



Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería Informática

Memoria del Trabajo de Fin de Grado

Memoria del Trabajo Fin de Grado
en Ingeniería Informática

Autor: Agustín Daniel Schöler Allub

Tutor: José Francisco Vélez Serrano

Agosto 2018

Índice general



1	Introducción	1
1.1	Motivación	1
1.2	Estado del arte	1
1.3	Objetivos	4
1.4	Estructura de la memoria	4
2	Análisis	7
2.1	Documento de especificación de requisitos.	7
2.1.1	Requisitos funcionales	7
2.1.2	Requisitos no funcionales	11
2.2	Diagrama de casos de uso	13
2.3	Recursos utilizados	14
2.3.1	Servidor Virtual Privado (VPS)	14
2.3.2	Dominio y certificado SSL	15
2.3.3	Otros recursos	15
3	Diseño e implementación	17
3.1	Herramientas utilizadas	17
3.1.1	Hardware	17
3.1.2	Software	17
3.2	Arquitectura del software	20
3.2.1	Spring API REST	20
3.2.2	Python API REST	29
4	Métricas	35
4.1	Tiempo empleado en el desarrollo del proyecto	35
4.2	Métricas relativas a la implementación	38
5	Conclusiones	43
5.1	Objetivos cumplidos	43
5.2	Futuros trabajos	44

Índice de figuras

2.2.1 Diagrama de casos de uso	16
3.2.1 Estructura del proyecto	21
3.2.2 Diagrama API REST en Spring	22
3.2.3 Diagrama Entidad-Relación	23
3.2.4 Swagger UI	26
3.2.5 Flujo de datos de OAuth2.0 con token JWT	28
3.2.6 Diagrama de actividad de la subida de fotos	31
3.2.7 Tabla de adaptadores para Bottle	33
4.1.1 Diagrama de Gantt sobre las fases del desarrollo.	39
4.2.1 Primer análisis de SonarQube sobre la aplicación	40
4.2.2 Segundo análisis de SonarQube sobre la aplicación	41
4.2.3 Tercer análisis de SonarQube sobre la aplicación.	42
5.2.1 Microservicios en Spring	45

Agradecimientos

Quiero agradecer sus consejos a Pablo Viniegra Picazo y a Vanessa Krebs Carretero.

Quiero agradecer a Jorge Aranda García y a Patricia de Gregorio Ruiz el tiempo que han compartido conmigo estos 4 años.

Quiero agradecer sus contribuciones a todos los desarrolladores de software libre que me han proporcionado herramientas para realizar este trabajo.

Finalmente, quiero agradecer su paciencia a mi familia durante el transcurso de la carrera.

Resumen

[Redacted text block]



Índice general

Capítulo 1

Introducción



En un principio, la idea principal era el desarrollo de una aplicación web, Por lo tanto, opté por la realización de servicios necesarios para el lado del cliente. Se decidió agregar un grado más de dificultad a dicha aplicación: El uso de la visión artificial. Dado que la aplicación web en ultima instancia es una red social, se decidió usar técnicas de Visión Artificial con el fin de mejorarla.

A continuación se detallaran tanto aspectos de desarrollo y análisis como aspectos de diseño, experimentación y métricas.

1.1. Motivación



En particular, siempre **quise** verme inmerso en el problema de desarrollar una aplicación web en su totalidad. Normalmente en las diferentes asignaturas de la carrera, se proponían tareas relacionadas con ello, pero nunca **sentí** que se profundizara mucho en el tema.

Como consecuencia, la búsqueda de documentación de las herramientas utilizadas y el ser autodidacta, han sido pilares importantes en la realización del TFG. Aunque cabe destacar que ha sido de los mejores proyectos en los que me he visto involucrado, sobre todo por la libertad que me proporcionó mi tutor para poder desarrollar dicho proyecto.

Por otra parte, aparte de un fin personal, se quería presentar dicha aplicación web con alguna mejora. Coincidiendo con la asignatura de Visión Artificial que hemos tenido este ultimo año, decidimos utilizar dicho campo para presentar dicha mejora. Con dicha mejora, se pretende realizar sugerencias a los usuarios sobre sus amistades cuando proceden a realizar una publicación en la que se incluye una foto.



1.2. Estado del arte

Por lo dicho anteriormente se pueden intuir las soluciones que pueden llegar a cubrir el problema que se desea abordar en este Trabajo de Fin de Grado.

Como **he** dicho en la sección anterior, **asumi** una parte concreta del desarrollo: **Administración del sistema** y la lógica que se encarga de que la pagina web esté en perfecto



funcionamiento. Una parte importante en el desarrollo de una aplicación web (o en una parte de ella) es escoger correctamente las tecnologías utilizadas.

En mi caso opté por herramientas con las que tenía soltura y además que estuviesen a la orden del día. Es importante esto último, dado que es importante que las herramientas cuenten con una documentación consistente y se encuentren actualizadas.

Lógica de la aplicación web

Actualmente existen multitud de herramientas informáticas que se encarguen del lado servidor en una aplicación web. Con casi total probabilidad la más utilizada sea el lenguaje de programación PHP, muchas veces viene preinstalado en la mayoría de sistemas y el lenguaje se parece bastante a otros bastante famosos. Además, PHP no se suele utilizar en su **versión regular**, se suele utilizar algún framework de dicho lenguaje. En mi caso **conocía** CakePHP, proporciona una arquitectura MVC, es de código abierto y además había trabajado con ello durante el tiempo suficiente como para tener cierta soltura.

