgen problemas de compatibilidad.

La última opción para crear una aplicación Android es mediante programación web (HTML5, CSS3, JS) y utilizando compiladores como Phonegap. Esta opción es interesante pero obliga a la aplicación a lanzarse mediante el navegador del móvil por lo que algunos no podrán ejecutarla a plena potencia.

Para tener el resultado más óptimo para Android decidí hacerlo con Java usando Android Studio.

Teniendo en cuenta esta decisión quise utilizar Java tanto para el programa conversor de Excel a XML como para el servicio web, mostrando así la potencia de este lenguaje.

Excel a XML

Con el entorno NetBeans creé un sistema de clases que corresponde al esquema XML. El programa recibe como argumento el nombre del fichero .xml con la información de sensibilidades antimicrobianas, lo recorre y extrae sus datos creando diferentes clases Java que representan a cada tipo de dato. Posteriormente se procesan los objetos Java creados y se crea el documento XML final que contiene todos los datos y respeta el esquema establecido. Al terminar, valida el documento XML generado con el XML Schema creado al inicio.

El programa, además del argumento obligatorio con el nombre de documento Excel, puede recibir dos más: el nombre del fichero XML a generar y el nombre del XML Schema contra el que se va a validar. Si se usa el argumento "?" se muestran las diferentes opciones del programa (*ver Anexo 5*). La *Figura 25* muestra un ejemplo de su uso.

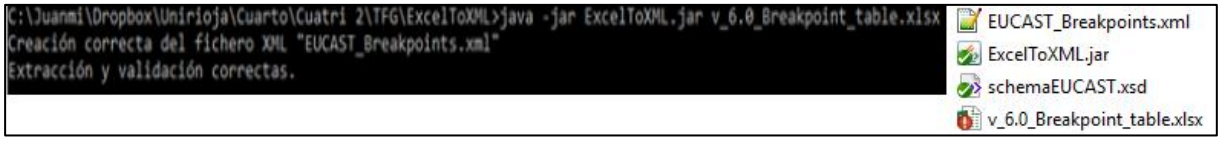


Figura 25. Ejemplo de uso del programa ExelToXML

De Excel a Java

Para la lectura del documento Excel y la extracción de los datos usé la librería Apache POI. Esta API proporciona métodos para navegar entre las hojas del documento y recorrer su contenido de forma iterativa como vemos en la *Figura 26*.

```
excel = new XSSFWorkbook(new File(args[0]));
Iterator<Sheet> iSheet = excel.sheetIterator();
while (iSheet.hasNext()) {//recorro TODO el documento hoja por hoja
    XSSFSheet hoja = (XSSFSheet) iSheet.next();
```

Figura 26. Iteración de las hojas del documento Excel

```
XSSFCell celdaDisk = getCelda(hoja, Columnas.G, 3);
if (celdaDisk != null) {
    familia.setDefined(celdaDisk.getStringCellValue().trim().contains("Disk diffusion"));
}
if(familia.isDefined()){
    procesarDefinida(familia, celdaDisk, hoja);
}else{
    procesarNoDefinida(familia, hoja);
}
```

Figura 27. Comprobación de Familia Definida o No Definida

Uno de los pasos más importantes era detectar si la Family es definida o no ya que determinaba la forma en la que había que procesarla y el contenido de la misma (*Figura 27*).

Un punto complicado e interesante fue que había contenido que podía tener notas en forma de superíndices, enlaces externos o incluso estar tachados (*Figura 28*). Para eso debía tratar esa información como texto enriquecido y no como texto normal.

8^{1}	8	**10**	$14^{A,B}$	14^{B}

~~**5/D.** Mecillinam (pivmecillinam) breakpoints relate to *E. coli, Klebsiella* spp. and *P. mirabilis* only.~~

Figura 28. Ejemplo de texto tachado y superíndices

Para ello creé diferentes métodos que utilizan funciones como *getFont().getStrikeout* para detectar fragmentos tachados o *numFormattingRuns() > 1* para detectar si contiene superíndices o formatos de letra complejos.

La última característica a mencionar es que no todas las hojas del Excel tenían la estructura idéntica por lo que hubo que detectar varios casos especiales e ir remodelando el código a modo de prueba y error.

Una vez el programa recorre todo el documento, genera el objeto Java BreakpointEUCAST con toda la información. El siguiente paso es guardar esa información en el XML.

De Java a XML

Para este paso utilicé la librería XStream (v1.4.9).

Utilizo un elemento encargado de realizar todo el proceso de transformación de tipo *XStream*. Lo primero es asignarle un alias que representa al elemento raíz del documento XML generado. Si queremos que las anotaciones se procesen de forma automática en lugar de irlas de-

finiendo manualmente sobre el objeto *XStream* debemos indicarlo con la opción *autodetectAnnotations(true)*. Por último se llama al método *toXML(Object)* pasando el objeto que representa al documento y obtenemos el resultado en forma de String que será escrito en un fichero. Este proceso lo podemos ver en la *Figura 29*.

```
//creo el objeto xstream
XStream xstream = new XStream();
xstream.alias("BreakpointEUCAST", BreakpointEUCAST.class);//elemento raíz
xstream.autodetectAnnotations(Boolean.TRUE);//analizar las anotaciones de cada objeto
xstream.aliasSystemAttribute(null, "class");//eliminar el atributo automático "class"

//obtengo en un String el documento parseado
String xml = xstream.toXML(documento);
```

Figura 29. Uso de XStream

El punto clave para que la API sepa hacer el trabajo como se desea es añadir las anotaciones adecuadas en cada uno de los atributos y clases (*Figura 30*).

```
//attributes
@XStreamAsAttribute
private int year;
//elements
@XStreamImplicit(itemFieldName="Family")
private List<Family> families;
@XStreamAlias("Dosages")
private Dosages dosages;
```

Figura 30. Ejemplo de anotaciones de XStream

Servicio web

Utilicé NetBeans 8.1 para el desarrollo del servicio web RESTful. Incorpora herramientas para facilitar este proceso así como para hacer pruebas de los servicios implementados.

El entorno estaba enlazado con un servidor Tomcat 8 usando Java 7. Inicialmente usé las versiones 8 de ambos pero había problemas de compatibilidad.

El uso de XQuery *(ver Figura 31)* en Java se basa en:

- Obtener origen de datos: A través de los objetos *OXQDataSource* y *XQConnection*
- Preparar una expresión con una consulta asociada: *XQExpression*
- Ejecutar la consulta obteniendo una secuencia de resultados: *XQSequence = XQExpression.executeQuery(query)*
- Recorrer los resultados: *while(XQSequence.next())*

```
OXQDataSource ds = new OXQDataSource();
XQConnection xqc = ds.getConnection();
XQExpression xqe;
XQSequence xqs;
xqe = xqc.createExpression();
InputStreamReader is = new InputStreamReader(
        new FileInputStream(new File(ctx.getRealPath(filePath))), "UTF-8");
xqe.bindDocument(new QName("doc"), is, null, null);
//ahora extraigo el antibiótico
query = "declare variable $doc external; ";
query += "$doc//Family[@name='" + familiaBacteriana
        + "']/Group/FamilyAntimicrobialAgent[@name='" + antibiotico + "'] ";
xqs = xqe.executeQuery(query);
String micro = "";
if(xqs.next())
    micro = xqs.getItemAsString(null);
else
    return Response.ok("{\"resistance\":\"not found\"}", MediaType.APPLICATION_JSON).build();
```

Figura 31. Uso de la API de Oracle para XQuery

Siguiendo esa estructura, el servicio realiza diferentes consultas para construir los resultados.

Concretamente hace 3 consultas:

- Obtiene la *sensibility* que puede tomar cuatro valores: *Sensitive, Intermediate, Resitant* y *Not Defined*.
- Obtiene la información básica de la *antibioticFamily* al que pertenece el medicamento: *name* y *notes*.
- Obtiene la información básica del *antibiotic*: *name, link, notes, MICBreakpoint, Zone-Breakpoin* y *DiskConten*

Los resultados de las consultas son texto plano o elementos XML. Para el texto plano, como el caso de la *sensibility*, simplemente las almacené en Strings. Los resultados XML tuve que transformarlos en objetos Java equivalentes y luego transformarlos nuevamente en JSON.

Las transformaciones se basan en utilizar los mismos objetos Java que utilicé para crear el documento XML. Estos objetos contienen la estructura correcta. Además, en el caso de la conversión de XML a objeto (unmarshall), ya cuentan con las etiquetas necesarias para hacerlo de forma automática. Para ello utilicé la librería XStream que emplea las anotaciones mostradas anteriormente (*Figuras 29* y *30)*.

EL método *fromXML()* recibe como parámetro un String correspondiente al XML. Si todo va bien, se hace una conversión al objeto adecuado.

Cuando ya tengo todos los objetos Java que necesito debo construir la respuesta. La librería que utilicé tanto para la construcción de los elementos JSON como para las transformaciones fue *gson 2.6.2* de Google.

En la *Figura 32* se ve cómo hago todas las transformaciones:

```
JsonObject jMain = new JsonObject();
JsonObject jGrupo = new JsonObject();
jGrupo.addProperty("name", name);
JsonElement jownNotes = gson.toJsonTree(ownNotes, new TypeToken<List<Note>>(){}.getType());
JsonElement jMicro = gson.toJsonTree(antimicro, Antibiotic.class);
jGrupo.add("ownNotes", jownNotes);
jMain.add("antibioticFamily", jGrupo);
jMain.add("antibiotic", jMicro);
jMain.addProperty("sensibility", sensibilidad);
return Response.ok(gson.toJson(jMain), MediaType.APPLICATION_JSON).build();
```

Figura 32. Creación del objeto JSON y envío del mismo como respuesta del servidor

Básicamente voy construyendo los elementos y los añado al elemento jMain que será el que se envíe como respuesta. Para transformar ese elemento en JSON se utiliza el método *Gson.toJson(JsonObject)*.

Un ejemplo muy simplificado de la respuesta se aprecia en la *Figura 33*. En el *Anexo 6* hay un ejemplo real.


```
{
    "group":{"name":"Penicillins","ownNotes":[{"id":"1","value":"value1"},{"id":"2","value":"value2"}]},
    "antimicrobial":{"link":"www.comomolo.com","name":"Ampicillin"},
    "resistance":"Resistant"
}
```


Figura 33. Ejemplo simplificado de JSON de respuesta del servidor

Durante las pruebas decidí desplegar el servicio en un servidor Linux propio. El servicio es accesible en http://jmv-develop.no-ip.biz:8080/EUCAST/EUCAST_breakpoints/analyzer

Finalmente el servicio se desplegó en el servidor Centos suministrado por la universidad. La ruta es http:// jmvdeveloper.unirioja.es:8080/EUCAST/EUCAST_breakpoints/analyzer

Aplicación móvil

El primer paso para la aplicación móvil fue probar la conexión con el servicio web.

Antes de explicar la estructura de mi aplicación quiero aclarar la diferencia entre Fragment y Activity. Las aplicaciones Android han funcionado siempre mediante objetos Activity. Una Activity representa una ventana de la aplicación y está compuesta por un documento XML que maneja el aspecto (colores, posición…) y un documento Java que controla la lógica.

Hasta la aparición de los Fragments cada ventana era una Activity y se cambiaba entre ellas mediante elementos Intent. Una Activity no puede contener en su interior otra Activity. Cuando una aplicación era muy grande algunos dispositivos no podían ejecutarla correctamente ya que al cambiar entre una Activity y otra se pasa por una serie de pasos (ciclo de vida de la Activity) y la creación, pausa y recuperación constante de elementos tan grandes generaba gran carga computacional.

Para aumentar el rendimiento y dotar a las aplicaciones de más dinamismo surgieron los Fragments. Podemos tener varios Fragments en una misma vista siendo totalmente independientes. Representaron una revolución a la hora de mejorar el diseño para aplicaciones destinadas a dispositivos con tamaños muy cambiantes ya que podían mostrarse más o menos Fragments según capacidad de la pantalla o su orientación.

Además, su creación y recuperación son mucho más rápidas. Lo habitual es crear una Activity que controla diferentes Fragments mejorando así el rendimiento. Siguiendo este patrón es como decidí implementar mi aplicación.

Por tanto, la estructura de la aplicación se basa en el uso de Fragments para las ventanas principales (Queries, Search y Options) contenidas en un único Activity. Además, otro Activity para los detalles de una consulta.

El caso de Options es especial. El Fragment Options que gestiona las preferencias de la aplicación es un tipo diferente de Fragment y no puede ser utilizado por el mismo gestor que los otros dos. Por ello tuve que crear una Activity adicional para gestionarlo y que sea esta Activity la que sea accesible desde el menú de navegación (*Figura 34*).

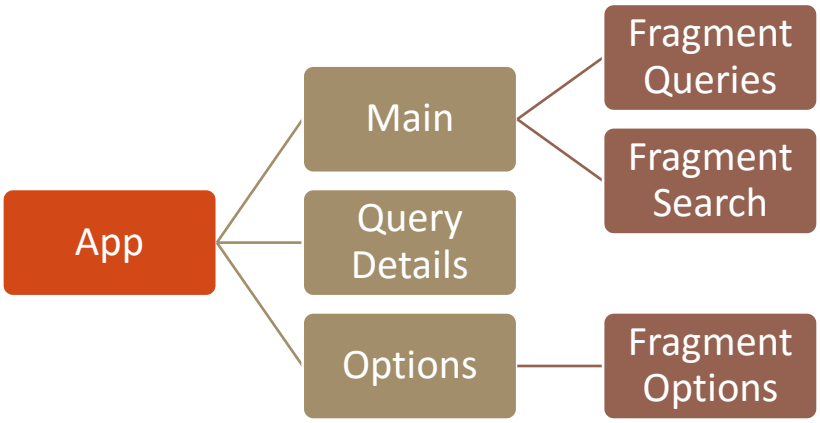


Figura 34. Estructura de la aplicación

Conexión con el servicio web

El método a implementar para la petición de la sensibilidad debe:

- Enviar parámetros recogidos por la app
- Entender si ha sido correcta la respuesta del servidor
- Procesar los datos si son correctos
- Mostrarlos y almacenarlos en la app

```
checkSensitivity(Context ctx, TextView tv, final String family,
                 final String antibiotic, final String diameter){
```

Figura 35. Cabecera del método *checkSensitivity*()

El método *checkSensitivity()* (*Figura 35)* recibe el contexto de la aplicación para poder iniciar la conexión a través de Volley, el TextView donde se mostrará el resultado y la familia, antibiótico y diámetro suministrados por el usuario.

Lo siguiente es crear la cola de peticiones (*Figura 36*). Esto sirve para meter en caché las peticiones por si se pueden reutilizar y ahorrar procesamiento de conexiones. Luego se genera la petición indicando que es de tipo POST, la URL del servicio e implementando los métodos de respuesta correcta y de error (*onResponse* y *onErrorResponse*).

```
RequestQueue queue = Volley.newRequestQueue(ctx);
StringRequest sr = new StringRequest(Request.Method.POST,
        "http://jmv-develop.no-ip.biz:8080/EUCAST/EUCAST_breakpoints/analyzer",
        new Response.Listener<String>() {
```

Figura 36. Cola de peticiones, URL del servicio y tipo de petición (POST)

Si la respuesta es correcta debo convertir el JSON obtenido al objeto *SensitivityQuery* que no es más que un típico JavaBean (*Figura 21)* con los elementos adecuados a la respuesta del servicio.

Este objeto almacenará todos los datos de la respuesta pero de momento solo quiero mostrar la sensibilidad a través del TextView (*Figura 37)*.

```
String sensibility = "ND";
Gson gson = new Gson();
SensibilityQuery rq = gson.fromJson(response, SensibilityQuery.class);
if(rq.getSensibility() == null || rq.getSensibility().isEmpty())
    rq.setSensibility(sensibility);
tv.setText(rq.getSensibility());
```

Figura 37. Asignación de la sensibilidad resultante de la consulta en la propiedad *Text* del *TextView*

Para enviar los parámetros suministrados por el usuario también debo implementar el método *getParams() (Figura 38)*. Esto pondrá los parámetros en la petición post *family=xx&antibiotic=xx&diameter=xx*.

```
protected Map<String,String> getParams(){
    Map<String,String> params = new HashMap<~>();
    params.put("family",family);
    params.put("antibiotic",antibiotic);
    params.put("diameter",diameter);
```

Figura 38. Parámetros a enviar al servidor

Ya solo queda capturar el evento de clic del botón y hacer que llame al método *checkSensitivity (Figura 39)*.

```
public void onClick(View v) {
    tv_respuesa.setText("...");
    String family = et_family.getText().toString().trim();
    String anbitiotic = et_antimicrobial.getText().toString().trim();
    String diameter = et_diameter.getText().toString().trim();
    if(!bd.isSensibilityQuery(family, anbitiotic, diameter))
        ConnectionHandler.checkSensitivity(getContext(),tv_respuesa,family,anbitiotic, diameter);
```

Figura 39. Cola de peticiones, URL del servicio y tipo de petición (POST)

En la *Figura 40* se aprecia la forma de mostrar diferentes respuestas del servicio.

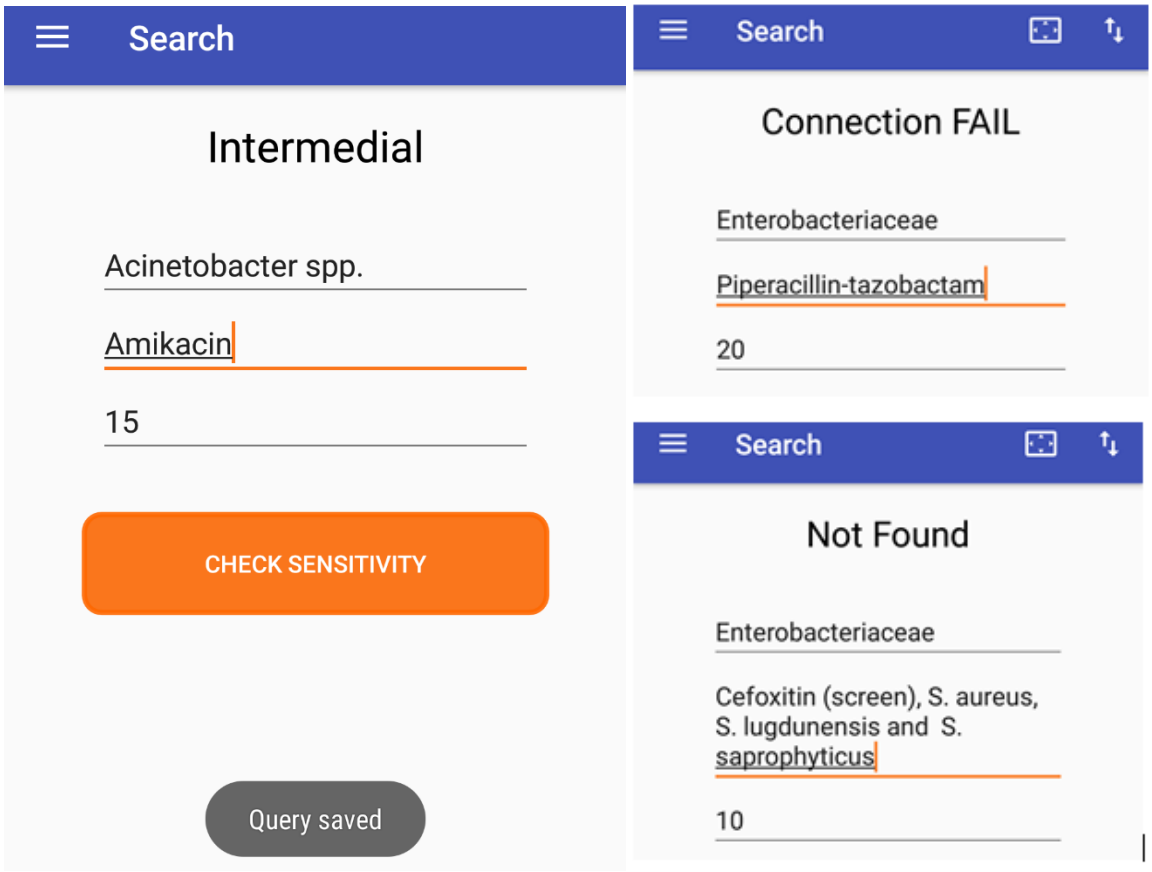


Figura 40. Respuesta correcta, problema de conexión y consulta sin resultado

El funcionamiento de la ventana *Search* pretende ser lo más sencillo posible.