Es necesario nombrar a Java, lenguaje fuertemente tipado y orientado a objetos. Ahora bien, si queremos hablar de Java y de lado servidor o lógica de la aplicación web hay que hablar de Spring Framework. Es el framework más antiguo en este campo, pero sigue siendo a día de hoy el mas popular entre los desarrolladores web. Consta de módulos que facilitan el trabajo a los desarrolladores. Antes de hablar a fondo sobre los módulos de Spring framework, es necesario hablar de Spring en si. Spring Framework sigue tres pasos a la hora del desarrollo:


1. Creación de un proyecto Maven/Gradle
2. Desarrollo de la aplicación web
3. Despliegue de la aplicación web en un servidor

Maven y Gradle son asistentes que facilitan la creación de proyectos Java y proporcionan herramientas para la gestión de dependencias. Existen diferencias consustanciales entre Maven y Gradle, como se puede ver en este artículo [11]. Pero, a fin de cuentas, las dos herramientas tienen la misma finalidad. Posteriormente se procede al desarrollo de la aplicación web y por ultimo para que la aplicación esté al servicio del cliente, es necesario el despliegue en un servidor.


Spring Framework, da la posibilidad de la creación de controladores utilizando las anotaciones, entre otras funcionalidad. Las anotaciones son directrices para controlar el comportamiento de Spring. Dichas anotaciones comienzan por el símbolo '@' seguido de la directiva que se dese ejecutar. Spring Framework [18] también es capaz de transformar los objetos en un formato de texto concreto, como JSON, mediante serializadores como Jackson con el fin de poder comunicarse debidamente con el lado del cliente. También proporciona anotaciones que ayudan a la comunicación con la Base de Datos, con un par de anotaciones en las clases es capaz de indicarle a la Base de Datos como deben de guardarse los datos y de que forma. Independientemente del tipo de Base de Datos (relacional o no relacional).

Además, existen diferentes tipos de anotaciones dependiendo de la necesidad del desarrollador: a nivel de método, a nivel de clase o a nivel de atributo.

En otro orden de cosas, tenemos los módulos de Spring Framework. Tenemos tres a destacar:

- Spring Boot [16]: Antes se ha hablado de que existen tres pasos a seguir para el desarrollo del lado servidor. El modulo Spring Boot de Spring Framework se encarga de simplificar esos pasos. Es decir, a fin de cuentas los pasos 1 y 3 no requieren del desarrollador para ser ejecutadas. Spring Boot se encarga de automatizar los pasos 1 y 3 para centrar todo el esfuerzo en la realización del código de la aplicación. Se puede ver mas detalladamente en este Blog [1]
- Spring MVC [19]: Este modulo hace referencia al Modelo-Vista-Controlador. Es el modulo más antiguo de Spring, fue de los primeros en incluirse. Y se puede intuir que fue el modulo que introdujo la arquitectura MVC. 
- Spring Data [17]: Posiblemente el modulo **más famoso** de Spring. Proporciona una capa de abstracción a Spring MVC, simplificando la conexión con la Base de Datos y facilitando las funciones de persistencia, manteniendo las funciones CRUD propias de estas gracias a los DAOs (Data Access Object), entre otras funciones.

Spring, en combinación con los módulos descritos, da la capacidad de la definición sencilla de Beans que luego serán traducidos a un fichero XML de configuración de la aplicación. Un Bean en Spring es un objeto que ha sido configurado e instanciado en el contenedor de Spring. Dichos beans, permanecerán en la aplicación web hasta que el propio desarrollador los destruya. El propio desarrollador define beans mediante anotaciones que luego se traduce en ficheros XML que Spring interpreta e integra en la aplicación web.

Por otra parte tenemos Python, no va a ser la primera vez que se hable de Python en este documento ni la última. Lenguaje de programación bastante sencillo, orientado a objetos y débilmente tipado. Normalmente viene instalado de serie en la mayoría de sistemas Linux. De nuevo: el potencial de Python a la hora del desarrollo web reside en los framework. En este caso destaco tres: Django, Flask y Bottle. 

Django es posiblemente el framework de referencia si se decide utilizar Python para el desarrollo de la lógica de la aplicación web. Proporciona arquitectura MVC, es de código abierto y proporciona herramientas que ayudan a la autenticación en la aplicación web. Por otro lado, tenemos Flask, sigue el mismo camino que Django, se trata de un framework de Python basado en Werkzeug y Jinja2. Werkzeug es una librería de utilidades para WSGI de Python y Jinja2 es un motor de sistemas de plantillas inspirado en Django. Por ultimo, tenemos Bottle. Bottle es bastante parecido a Flask y es el enfoque que yo he utilizado para el desarrollo de parte de la aplicación. Es un microframework (al igual que Flask) basado en WSGI que sigue también la arquitectura MVC. En anteriores secciones he mencionado que tenia como objetivo incluir alguna mejora para la lógica de la aplicación web. Desde aquí se recomienda cualquiera de los tres frameworks de Python para el desarrollo de la aplicación web es válido y no supone ningún obstáculo a la hora del desarrollo web.



Por ultimo, **esta sección no podría acabar si no hablásemos de JavaScript**, lenguaje sencillo bastante parecido a Java, débilmente tipado y orientado a objetos. Normalmente se conoce a JavaScript por su utilidad en el lado cliente de las aplicaciones, pero en este caso vamos a hablar de él en el lado del servidor. Si hablamos de JavaScript en el lado del servidor, entonces tenemos que hablar de NodeJS. NodeJS es un entorno en tiempo de ejecución en JavaScript en el lado del servidor. Este entorno por si solo es posible que

tenga menos utilidad que los frameworks que se han nombrado anteriormente, pero es una gran base para un framework web. La lógica a la hora de desarrollar el lado servidor no es diferente al que se usa en las herramientas anteriores. La diferencia reside en la facilidad que proporciona JavaScript a la hora del desarrollo. NodeJS tiene, indirectamente, la facilidad que proporciona JavaScript a la hora de programar. No contiene la dificultad que presenta Java a la hora del tipado. Por ejemplo, mientras que en Java se tiene que modificar el tipo de lo que devuelve una función porque el cliente necesita otra lógica en la respuesta, en NodeJS bastaría con agregarlo lo que se necesita en el objeto a devolver.