El usuario introduce un resultado derivado de un antibiograma. Pulsa el botón de *Check Sensitivity* y en el *TextView* donde inicialmente pone *Response* aparecerá la sensibilidad.

Es posible que no haya conexión. En este caso el *TextView* mostrará el texto "Connection FAIL". El usuario podrá ir a *Options* para introducir de nuevo la URL del servicio y comprobar allí mismo si se ha establecido conexión. Otra posible excepción se da en el caso de que la consulta realizada no genere resultado. Por ejemplo, la familia bacteriana *Enterobacteriaceae* no tiene pruebas realizadas con el antibiótico *Cefoxitin…*

Un elemento de gran ayuda para realizar las consultas de forma precisa es el autocompletado (*Figura 42*). Al iniciar la aplicación, si la base de datos con los nombres de familias bacterianas y antibióticos está vacía se importan desde el servidor.

```
BDHandler bd = new BDHandler(getContext());
List<String> families = bd.getFamilyNames();
List<String> antibiotics = bd.getAntibioticNames();
ArrayAdapter<String> adptFamilies = new ArrayAdapter<~>(getContext(),
        android.R.layout.simple_dropdown_item_1line, families);
et_family.setAdapter(adptFamilies);
ArrayAdapter<String> adptAntibiotics = new ArrayAdapter<~>(getContext(),
        android.R.layout.simple_dropdown_item_1line, antibiotics);
et_antimicrobial.setAdapter(adptAntibiotics);
```

Figura 41. Asignación de los elementos a autocompletar

Luego, mediante un *Adapter* (*Figura 41)* se asignan los valores al *EditText* correspondiente y este sugiere los nombres según el usuario va escribiendo en él.

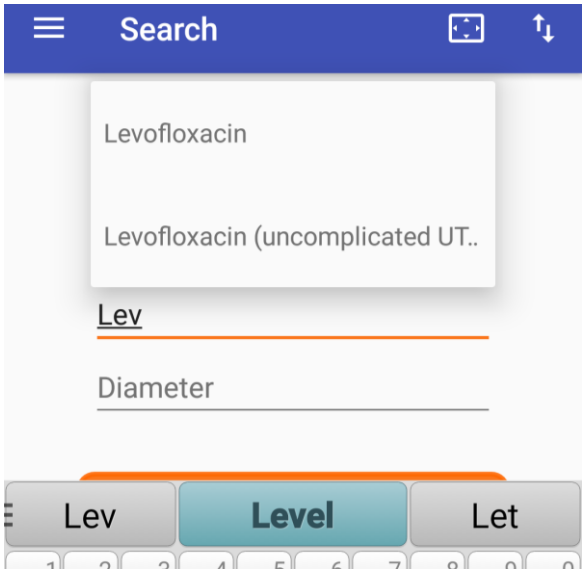


Figura 42. Ejemplo de sugerencias de autocompletado

El autocompletado asegura que los nombres que introducimos están exactamente iguales que en el servidor impidiendo así errores de sintaxis al hacer las consultas.

Opciones de configuración

Para poder dotar a la aplicación de cierta personalización y para mejorar su dinamismo he utilizado las *SharedPreferences*. Estas preferencias se almacenan en un fichero XML en la carpeta de instalación de la app. Por este motivo los cambios que almacenemos de esta forma permanecerán de manera persistente incluso al cerrar la aplicación. La *Figura 43* recoge las preferencias almacenadas en el móvil.

```
<map>
    <string name="order">DESC</string>
    <string name="url_analyzer">http://jmv-develop.no-ip.biz:8080/EUCAST/
    <string name="grid_columns">2</string>
    <boolean name="grid" value="false" />
    <string name="anterior">Queries</string>
</map>
```

Figura 43. Documento XML de preferencias

Es útil, por ejemplo, si el usuario elije la vista en *grid* en lugar de en *list*. Si no almacenamos este cambio de forma persistente, al reiniciar la aplicación volverá al estado original (*list*).

Para gestionar esta configuración creé la clase *Configuracion* que simplifica la manipulación de las preferencias. En la *Figura 44* se aprecia la diferencia entre usar los métodos de Android para crear y almacenar una preferencia y los métodos de mi clase.

```
//método completo en Android
SharedPreferences preferencias = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editor = editor = preferencias.edit();
//asignar preferecia de tipo String. Si existe queremos cambiar su valor
if (preferencias.contains(nombre))
    editor.remove(nombre);
editor.putString(nombre, valor);
editor.commit();

//método con la clase Configuracion
Configuracion.setPreferencia(nombre, valor);
```

Figura 44. Comparativa entre procedimiento nativo y mediante la clase Configuracion

Para terminar el tema relacionado con las preferencias debo decir que desarrollé una Activity que permite al usuario modificarlas (*Figura 45*).

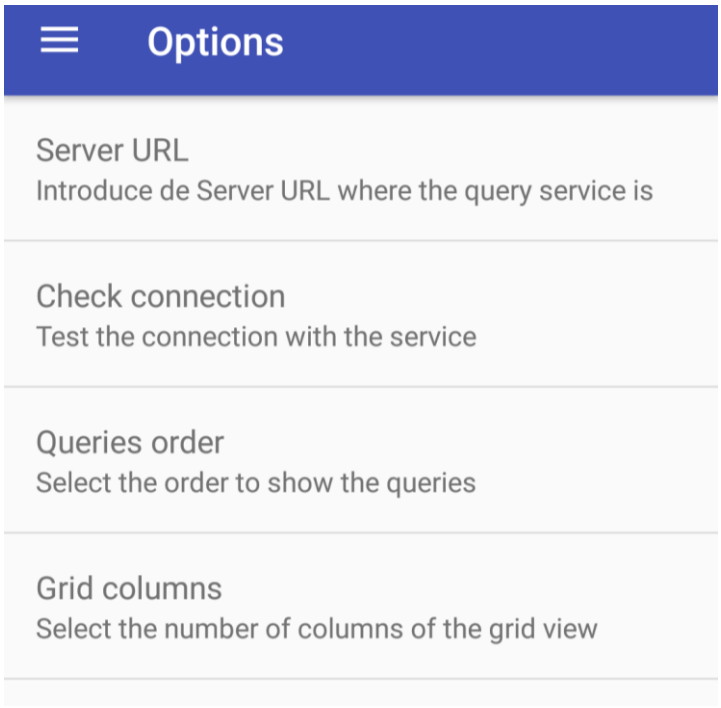


Figura 45. Vista de preferencias

Esta clase tiene las siguientes opciones:

- Server URL: Permite definir la URL donde el servicio web está escuchando.
- Check connection: Comprueba con un click si la dirección introducida realmente obtiene respuesta del servidor.
- Queries order: Establece el orden en el que se muestra la lista de consultas. También se puede cambiar con el icono de flechas de la ventana de consultas.
- Grid columns: Establece el número de columnas que tiene la vista de consultas. Por defecto se muestran dos columnas pero el usuario puede establecer las que más le convengan.

Vista de consultas

En la *Figura 22* veíamos el diseño inicial de la vista que contiene la lista con las consultas realizadas. El diseño final ha cambiado muy poco. Concretamente se han resuelto problemas de desplazamiento y se ha cambiado el icono para pasar de modo de vista como vemos en la *Figura 46*.

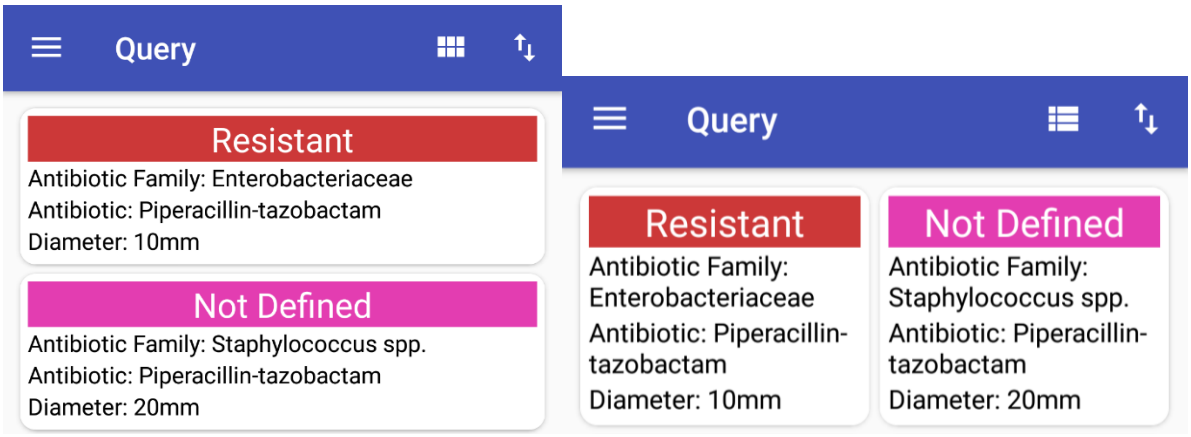


Figura 46. Diferentes vistas de la pantalla con la lista de consultas

Los elementos de la lista son elementos *CardView*, un elemento introducido en la versión 5 de Android junto a *Material Design* y que se ha vuelto muy popular al permitir al usuario identificar mejor los elementos ya que incorpora efectos de sombreado y contorno.

Cada *CardView* de esta vista tiene los siguientes elementos:

- Sensitivity: Resultado de la consulta que se muestra en la cabecera del elemento. En la *Figura 22* se aprecia el código de colores. Este código ayuda al usuario a saber de un vistazo rápido el resultado de cada consulta.
- Antibiotic Family: Familia del antibiótico consultada.
- Antibiotic: Antibiótico consultado.
- Diameter: Diámetro consultado.

Como se muestra en la *Figura 47*, si dejamos presionado sobre un elemento de la lista podremos borrarlo. Por supuesto antes de hacerlo nos sale una ventana avisando de la acción y solicitando confirmación.

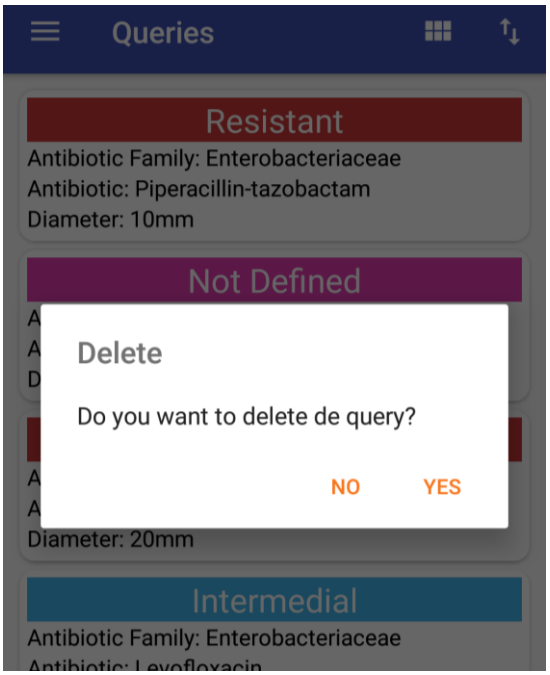


Figura 47. Confirmación de borrado de un elemento de la lista de consultas

Detalles de una consulta

Al pulsar sobre un elemento de la vista (click simple) de consultas accederemos a su ventana de detalle.

En la *Figura 48* se aprecia que la estructura de esta vista está encabezada por el elemento en el que se ha pulsado seguido por un elemento llamado *Antibiotic* y un tercer elemento llamado *Antibiotic Family*.

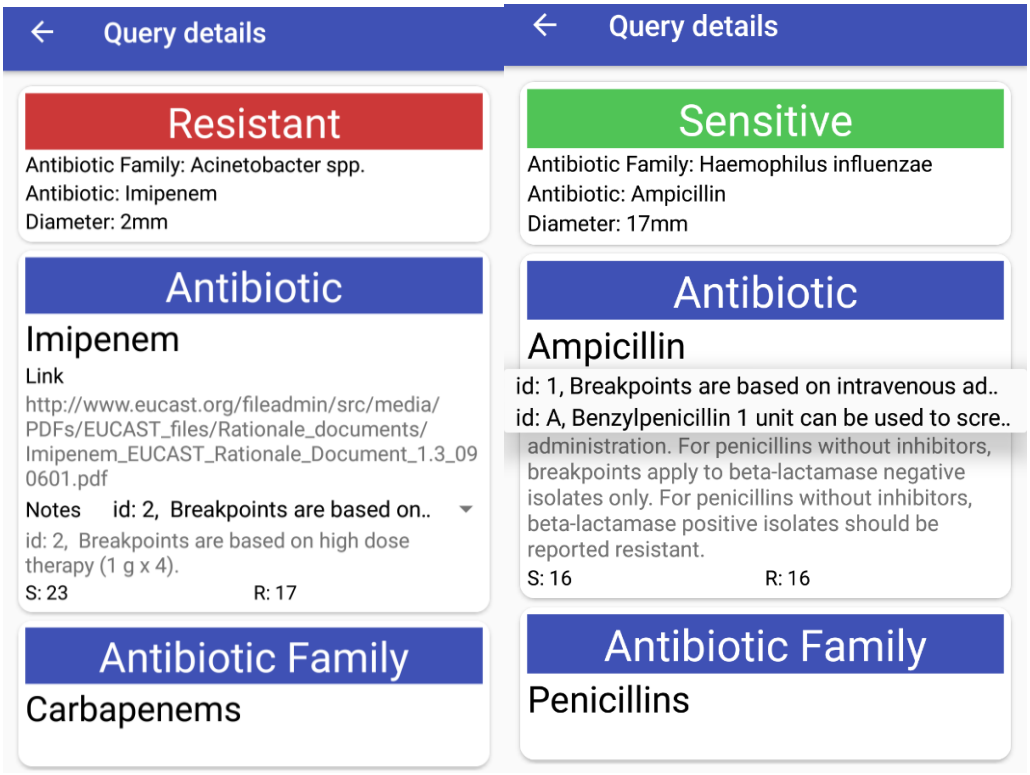


Figura 48. Vista de detalles de una consulta

El apartado *Antibiotic* tiene los elementos:

- Name: Nombre del antibiótico consultado.
- Link: Elemento opcional. Si el antibiótico tiene un enlace aparecerá en este apartado. Si se deja presionado sobre este elemento, el enlace se copiará al portapapeles y podremos pegarlo en el navegador para acceder a él.
- Notes: Elemento opcional. Si tiene alguna nota relacionada saldrá en este apartado. Las notas se agrupan en un elemento desplegable en el que se podrá elegir una. Al elegir una nota se mostrará su contenido extendido en la sección inferior.
- S y R: Valores de S y de R del antibiótico de la consulta.

Por su parte el apartado *Antibiotic Family* tan solo tiene *Name* y un elemento *Notes* igual que el apartado anterior. En la *Figura 46* se ve como ninguna de las familias bacterianas consultadas tenía ninguna nota relacionada.

Usando la aplicación *SQLite Debugger* puedo ver la base de datos de la aplicación después de las consultas mostradas en los ejemplos anteriores. En las figuras *Figura 49* y *Figura 50* se aprecian las notas y links de la *Figura 48*.

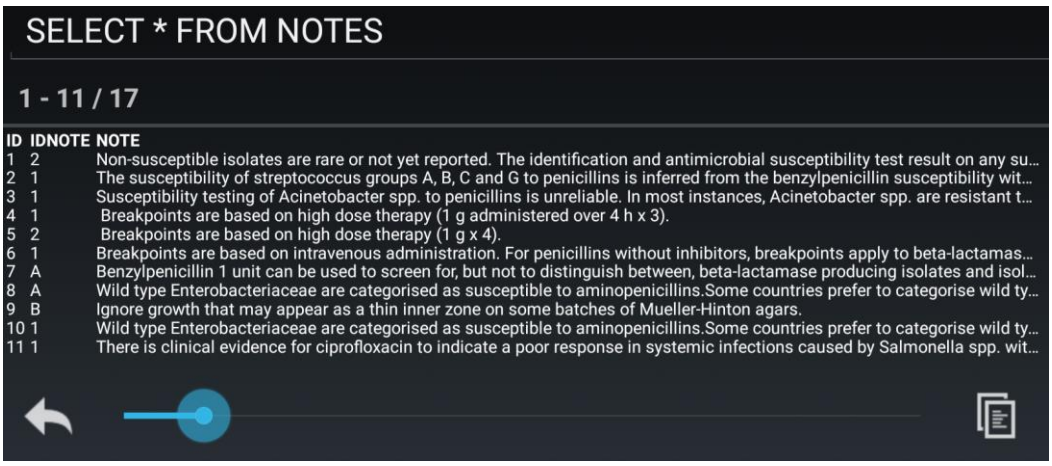


Figura 49. Notas almacenadas en la base de datos del dispositivo

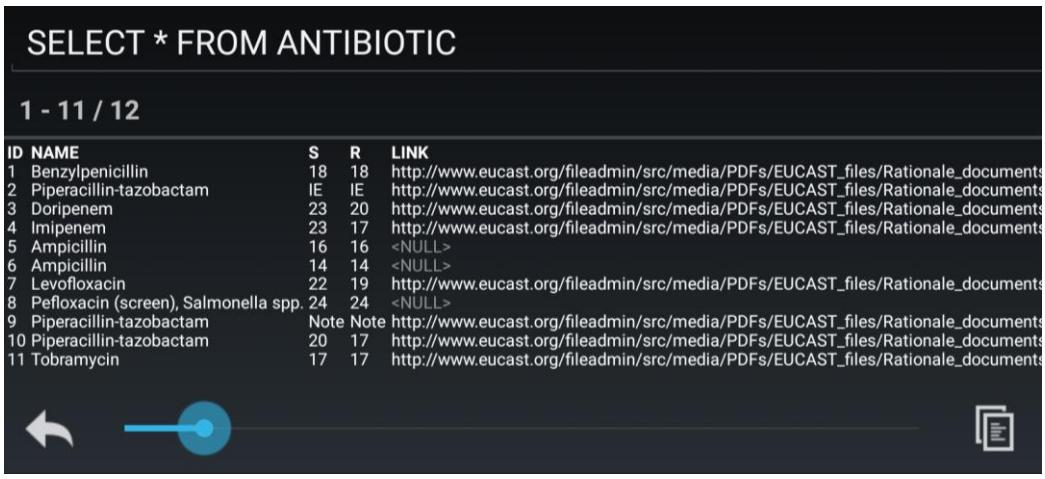


Figura 50. Antibióticos almacenados en la base de datos del dispositivo

Pruebas

Para las pruebas del servicio web utilicé la opción de NetBeans de “Test RESTful Web Services” eligiendo la opción local como se ve en la *Figura 51.*

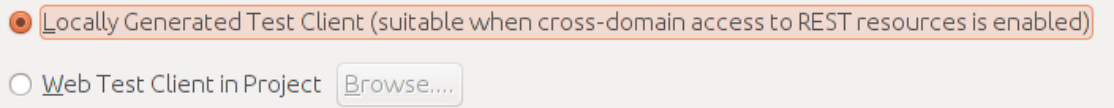


Figura 51. Lanzamiento local de la prueba del servicio REST

Esto nos despliega una web en el navegador por la que podemos acceder a los servicios web.