Ahora bien, desde aquí se recomienda el uso de Spring Framework para el desarrollo del lado del servidor, juntos con sus módulos. Posee una documentación consistente y constantemente actualizada, dada su popularidad entre los desarrolladores del lado servidores. Hace de desarrollar una aplicación, una tarea bastante cómodo y sencilla. Además, Spring Framework se suele utilizar en combinación de algún entorno de programación. (IntelliJ, Spring Tool Suite, Netbeans, Eclipse, etc.)

1.3. Objetivos

el desarrollo de servicio que ofrezca las funciones necesarias para c

Este trabajo tiene como objetivo [REDACTED]. Dicho objetivo principal puede desglosarse en diferentes objetivos específicos:

- Proporcionar al lado cliente una serie de servicios que le provea los recursos que necesita para: gestionar los usuarios, gestionar las entradas en la red social, ges
- Proporcionar al lado cliente una documentación consistente acerca de la aplicación
- [REDACTED]
- Entender en profundidad lo que significa desarrollar un software desde cero
- Tener un entorno en local para el desarrollo y un entorno en producción para probar lo desarrollado [REDACTED]
- Búsqueda y uso de herramientas que hagan posible la realización de los objetivos nombrados hasta ahora.

. En particular: un manual de instalación y ur

1.4. Estructura de la memoria

El resto de la memoria de este TFG se estructura

[REDACTED] de la siguiente manera:

- [REDACTED]
- **Capítulo 2:** Análisis. En este capítulo se realiza una descripción completa del sistema que se desea construir.

- **Capítulo 3:** Diseño e implementación. En este capítulo se describe la solución que se creará para conseguir los objetivos y los requisitos que se han planteado previamente.
- **Capítulo 4:** Capítulo de experimentos, pruebas y métricas. En este capítulo se explican las métricas que se han acumulado durante el desarrollo del proyecto
- **Capítulo 5.** Conclusiones. En este capítulo se explican cómo se han cumplido los objetivos que se detallaron al principio de la memoria.

Capítulo 2

Análisis

A continuación se va a proceder a realizar un análisis detallado sobre el proyecto. En primer lugar, se realiza un listado (con su respectiva explicación) de requisitos tanto funcionales como no funcionales de la solución que se pretende desarrollar. Posteriormente, se presenta un diagrama de casos de uso, presentando un modelo del problema que se desea resolver. Finalmente se hace un análisis de los recursos utilizados desde diferentes puntos de vista.

aquí se especifica consta de un API y de la implementación correspondiente.

2.1. Documento de especificación de requisitos.

Antes de empezar a enumerar los requisitos y a presentar una serie de aclaraciones con respecto a ellos, es necesario aclarar varios conceptos. Como introducción a los requisitos cabe destacar que el sistema que [REDACTED]. Concretamente se trata de una API REST. Una API REST es una biblioteca apoyada totalmente en el estándar HTTP.

Dicha API REST se conecta a una Base de Datos MySQL para asegurar la persistencia de los datos. El desarrollo de la API se ha llevado a cabo a través de la herramienta Spring Tool Suite, utilizando Spring Boot y Spring Data, de lo cual se ha hablado en capítulos anteriores. Se tiene un entorno de desarrollo (en local) y otro entorno de producción (en un Servidor Virtual Privado) para que el cliente pueda hacer uso de ella, todo esto se detallará más en capítulos posteriores.

2.1.1. Requisitos funcionales

He creído conveniente separar los requisitos por secciones, al igual que la API separa los servicios que ofrece por controladores.

Requisitos relacionados con los usuarios

- RF01.U: Obtener un usuario.

- Requisito que hace referencia a obtener un determinado usuario

- **RF02.U: Obtener todos los usuarios.**

- Requisito que hace referencia a obtener todos los usuarios que hay almacenados en la Base de Datos

- **RF03.U: Añadir un usuario**

- Requisito que hace referencia a añadir un determinado usuario a la Base de Datos

- **RF04.U: Editar un usuario**

- Requisito que hace referencia a editar un determinado usuario de la Base de Datos

- **RF05.U: Eliminar un usuario**

- Requisito que hacer referencia a eliminar un determinado usuario de la Base de Datos

- **RF06.U: Eliminar todos los usuarios**

- Requisito que hacer referencia a eliminar todos los usuarios de la Base de Datos

Cabe destacar que el requisito RF02.U y RF06.U están sujetos a restricciones dado que son agujeros de seguridad. Solo podrán hacer uso de dichas funcionalidades los administradores.

Requisitos relacionados con las publicaciones

- **RF01.P: Obtener una publicación**

- Requisito que hace referencia a obtener una determinada publicación de la Base de Datos

- **RF02.P: Obtener todas las publicaciones**

- Requisito que hacer referencia a obtener todas las publicaciones almacenadas en la Base de Datos

- **RF03.P: Añadir una publicación**

- Requisito que hace referencia a añadir una publicación a la Base de Datos

- **RF04.P: Editar una publicación**

- Requisito que hace referencia a editar una determinada publicación

- **RF05.P: Eliminar un publicación**

- Requisito que hace referencia a eliminar una determinada publicación de la Base de Datos

- **RF06.P: Eliminar todas las publicaciones**

- Requisito que hace referencia a eliminar todas las publicaciones de la Base de Datos

Los requisitos RF02.U y RF06.U están sujetos a restricciones, dado que son agujeros de seguridad. Solo podrán hacer uso de dichas funcionalidades los administradores

Requisitos relacionados con los comentarios

■ RF01.C: Obtener un determinado comentario

- Requisito que hace referencia a obtener un determinado comentario de la Base de Datos

■ RF02.C: Obtener todos los comentarios

- Requisito que hacer referencia a obtener todos los comentarios almacenadas en la Base de Datos

■ RF03.C: Añadir un comentario

- Requisito que hace referencia a añadir un comentario a la Base de Datos

■ RF04.C: Editar un comentario

- Requisito que hace referencia a editar un determinado comentario

■ RF05.C: Eliminar un comentario

- Requisito que hace referencia a eliminar un determinado

■ RF06.C: Eliminar todos los comentarios

- Requisito que hace referencia a eliminar todos los comentarios de la Base de Datos

Los requisitos RF02.C y RF06.C están sujetos a restricciones, dado que son agujeros de seguridad. Solo podrán hacer uso de dichas funcionalidades los administradores

Requisitos relacionados con la búsqueda de usuarios dado un determinado criterio.

■ RF01.S: Buscar a un usuario dado su nombre completo

- Requisito que hace referencia a buscar al usuario a través de la Base de Datos, teniendo en cuenta que el usuario nos proporcionan memora el nombre de dicho usuario

■ RF02.S: Buscar a un usuario dado su nombre de usuario (nick o alias):

- Requisito que hace referencia a buscar al usuario a través de la Base de Datos, teniendo en cuenta que el usuario nos proporciona el nick o alias de dicho usuario



Requisitos relacionados con el perfil de usuario.

- **RF01.PR: Obtener el perfil de un determinado usuario** Pero para poder desplegar la API REST
 - Requisito que hace referencia a obtener el perfil de un determinado usuario
- **RF02.PR: Editar el perfil de un determinado usuario**
 - Requisito que hace referencia a editar el perfil de un determinado usuario

Requisitos relacionados con la descarga de archivos

Como hemos especificado anteriormente, la API REST está especialmente diseñada para dar sus servicios a una plataforma con algún tipo de estructura social. Por tanto, es posible la descarga de determinados archivos.

- **RF01.D: Descarga de imágenes**
 - Requisito que hace referencia a descargar una determinada imagen de la Base de Datos. Solo podrán hacer uso de esta funcionalidad los administradores.

Requisitos relacionados con las relaciones de amistad.

- **RF01.F: Obtener un determinado amigo**
 - Requisito que hace referencia a obtener un determinado amigo de la Base de Datos
- **RF02.F: Agregar un determinado amigo**
 - Requisito que hace referencia a agregar a un amigo a la lista de amigos.
- **RF03.F: Eliminar un determinado amigo**
 - Requisito que hace referencia a eliminar a un amigo de la lista de amigos.
- **RF04.F: Obtener las publicaciones de tus amigos**
 - Requisito que hace referencia a obtener las publicaciones de todos tus amigos (ordenadas por fecha de subida)
- **RF05.F: Obtener todos tus amigos**
 - Requisito que hace referencia a obtener todos los amigos de la lista de amigos
- **RF06.F: Obtener una sugerencia.**
 - Requisito que hace referencia a obtener una sugerencia teniendo en cuenta los amigos con los que has tenido contacto hasta el momento.

Requisitos relacionados con el inicio de sesión y el registro de usuarios.

- **RF01.LR: Iniciar sesión**

- Requisito que hace referencia a, dado un nombre de usuario y una contraseña, iniciar sesión.



- **RF02.LR: Cerrar sesión**

- Requisito que hace referencia a poder cerrar la sesión de un determinado usuario.

- **RF03.LR: Registrar a un determinado usuario**

- Requisito que hace referencia a, dados una serie de datos que se requieren por parte del usuario, registrar a un determinado usuario dentro del sistema, Con el objetivo de poder iniciar sesión.

- **RF04.LR: Obtener al usuario que ha iniciado sesión**

- Requisito que hace referencia a obtener el usuario que ha iniciado sesión en ese momento.

Requisitos relacionados con el reconocimiento facial

- **RF01.VA: Obtener caras de una determinada foto**

- Requisito que hace referencia a obtener las caras de las amistades de un determinado usuario en una foto

- **RF02.VA: Obtener sugerencias sobre las amistades de un determinado usuario**

- Requisito que hace referencia a, dadas una serie de caras relacionadas con usuarios, obtener sugerencias para los usuarios.

- **RF03.VA: Enlazar caras con usuarios**


- Requisito que hace referencia a la posibilidad de enlazar caras en una foto con un usuario.

2.1.2. Requisitos no funcionales

- **RNF01: Conexión estable a Internet**

- Dado que la API REST es desplegada en un servidor virtual privado, es necesario que el usuario que pretenda hacer uso de dicho servicio esté conectado a Internet.