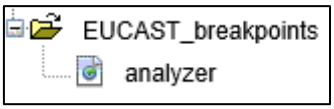


Figura 52. Estructura del servicio web probado

En */analyzer* tenemos tres tipos de petición permitida y podemos probarlas todas. En la *Figura 53* aparecen las tres opciones.

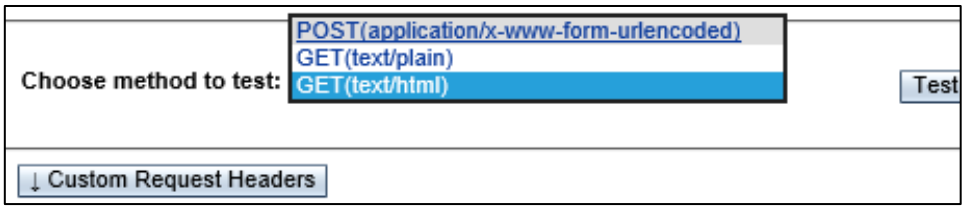


Figura 53. Peticiones permitidas en /analyzer

La petición interesante es la petición POST. En *Content* pongo los parámetros que serán enviados *Figura 54*.

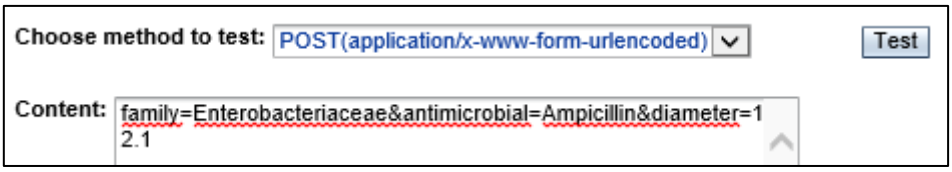


Figura 54. Parámetros de la petición POST

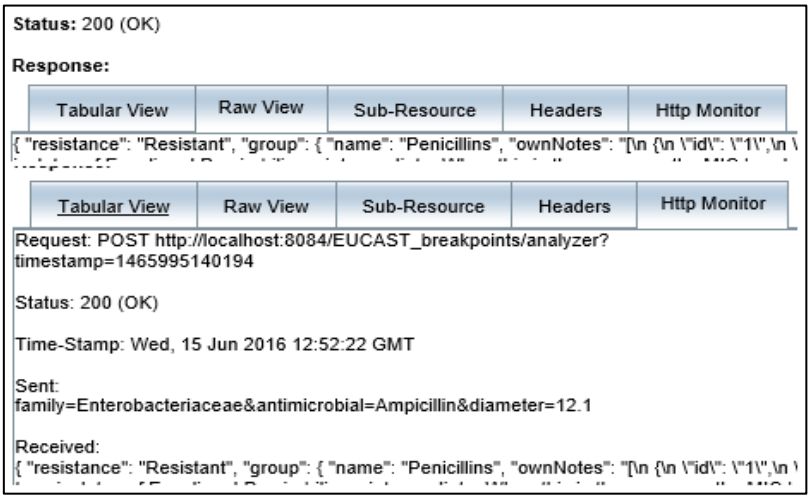


Figura 55. Visualización de la respuesta del servidor

En la opción *Http Monitor* (*Figura 55)* podemos ver la petición realizada y la respuesta recibida. Por defecto muestra el estado de la respuesta y el contenido de la misma.

Una herramienta de gran utilidad con la que pude testear la aplicación de forma cómoda.

La *Figura* 56 muestra un ejemplo de petición incorrecta (falta un parámetro). Recibo el error 400 y el mensaje *Incorrect parametrs* tal y como había programado en el servicio web.

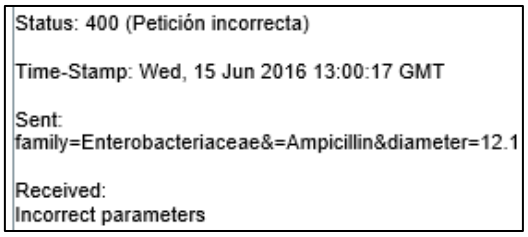


Figura 55. Parámetros de la petición POST

Seguimiento y control

El alcance del proyecto, así como su pila, cambiaron durante el desarrollo. Inicialmente se le dio bastante importancia a la aplicación móvil pero decidí centrarme más en pulir la parte de la extracción de datos y generación del XML y la parte del servicio web.

El principal motivo de esta decisión fue que hubo importantes desviaciones temporales en las dos primeras partes quedando menos tiempo para la app. Además, el cliente del proyecto remarcó que sus intereses también cambiaron orientándose más hacia el desarrollo del servicio y la extracción y validación de los datos.

Tema	**ID**	**Nombre**	**Importancia**	**Dependencias**	**Coste (h)**	
Sprint 0		Planificación e investigación	-	-	40	
1	H1.1	XML Schema	8	-	20	80
	H1.2	XML	10	H1.1	60	
2	H2.1	Consultas XQUERY	8	H1.2	10	80
	H2.2	Servicio en Java	10	H1.2, H2.1	60	
	H2.3	Despliegue del servicio	9	H2.2	10	
3	H3.3	Resultados	10	H2.3	40	100
	H3.1	Gestión de consultas	9	H2.3	30	
	H3.2	Diseño de interfaz	8	-	30	
Total					**300**	

Tabla 2. Pila final del producto

La *Tabla 2* describe la pila resultante al terminar el proyecto. Centré los esfuerzos en los dos primeros temas y reduje la complejidad de la app que pasó a ser una demo de cliente del servicio web.

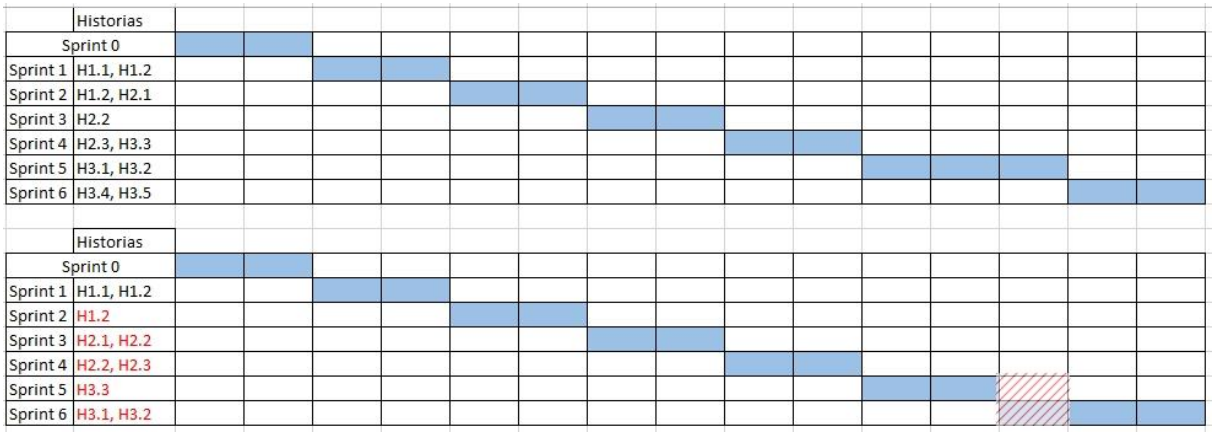


Figura 56. Diagrama del desarrollo temporal del proyecto

Como vemos en la *Figura 56*, los cambios importantes empezaron desde el primer sprint que duró más de lo planeado. El segundo también se extendió dedicándose únicamente a finalizar la parte del XML.

Durante el desarrollo del tercer sprint se decidió la nueva pila del producto y pasé a centrarme en pulir el XML e implementar el servicio web.

Para encajar los sprints de 40h moví ligeramente el cronograma y la historia H3.3, Resultados, se desarrolló íntegramente en el sprint 5.

Finalmente, el último sprint duró una semana más completando así las tareas y las 300h.

Sprints

En este apartado describo el objetivo de cada sprint así como su pila inicial, cambios sufridos y pila final.

Sprint 1: XML Schema

El objetivo del sprint es conseguir el XML Schema y empezar a desarrollar la transformación de Excel a XML.

Pila del Sprint

En este sprint trabajo con las historias H1.1 XML Schema y H1.2 XML. La *Tabla 3* recoge la pila del sprint y detalla la duración de cada una de las tareas a desarrollar.