- **RNF02: Tiempo de respuesta razonable**

- Requisito no funcional [REDACTED] imprescindible en un servicio de este tipo. [REDACTED] Este requisito puede depender de otros factores, como el tipo de conexión que use el cliente a la hora de usar la API REST. Pero también es responsabilidad del desarrollador hacer que estos tiempos se reduzcan lo máximo posible. Mediante búsquedas eficientes, uso de estructuras de datos adecuadas, alojamiento con buenas prestaciones, procurando reducir la complejidad del código lo máximo posible, etc.
- **RNF03: Conexión estable a la Base de Datos**
 - En este caso, la API REST se conecta directamente a la Base de Datos para poder almacenar los datos. Una mala conexión a la Base de Datos implicaría directamente la inutilización del servicio REST.
- **RNF04: Disponibilidad de un servidor remoto donde poder desplegar la API**
 - El desarrollo del servicio REST se realiza en local, pero es necesario tener un alojamiento remoto donde poder desplegar la API. Es necesario que el cliente pueda utilizar el servicio esté donde esté. Como ya he dicho antes, en mi caso contraté un alojamiento de tipo VPS (Servidor Virtual Privado). Se decidió que el Sistema Operativo fuese un Debian GNU/Linux versión 9 (stretch), la cual es la versión más estable del SO. 
- **RNF05: Concordancia con respecto al estándar HTTP**
 - Este requisito está implícito en la definición de una API REST. [REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON 3]. Recibir peticiones HTTP, interpretarlas y en función de ello contestar con una respuesta HTTP con las cabeceras adecuadas es crucial en un servicio de este tipo. Se entra en detalle en requisitos posteriores.
- **RNF06: Asegurar la disponibilidad, integridad y confidencialidad de los datos**
 - Requisito que hace referencia a cumplir los tres pilares básicos de la Seguridad Informática: disponibilidad, integridad y confidencialidad de la información. En un servicio de este tipo, y además enfocado hacia una plataforma de carácter social, es importante de cara al cliente asegurarse de que la información no se ve comprometida. Ello se consigue cifrando la información, utilizando certificados SSL proporcionados por alguna entidad certificadora, no dar a los usuarios ordinarios el poder de entrar a cualquier punto del servicio, poseer administradores que controlen la actividad de los usuarios ordinarios, etc.
- **RNF07: Recibir peticiones HTTP**
 - Requisito que hace referencia a la capacidad de recibir peticiones que siguen el estándar HTTP. Dichas peticiones tienen que pasar por los controles de seguridad del servicio. Por ejemplo, a todas las peticiones HTTP les precede una petición de tipo especial que establece las condiciones que se deben de

cumplir para que el servicio puedan intercambiar información. La API REST debe de comprender dichas peticiones y actuar en consecuencia.

■ RNF08: Contestar peticiones HTTP

- Requisito que hace referencia a la capacidad de responder peticiones que siguen el estándar HTTP. Siguiendo con lo dicho en el requisito **RNF07**, el servicio debe de ser capaz contestar esas peticiones y dejarle claras las condiciones al cliente para así poder empezar a intercambiar información.

■ RNF09: Mantenibilidad

- Requisito que hace referencia a mantener la mantenibilidad del código. En este caso, la mantenibilidad del código está asegurada, como se ha dicho anteriormente, se usa para el desarrollo de la aplicación el entorno de programación derivado de Eclipse: Spring Tool Suite. Hace capaz el despliegue del servicio en un entorno local, donde modificar el código sin riesgo. Por otra parte, he asegurado la mantenibilidad separando el servicio internamente por controladores, los cuales sirven a diferentes tipos de peticiones. Pasa lo mismo con los archivos encargados de la configuración del servicio, acceso a la Base de Datos, encriptación de la información, etc.

■ RNF10: Documentación

- Requisito que hace referencia a proporcionar documentación consistente al cliente, para que pueda hacer uso del servicio. Gracias al uso de Spring Boot, fue simple construir una documentación consistente gracias a Swagger. Swagger analiza los paquetes existentes en tu proyecto y genera la documentación de la API REST. La documentación se genera en formato HTML y se puede llegar a ella fácilmente a través de cualquier navegador. Dicha documentación separa por secciones los diferentes controladores, con sus métodos, lo que necesita el método, etc. La documentación de Swagger incluso permite simular la petición pasando los datos pedidos.

■ RNF11: Escalabilidad

- Requisito que hace referencia a que el servicio sea capaz de escalar debidamente, por ejemplo para incorporar mas funcionalidades, ser capaz de almacenar mas información, atender peticiones desde diferentes orígenes, etc.

2.2. Diagrama de casos de uso

Una vez definido el documento de especificación de requisitos se procede a la realización del modelo del problema que se desea resolver, para ello se realiza un diagrama de casos de uso.

Solo existe un tipo de actor, que es el [REDACTED] cliente. Las acciones que realiza el servicio [REDACTED] la funcionalidad que el cliente puede usar. [REDACTED] los casos de uso derivados de los requisitos funcionales ya mencionados [REDACTED].

define

2.3. Recursos utilizados

Antes de finalizar el análisis del proyecto desarrollado, es necesario hablar sobre los recursos utilizados. Ya se detallo anteriormente las herramientas utilizadas en este proyecto. Pero han sido necesarios otros recursos importantes para poder llevar a cabo este proyecto.

2.3.1. Servidor Virtual Privado (VPS)

Antes de entrar en detalles, comencemos en una pequeña definición extraída del soporte de GoDaddy[8], un distinguido proveedor de alojamientos web y dominios:[Al ocupar el espacio entre los formatos de alojamiento dedicado y compartido, un servidor virtual privado (VPS, por sus siglas en inglés) ofrece muchas de las capacidades y funciones de los servidores dedicados, incluyendo acceso a administrador (raíz) y direcciones IP dedicadas, pero a un precio mucho más bajo 15]. Es decir, un VPS no es más que un servidor dedicado de menos coste compartido con otros usuarios. Así que se decidió que podría ser una buena elección. En mi caso, el servidor no se contrató en GoDaddy, si no a una empresa francesa llamada OVH, proveedora de alojamiento web y dominios[13]. Lo que quiero destacar de haber contratado el alojamiento de la aplicación en dicha empresa es que consta con una interfaz de gestión para el cliente impecable, con una gran usabilidad y con un diseño acorde al estilo de la pagina web de la empresa. Además, dicha interfaz hace posible poder reinstalar el servidor entero, reiniciarlo, cambiar de propietario, la posibilidad de mejorar el VPS, etc. Además, de información relativa al servidor como la localización, su IP, quien es el administrador, el tipo de SO, el estado del servidor, etc.

A la hora de contratar el servidor se le pide al cliente que Sistema Operativo utilizar, se selecciona y la empresa te proporciona las herramientas necesarias para empezar a utilizarlo. De hecho, este fue uno de los primeros pasos cuando se quiso abordar el problema especificado anteriormente, el contratar un alojamiento web.

Tomcat 8

Para que fuese posible el despliegue de la API REST desarrollada en Spring, era necesario un contenedor web donde poder desplegarla. Tomcat da la capacidad de desplegar aplicaciones web que vienen empaquetadas en formato WAR o JAR, dependiendo de lo que se quiera hacer. Tomcat provee un administrador donde poder ver las aplicaciones que se tienen desplegadas y su ruta de acceso. Por eso, mientras que en el local, durante el desarrollo en local de la aplicación se accede a la propia maquina (<http://localhost:8080>), en producción se accede a la API a través de la IP remota del VPS, apuntando al puerto correspondiente y haciendo referencia a la aplicación web desplegada (<http://IP:8080/APP>). Tomcat casi siempre va en combinación de Apache, con el objetivo de configurar el proxy, del cual hablaremos más adelante. La instalación de Tomcat 8 también fue uno de los primeros pasos a la hora de abordar el problema.

Python, Anaconda y librerías

Ya se ha hablado de que se tiene como objetivo incluir una mejora a la aplicación web que tenga relación con la Visión Virtual. Y: ¿qué es la Visión Artificial sin Python?

Cabe destacar que el VPS tenia Python instalado. Cambien hay que destacar que existen distribuciones de Python, especializadas en aprendizaje automático entre otros campos. Además con Anaconda ya no es necesario utilizar el comando pip que se utiliza con Python para la instalación de librerías, se utiliza el comando conda.

Por otra parte, para poder proceder con el desarrollo, fue necesario la instalación de librerías relacionadas con la conexión a la Base de Datos (mysql), con el framework que permite el desarrollar la API en Python (bottle), con la lectura de directorios y archivos del servidor (os), con el reconocimiento facial (face_recognition), con el deep learning (dlib), etc.

2.3.2. Dominio y certificado SSL

Acceder a la aplicación web a través de la IP no es seguro, así que se decidió la contratación de un dominio acompañado del cifrado SSL pertinente. Primero se procedió a registrar el nombre del dominio. Se realizó a través de la misma empresa donde se contrató el servidor. Después de la contratación, simplemente se creo un registro de tipo A apuntando a la IP del VPS para poder acceder directamente al dominio.

Por otra parte, por mucho dominio que se tuviese, era necesario que las peticiones y las respuesta estuviesen cifradas. Así que, a través de Let's Encrypt [10], empresa encargada de proveer certificados SSL gratuitos, se cifró el trafico de la API y el lado cliente.

Al igual que con el servidor privado, la interfaz de gestión que proporciona OVH, da la funcionalidad necesaria para poder administrar el dominio. La creación, edición y eliminación de registros relacionados con el dominio, gracias a esto se ha conseguido apuntar el dominio a la IP consultar los servidores DNS, realizar redirecciones, etc.

2.3.3. Otros recursos

Aparte de lo ya mencionado, han sido necesarios otros recursos para el correcto funcionamiento de la aplicación. Después de contratar el dominio y enlazarlo con la IP correspondiente, se presentó el problema de que era necesario utilizar el puerto 8080 para poder acceder a la API. Por tanto, fue necesario la instalación de Apache con el fin de configurar un proxy que recibiese todo el tráfico de la API y así poder eliminar el puerto del dominio.

Para realizar la conexión remota con el servidor y poder realizar labores administrativas en el mismo e instalar muchos de los recursos nombrados anteriormente fue necesario SSH. Mediante un programa llamado PuTTY **me** fue posible la conexión remota desde cualquier dispositivo. Ahora bien, en este caso concreto, la empresa a la cual se le contrato el servidor provee la funcionalidad dentro de su Interfaz de Gestión de poder conectarse al servidor con un simple clic, así que en cierta manera se podría prescindir de este recurso.