Histo-ria	**Tarea**	**Coste (h)**
H1.1	T1.1.1 Estudio de la estructura del documento Excel	6
	T1.1.2 Estudio de la herramienta oXygen	2
	T1.1.3 Creación del XML Schema correspondiente	12
H1.2	T1.2.1 Estudio de librerías Java para uso de XML y Excel	8
	T1.2.2 Inicio del programa en Java para crear el XML a partir del Excel	12
Total		**40**

Tabla 3. Pila del sprint 1

Revisión del Sprint

En la *Tabla 4* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	Tarea	Coste estimado (h)	Coste real (h)	Desviación (h)	Motivos
H1.1	T1.1.1	6	10	4	
	T1.1.2	2	3	1	
	T1.1.3	12	13	1	La estructura era algo más compleja de lo esperado
H1.2	T1.2.1	8	10	2	Muchas pruebas hasta encontrar el uso adecuado
	T1.2.2	12	15	3	Hay implícitas varias librerías (aprendizaje de cada una) y requirió de muchas pruebas
Total		**40**	**51**	**11**	

Tabla 4. Desviaciones de la pila del sprint 1

El mayor motivo por el que perdí horas fue la dificultad de analizar los resultados para cada parte del programa. Debía ir mostrando por pantalla para verificar que se procesaba adecuadamente. Otra razón relevante fue que el documento Excel no era totalmente consistente, por lo que tuve que analizar página por página y realizar algunas adaptaciones hasta conseguir el resultado correcto.

Sprint 2: Fin XML y XQuery

El objetivo del sprint es terminar la creación del programa conversor de Excel a XML y el estudio de XQuery para usarlo en las consultas del servidor.

Pila del Sprint

En este sprint trabajo con las historias H1.2 XML y H2.1 Consultas XQuery.

Historia	Tarea	Coste (h)
H1.2	Terminar programa conversor	30
H2.1	Estudiar XQuery y probar consultas sobre el XML generado	10
Total		**40**

Tabla 5. Pila del sprint 2

Revisión del Sprint

En la *Tabla 6* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	Tarea	Coste estimado (h)	Coste real (h)	Desviación (h)	Motivos
H1.2	1.2	30	40	10	Documento con estructura variada, campos concretos.
H2.1	2.1	10	0	-10	Desplazado al siguiente sprint
Total		**40**	**40**	**0**	

Tabla 6. Desviaciones de la pila del sprint 2

Para finalizar el programa que extrae los datos del Excel y general el XML equivalente encontré nuevos casos en el Excel que no coincidían con los demás por lo que se generaba un XML incorrecto al intentar tratar los datos de esos casos de la misma forma que el resto.

Concretamente había una de las familias bacterianas cuyos antibióticos tenían partes del nombre en cursiva y además tenían notas en forma de superíndice (*Figura 57*). Esto hacía que el programa no detectase la cursiva y almacenase el nombre cortándolo de forma incorrecta.

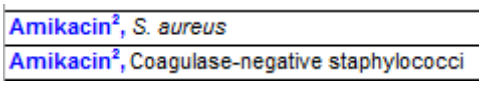


Figura 57. Ejemplo de antibiótico con cursiva y superíndice

Con este retraso decidí dejar la historia H2.1 para el siguiente sprint. Además tuve una reunión con el cliente y decidimos centrarnos en la correcta construcción del documento XML y la futura implementación de un servicio web básico antes que en la aplicación móvil. Por ello decidí reconstruir la pila del producto.

Sprint 3: XQuery e inicio del servicio web

El objetivo del sprint es aprender la forma de crear un servicio web en Java, crear su interfaz e intentar desplegarlo en un servidor local. Además, aprender XQuery para usarlo en el servicio como lenguaje de consulta sobre el documento XML.

Pila de Sprint

En este sprint desarrollo las historias H2.1, Consultas XQuery y H2.2, Servicio Java.

Historia	**Tarea**	**Coste (h)**
H2.1	Estudiar XQuery y probar consultas sobre el XML generado	10
H2.2	T2.2.1.1 Aprendizaje de servicios web en Java	15
	T2.2.1.2 Creación de interfaz y prueba de despliegue	15
Total		**40**

Tabla 7. Pila del sprint 3

Revisión del Sprint

En la *Tabla 8* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	**Tarea**	**Coste estimado (h)**	**Coste real (h)**	**Desviación (h)**	**Motivos**
H2.1	2.1	10	10	0	
H2.2	T2.2.1.1	15	15	0	
	T2.2.1.2	15	15	0	
Total		**40**	**40**	**0**	

Tabla 8. Desviaciones de la pila del sprint 3

Este sprint se desarrolló según lo previsto.

Comentar que encontré una web dónde poder probar las consultas XQuery de forma online que fue de gran utilidad para agilizar el proceso. La web es http://www.xpathtester.com/xquery

Sprint 4: Servicio web

El objetivo del sprint es terminar el desarrollo del servicio web en Java y desplegarlo en el servidor Centos suministrado por la universidad.

Pila de Sprint

En este sprint trabajo con las historias H2.2 Servicio Java y H2.3 Despliegue del servicio.

Historia	Tarea	Coste (h)
H2.2	T2.2.2.1 Estructura RESTful que permita peticiones GET y POST	10
	T2.2.2.2 Métodos de consulta de sensibilidad y de obtención de familias	20
H2.3	Despliegue del servicio en el servidor	10
Total		**40**

Tabla 9. Pila del sprint 4

Revisión del Sprint

En la *Tabla 10* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	Tarea	Coste estimado (h)	Coste real (h)	Desviación (h)	Motivos
H2.2	T2.2.2.2	10	10	0	
	T2.2.2.1	15	20	5	Uso de JsonObject y estructura de consultas XQuery compleja
H2.3	2.3	15	15	0	
Total		**40**	**45**	**5**	

Tabla 10. Desviaciones de la pila del sprint 4

Por defecto NetBeans crea una estructura básica para el servicio RESTful. Esta estructura está bien para servicios sencillos. Para el mío tuve que crear un fichero *web.xml*, añadir un par de elementos al *context.xml*, añadir un atributo *ServletContext* (cosa que me llevó mucho tiempo descubrir) para acceder de forma correcta a los recursos disponibles y crear un filtro para que todas las comunicaciones se hagan con la codificación UTF-8 ya que tomcat no lo hace por defecto. Además las primeras respuestas del servidor dieron problemas al convertirlas en JSON ya que había un pequeño problema de sintaxis al generarlos y fue complicado ver el punto exacto del error.

Sprint 5: Aplicación móvil – Conexión con el servidor

El objetivo del sprint es realizar una versión simple de la aplicación que se conecte con el servidor, envíe una petición bien formada y procese la respuesta.

Pila del Sprint

En este sprint trabajo con la historia H3.3 Resultados.

Historia	Tarea	Coste (h)
H3.3	T3.3.1 Conexión con el servicio web	20
	T3.3.2 Procesamiento de los datos	20
Total		**40**

Tabla 11. Pila del sprint 5

Revisión del Sprint

En la *Tabla 12* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	Tarea	Coste estimado (h)	Coste real (h)	Desviación (h)	Motivos
H3.3	T3.3.1	20	22	2	Peticiones POST en Volley con poca documentación
	T3.3.2	20	24	4	Problemas con el entorno
Total		**40**	**46**	**6**	

Tabla 12. Desviaciones de la pila del sprint 5

Android Studio va actualizándose regularmente. Desde que lanzó la versión 2.0 se han encontrado diversos problemas de compatibilidad. Uno de ellos me ha consumido unas cuantas horas. El problema es que, al intentar usar librerías externas a Android, tiene problemas al intentar usar Java 7 en lugar de Java 8 y saltan errores (principalmente por diferencias en el procesamiento de la aritmética de los números de doble precisión). La solución es la mostrada en la *Figura 58*: Forzar a construir el proyecto en la versión "24rc4" en lugar de la "23.0.3", activar las opciones Jack y forzar la compilación con java 8.

```
buildToolsVersion "24rc4"

defaultConfig {
    jackOptions {
        enabled true
    }
}
dexOptions {
    incremental true
}

compileOptions {
    sourceCompatibility = org.gradle.api.JavaVersion.VERSION_1_8
    targetCompatibility = org.gradle.api.JavaVersion.VERSION_1_8
}
```

Figura 58. Opciones del Gradle de Android para solucionar problemas con Java

Sprint 6: Aplicación móvil – Gestión de consultas e interfaz

El objetivo del sprint es realizar una versión simple de la aplicación que se conecte con el servidor, envíe una petición bien formada y procese la respuesta.

Pila del Sprint

En este sprint trabajo con las historias H3.2 Gestión de consultas y H3.1 Diseño de interfaz.

Este sprint es de tres semanas (60h).

Historia	Tarea	Coste (h)
H3.2	T3.2.1 Almacenamiento de las consultas	20
	T3.2.2 Obtención de las consultas almacenadas	10
H3.1	T3.1.1 Diseño de ventana de realización de consultas (*Search*)	5
	T3.1.2 Diseño de ventana de listado de consultas guardadas (*Queries*)	10
	T3.1.3 Diseño de ventana de detalles de consulta (*Details*)	10
	T3.1.4 Diseño de ventana de opciones (*Options*)	5
Total		**60**

Tabla 13. Pila del sprint 5

Revisión del Sprint

En la *Tabla 14* se recogen los tiempos reales empleados en cada una de las historias de usuario:

Historia	Tarea	Coste estimado (h)	Coste real (h)	Desviación (h)	Motivos
H3.2	T3.2.1	20	18	-2	
	T3.2.2	10	16	6	Problemas con el adaptador que gestiona la vista de las consultas almacenadas
H3.1	T3.1.1	5	5	0	
	T3.1.2	10	12	2	Problemas con el scroll y el toolbar
	T3.1.3	10	13	3	Scroll y estructura de CardView
	T3.1.4	5	2	-3	Casi automática
Total		**60**	**66**	**6**	

Tabla 14. Desviaciones de la pila del sprint 5

El motivo principal de la desviación temporal en este spring fue un problema recurrente con el scroll (desplazamiento) de las vistas y sus conflictos con la *Toolbar*.