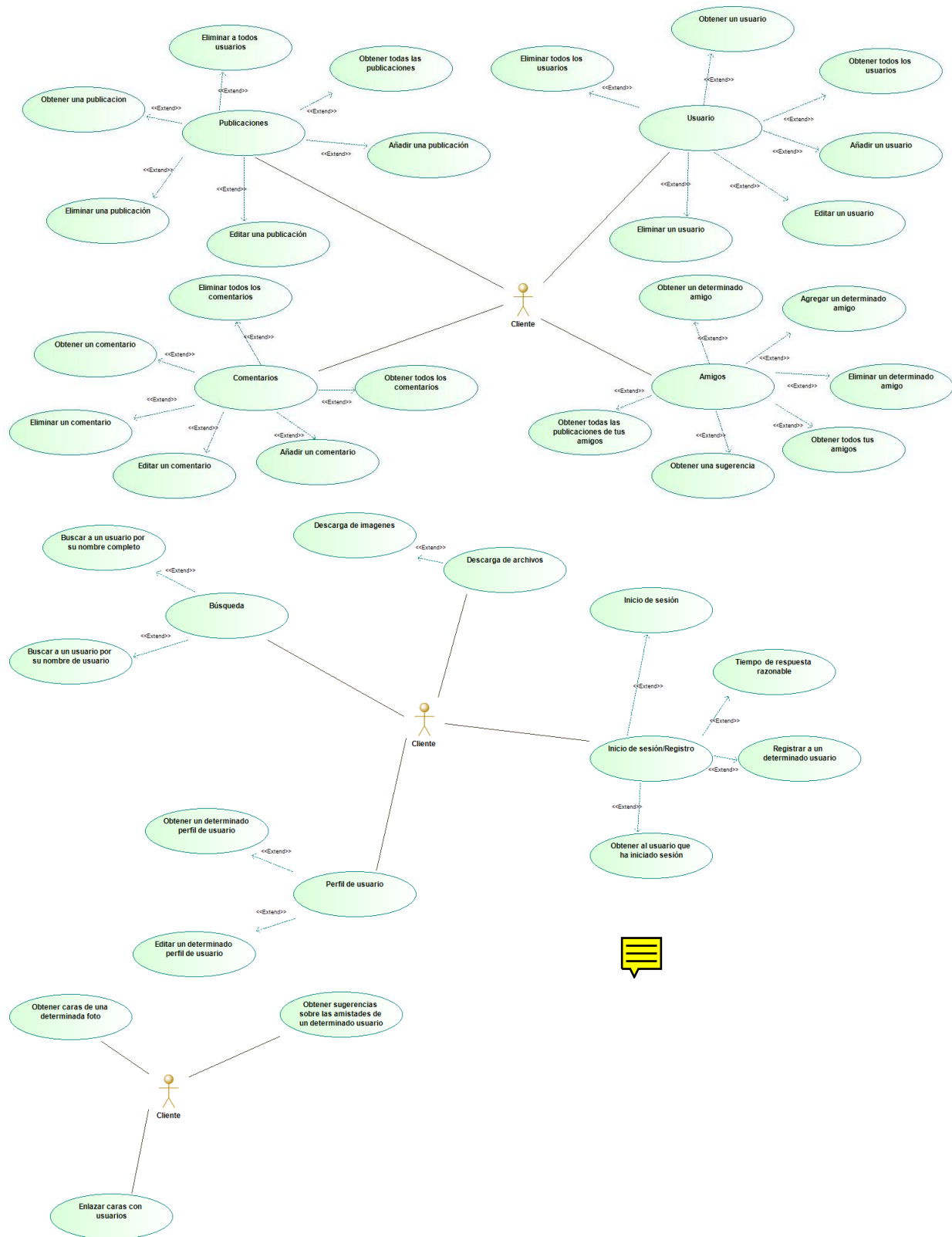


Figura 2.2.1: Diagrama de casos de uso

Capítulo 3

Diseño e implementación

Después del análisis del proyecto, se procede a explicar como se ha realizado el diseño e implementación de la solución al problema presentado al principio de este documento.

3.1. Herramientas utilizadas

En el capitulo de Análisis, concretamente en el capitulo de recursos utilizados, se entro un poco en detalle de las herramientas utilizadas. En esta sección se hablará más en detalle de ello.

3.1.1. Hardware

Dado que se trata del desarrollo de una aplicación software, las herramientas hardware utilizados han sido escasas en número. Entre las herramientas hardware utilizadas, se encuentran:

- Ordenador de sobremesa. Componentes a destacar:
 - **Procesador (CPU):** AMD Ryzen 3 1300X 3.7Ghz
 - **Tarjeta gráfica (GPU):** Asus GeForce GTX 1060 OC Dual 3GB GDDR5
 - **Memoria RAM:** Crucial DDR4 2133 PC4-17000 8GB CL15
 - **Disco Duro:** Seagate BarraCuda 3.5" 1TB SATA3
- Ordenador portátil ASUS. Componentes a destacar:
 - **Procesador (CPU):** Intel Core i5-6300HQ
 - **Tarjeta gráfica (GPU):** Nvidia GeForce GTX 950M
- Monitor Samsung 32".

3.1.2. Software

Al contrario que en la subsección anterior, aquí las herramientas software utilizadas son abundantes. A continuación se procede a detallar dichas herramientas:

GitHub y SmartGit

Antes de hablar de estas herramientas, creo necesario hablar de Git. Herramienta que ha supuesto un pilar muy importante para el desarrollo del proyecto. Git es una herramienta de control de versiones. Es especialmente recomendable cuando el mismo código es utilizado por mas de una persona, dado que tiene una herramienta para la solución de conflictos. En mi caso, es cierto que el proyecto a realizar tenia algún tipo de relación con una de mis compañeras, pero dado que nuestras partes en el desarrollo están perfectamente parceladas, prácticamente ninguno de los dos tocaba el trabajo del otro.

Por otra parte, en mi caso he necesitado Git para poder tener constancia de versiones estables de la aplicación, y sobre todo para tener un registro de lo que se ha haciendo y de los avances durante el tiempo que ha durado el desarrollo.

En otro orden de cosas, GitHub es un sitio web donde alojar proyectos que han sido desarrollados colaborativamente. En mi caso, usé GitHub para poder alojar el proyecto y poder hacer uso de él. Consta de una documentación impecable y proporciona una serie de facilidades que muchas otras no ofrecen. Después de subir el proyecto y por tanto crear un repositorio, GitHub te proporciona una URL pública con la que cualquiera puede acceder y revisar el código.

Por último, tenemos SmartGit. SmartGit es un cliente con interfaz gráfica para Git, y obviamente tiene la posibilidad de conectar con GitHub y los repositorios que se posean. Con esta herramienta, se fue capaz de tener un control gráfico del proyecto. Simplemente proporcionando la URL del repositorio en GitHub, es posible tener el control del mismo desde el cliente. Cabe destacar que dicho cliente no es un sitio web, si no una aplicación de escritorio, por lo que resulta más útil a la hora de hacer uso de ella.

Lenguaje de programación y framework.