El elemento *Toolbar* es un elemento implementado en las últimas versiones de Android por lo que si se desea utilizar en versiones anteriores hay que usar librerías de compatibilidad. Por suerte, Android viene preparado para ello y proporciona herramientas para hacerlo de forma casi transparente.

El problema es que en la mayoría de documentación y ejemplos que consulté inicialmente mostraban una forma de incorporar scroll en una vista y es envolviendo la misma en un elemento *ScrollView*. Después de encontrarme con muchos problemas de visualización (se cortaba bien la parte superior o bien la inferior, pero nunca se adaptaba completamente a la ventana), descubrí que este elemento entraba en conflicto con el elemento *Toolbar* en algunas versiones de Android.

El *Toolbar* corresponde a la barra superior de la aplicación que contiene el nombre de la vista actual junto con diferentes opciones (*Figura 59*).

Figura 59. Toolbar de la aplicación en diferentes vistas

Para aumentar la visualización de los elementos de las vistas quise hacer que el *Toolbar* desapareciera al hacer scroll (*Figura 60*).

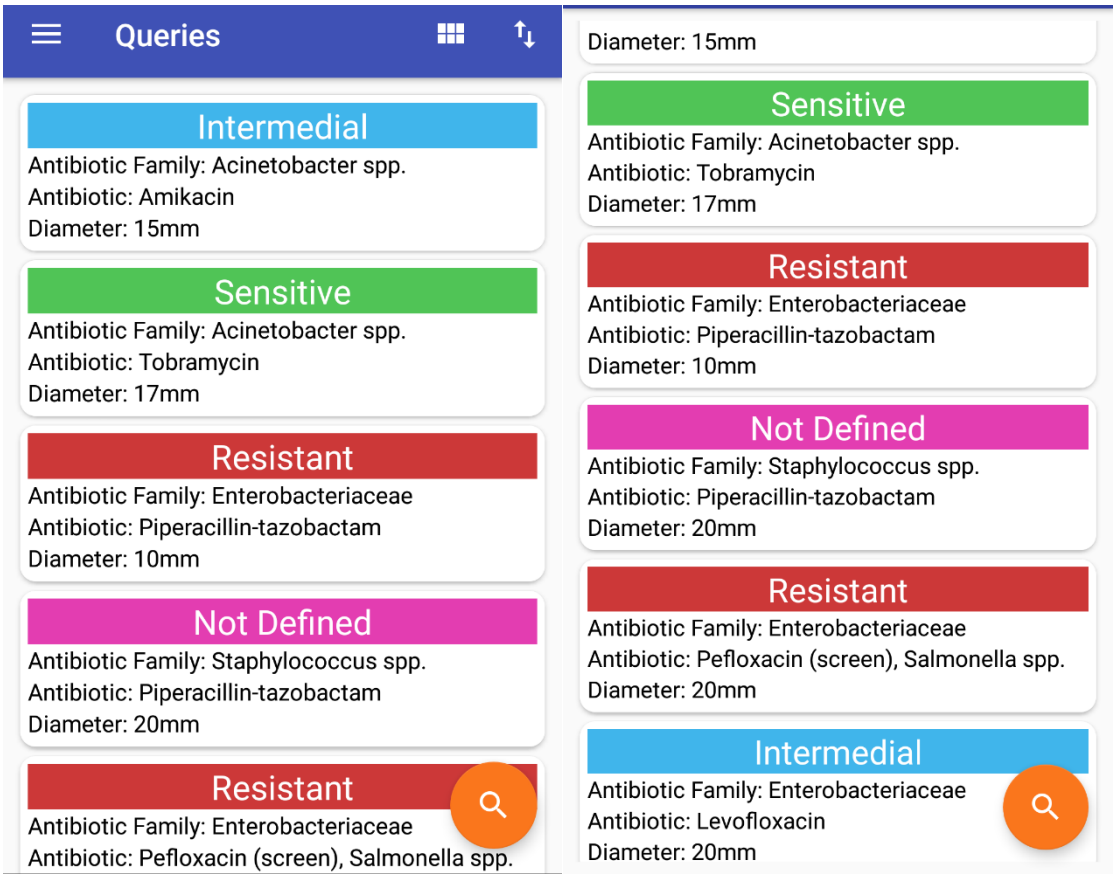


Figura 60. Toolbar de la aplicación en diferentes vistas

Al intentar implementarlo la vista detectaba que no había *Toolbar* y se cortaba en la parte superior. Al lograr que se fijara en la parte superior se cortaba en la parte inferior e incluso desaparecía el botón flotante.

La solución fue cambiar el elemento *ScrollView* por un *NestedScrollView* que está preparado para el comportamiento dinámico del *Toolbar*.

Finalmente, otra complicación fue que nuevamente el entorno Android Studio se quejó por errores de compatibilidad al compilar al sacar una nueva versión de la API ya testeada (v24). Me obligó a actualizar a dicha versión y tuve que modificar varios ficheros y librerías para poder ponerlo todo en orden de nuevo como se aprecia en la *Figura 59*.

```
compileSdkVersion 24
buildToolsVersion "24.0.0"
```

```
compile 'com.android.support:appcompat-v7:24.0.0'
compile 'com.android.support:design:24.0.0'
compile 'com.android.support:cardview-v7:24.0.0'
compile 'com.android.support:recyclerview-v7:24.0.0'
```

Figura 61. Nueva configuración del Gradle

Desviaciones finales

Finalmente finalicé el trabajo en 328 horas. 28 horas más de las previstas y eliminando contenido de la pila inicial. La *Tabla 15* contiene las desviaciones temporales finales.

Tema	**ID**	**Nombre**	**Coste pila inicial (h)**		**Coste pila final (h)**		**Coste real(h)**		**Desviación (h)**	
Sprint 0		Planificación e investigación	40		40		40		0	
1	H1.1	XML Schema	20	70	20	80	26	91	6	11
	H1.2	XML	50		60		65		5	
2	H2.1	Consultas XQUERY	10	60	10	80	10	85	0	5
	H2.2	Servicio en Java	40		60		65		5	
	H2.3	Despliegue del servicio	10		10		10		0	
3	H3.3	Resultados	30	130	40	100	46	112	6	12
	H3.1	Gestión de consultas	30		30		32		2	
	H3.2	Diseño de interfaz	30		30		34		4	
	H3.4	Base de datos	20		-		-		-	
	H3.5	Almacenamiento externo	20		-		-		-	
Total			**300**		**300**		**328**		**28**	

Tabla 15. Desviaciones finales del trabajo

Conclusiones

Al principio, el tema de este trabajo se presentó algo complicado de entender. Posteriormente me interesé más por él y ahora mismo agradezco haber podido desarrollar herramientas que puedan ayudar a agilizar este tipo de procedimientos.

Puedo decir que he disfrutado aprendiendo y desarrollando la parte de la aplicación móvil y viendo el resultado final. Las otras partes del trabajo se me hicieron algo más pesadas de desarrollar pero fuero igualmente enriquecedoras. Un informático siempre tendrá que aprender cosas nuevas y renovarse, por lo que ningún conocimiento sobra.

Otro factor con el que quise tratar al desarrollar este proyecto es el uso de metodologías ágiles. Pude comprobar que realmente son útiles para proyectos de desarrollo de software donde se sufren cambios constantemente. De haber realizado el proyecto usando una metodología tradicional no hubiera podido terminarlo y seguramente no dispondría de un producto final que entregar al cliente. Al usar metodología ágil pude entregar una versión de la aplicación muy avanzada. No con toda la funcionalidad planeada pero sí con la suficiente para que el cliente pudiera usar el servicio web y organizar sus consultas.

Ver que el cliente quedó contento con el resultado compensó más el esfuerzo que la finalización del proyecto en sí. Sentí que realmente pude crear algo útil.

Bibliografía

Web del comité europeo: http://www.eucast.org/

Antibiogramas: http://www.apcontinuada.com/es/el-antibioigrama-interpretacion-del-antibiograma/articulo/80000504/

http://www.microinmuno.qb.fcen.uba.ar/SeminarioAntibioticos.htm

API de Apache POI: http://poi.apache.org/apidocs/index.html

XQuery: http://www.w3schools.com/xsl/xquery_intro.asp

GSON: http://www.vogella.com/tutorials/JavaLibrary-Gson/article.html

Volley: https://android.googlesource.com/platform/frameworks/volley/

XStream: http://x-stream.github.io/

Android:

https://developer.android.com/develop/index.html?hl=es

http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/

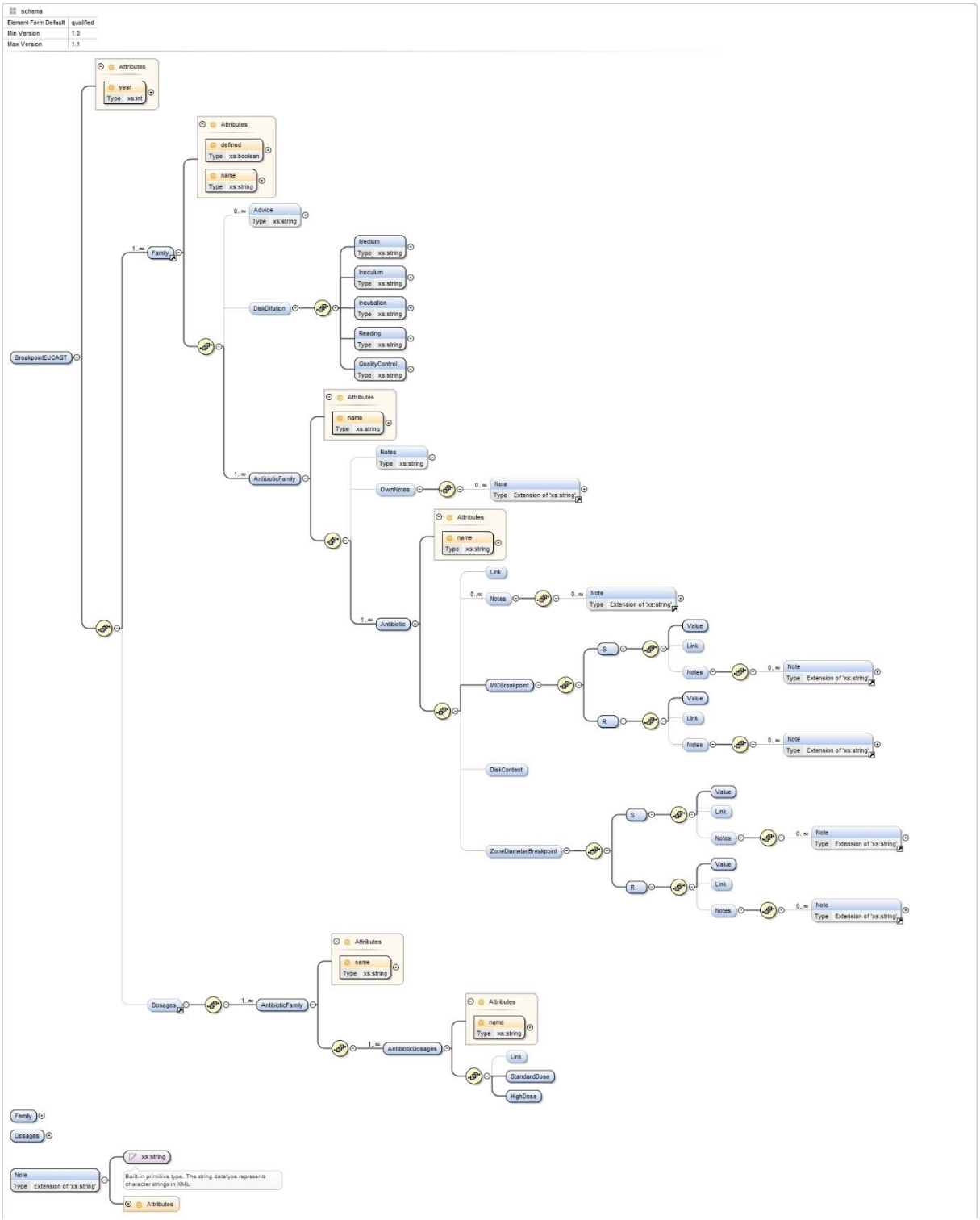
http://desarrollador-android.com/

http://www.materialpalette.com/

http://stackoverflow.com/

http://code.tutsplus.com/es/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465

Anexos



Anexo 1. Árbol del esquema XML correspondiente al documento EUCAST

Penicillins[1]	MIC breakpoint (mg/L)		Disk content (µg)	Zone diameter breakpoint (mm)	
	S ≤	R >		S ≥	R <
Benzylpenicillin	-	-		-	-
Ampicillin	8[1]	8	10	14[A,B]	14[B]
Ampicillin-sulbactam	8[1,2]	8[2]	10-10	14[A,B]	14[B]
Amoxicillin	8[1]	8	-	Note[C]	Note[C]
Amoxicillin-clavulanic acid	8[1,3]	8[3]	20-10	19[A,B]	19[B]
Amoxicillin-clavulanic acid (uncomplicated UTI only)	32[1,3]	32[3]	20-10	16[A,B]	16[B]
Piperacillin	8	16	30	20	17

Anexo 2. Ejemplo de las mediciones S y R

Stenotrophomonas maltophilia

Trimethoprim-sulfamethoxazole is the only agent for which EUCAST breakpoints are currently available. For further information, see guidance document on www.eucast.org.

Anexo 2. Ejemplo de *Advice* de una familia bacteriana

Notes

Numbered notes relate to general comments and/or MIC breakpoints.
Lettered notes relate to the disk diffusion method.

1. Trimethoprim:sulfamethoxazole in the ratio 1:19. Breakpoints are expressed as the trimethoprim concentration.
2. Breakpoints are based on high dose therapy, at least 240 mg trimethoprim and 1.2 g sulfamethoxazole administered together twice daily.

A. Ignore haze or fine growth within the inhibition zone (**see pictures below**).

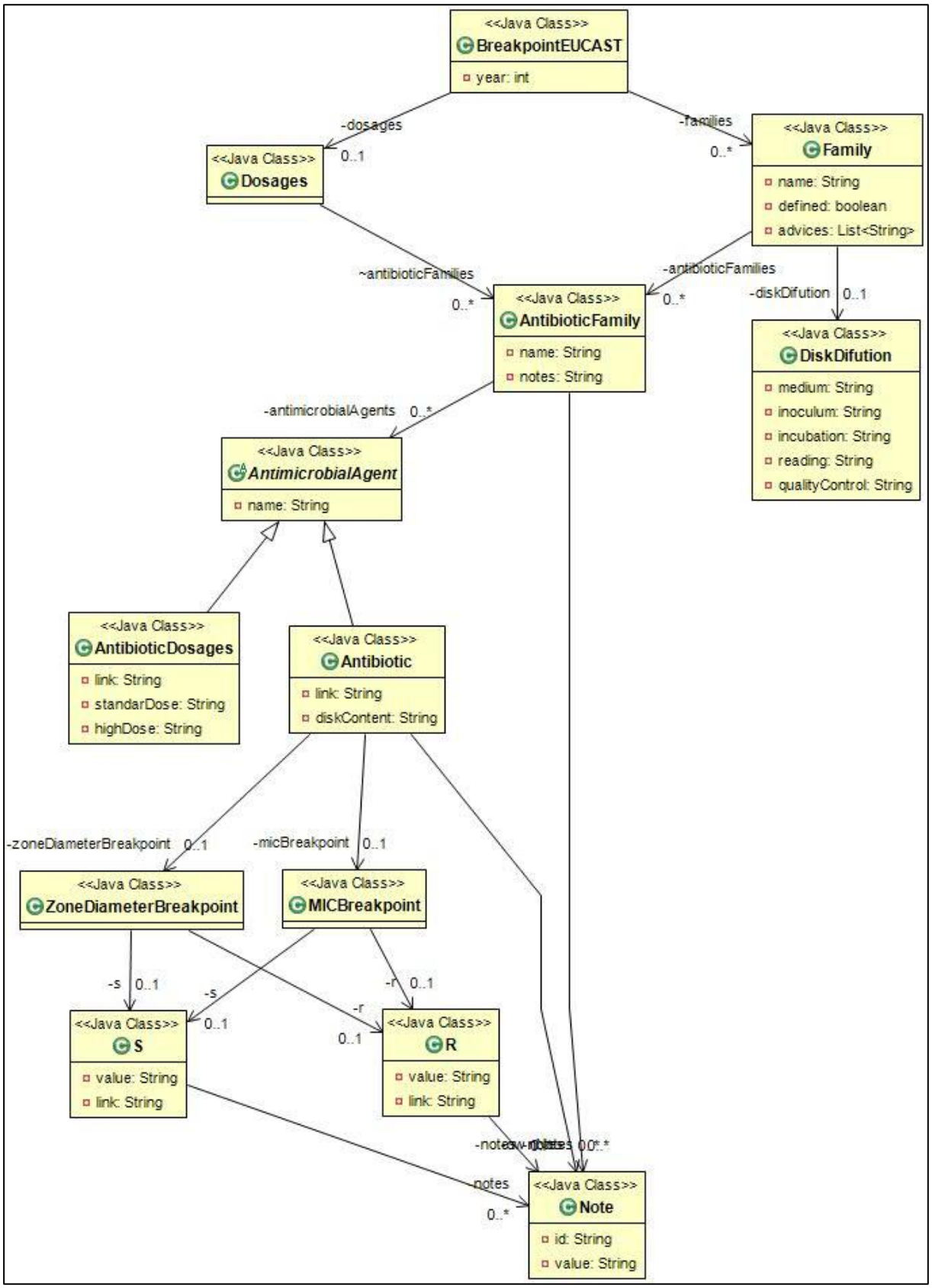
Anexo 3. *Notes* de una familia de antibióticos

```
C:\Juanmi\Dropbox\Unirioja\Cuarto\Cuatri 2\TFG\ExcelToXML>java -jar ExcelToXML.jar ?
Puedes introducir los siguientes argumentos:
-arg1: nombre del documento Excel
-arg2: nombre que tendrá el documento XML generado
-arg3: nombre del XML Schema contra el que validar el documento XML generado
```

Anexo 4. Ejemplos de ejecución del programa


```
{
    "antibioticFamily":{
        "name":"Penicillins",
        "ownNotes":[{"id":"1","value":"Wild type Enterobacteriaceae are categorised as susceptible to aminopen
    },"antibiotic":{
        "micBreakpoint":{"s":{"value":"8","notes":[{"id":"1","value":"Wild type Enterobacteriaceae are categor
        "name":"Ampicillin"
    },
    "sensibility":"Sensitive"
}
```


Anexo 5. Ejemplos respuesta completa en JSON



Anexo 6. ***Diagrama de clases equivalente al esquema XML***