El lenguaje de programación que se ha utilizado ha sido Java. Una de las razones principales fue que se decidió utilizar Spring, como framework de Java para el desarrollo web. Spring trae consigo una serie de módulos para hacer más sencillo aún el desarrollo de la aplicación web. En mi caso, he utilizado los módulos de Spring: Spring Boot, Spring Data y Spring MVC. Gracias a Spring y a sus módulos, fue posible no solamente el desarrollo de los controladores encargados de atender las peticiones HTTP desde el lado del cliente, si no ademas establecer la configuración de ciertas funcionalidades de la aplicación web.

A parte de los módulos enumerados anteriormente, se utilizó otro más Spring Security. Spring Security es el modulo de Spring encargado de la seguridad de la aplicación. Gracias a dicho módulo se ha sido capaz de:

- Establecer la configuración CORS (Intercambio de Recursos de Origen Cruzado),
- Establecer una blacklist de endpoints
- Establecer los endpoints que necesitan autenticación para acceder a ellos
- Establecer qué mecanismo se seguirá para el inicio de sesión
- Encriptación de datos sensibles

Spring, además, gracias a los módulos mencionados anteriormente, abstrae al desarrollador de la creación de las tablas a la hora de la persistencia con la Base de Datos. Me bastó con crear las clases necesarias, anotar dichas clases para asegurar que Spring hiciese la conversión, y con desplegar la aplicación, las clases eran transcritas a tablas en la Base de Datos.

En otro orden de cosas, no solo se ha utilizado Java. Dado que se quiere hacer uso de la Visión Artificial para incluir una mejora a la API, se usa Python. La idea es construir otra API con Python, que proporcione servicios relacionados con el reconocimiento facial. A raíz del uso de Python para incluir dichas mejoras, fueron necesarias el uso de varias librerías. A destacar tenemos:

- Librería relacionada con la conexión a la Base de Datos. Dado que es necesario que se pudiese acceder a la Base de Datos, tanto para consulta como para escritura, fue necesario el uso de esta librería. (mysql.connector [14])
- Librería relacionada con el reconocimiento facial. (face_recognition [6])
- Librería relacionada con la validación de los token JWT. (PyJWT [9])
- OpenCV. Librería relacionada con la Visión Artificial. (cv2 [12])
- Framework web para el desarrollo de la API en Python (bottle [4])

Se podría decir, que estas herramientas han sido las mas importantes a la hora del desarrollo de la aplicación. Gracias a ellas, ha sido posible abordar prácticamente todos los problemas planteados y así poder llegar a una solución.

Entorno de programación

Después de la subsección anterior, se podría intuir qué entorno de programación es el utilizado en este proyecto. El entorno de programación utilizado es STS (Spring Tool Suite). STS es un entorno de programación derivado de Eclipse enfocado especialmente al desarrollo de aplicaciones en Spring Framework. Se decidió utilizar este entorno de programación para el desarrollo porque este proyecto no es el primero que se aborda, durante la carrera he tenido que realizar desarrollos similares y este entorno es el ideal si se decide utilizar Spring como framework para el desarrollo. Dicho entorno de programación, al igual que cuando es necesario utilizar Eclipse, requiere tener instalado Java y Java JDK.

Además, se fue capaz de desplegar la aplicación en local para la prueba de la misma, simulando su puesta en funcionamiento en producción. Una vez que se desarrollaba una funcionalidad específica de la API, se fue capaz de realizar el empaquetamiento de la misma, con el fin de poner la aplicación en producción. El entorno de programación da la posibilidad de realizar el empaquetamiento ajustando un par de opciones.

Por otra parte, al igual que Eclipse, proporciona soporte para test unitarios, utilizando Spring para ello. En una de las subsecciones anteriores se ha hablado de Git para el control de versiones, esta herramienta me ha permitido realizar operaciones de Git desde el propio entorno. Además, se indica junto al proyecto en cada momento la rama en la que te encuentras y el repositorio actual.

Por último, tenemos PyCharm. Entorno de programación especializado en Python. Dado que se pretende realizar tareas de Visión Artificial que implican el uso de Python, es necesario el uso de PyCharm. Permite configurar un entorno de Anaconda para la ejecución del código. Además, da la capacidad de instalar paquetes de Python desde las opciones del proyecto, tiene una potente herramienta para depurar el código, etc.

Gestor de la Base de Datos.

Dado que se trata de una Base de Datos MariaDB, fue necesario una herramienta que facilitase la gestión de dicha Base de Datos. Después de instalar MySQL en el VPS, se decidió el uso de DBeaver. DBeaver es una herramienta de gestión de Bases de Datos MySQL, de código libre y con una interfaz de usuario muy intuitiva. Basta con elegir qué tipo de Base de Datos se quiere gestionar y especificar los datos de conexión. La herramienta hace lo demás. Con esta herramienta, he sido capaz de hacer cambios en la Base de Datos permitiendo la posibilidad de tanto hacerlo mediante la interfaz de gestión como de ejecutar sentencias SQL que la herramienta se encarga de gestionar.

Métricas

Por último, sobre todo enfocado para el tema de métricas, se ha utilizado SonarQube. Esta herramienta código abierto, se encarga de analizar el código fuente en busca de errores, código que en un futuro podría ocasionar errores graves, mide la cobertura del código, etc. Basta con indicarle el tipo de proyecto a analizar, el lenguaje utilizado y la herramienta da las indicaciones pertinentes para realizar el análisis del código. Dicho análisis se traducirá en las medidas mencionadas anteriormente. Se entrará más en detalle sobre estas métricas en el capítulo de métricas.

3.2. Arquitectura del software

En esta sección se describirán cada una de las partes de la solución que se desarrolla.

Indicaré las características en cada subsección, separando dicha sección en dos grandes puntos: el servicio de gestión de la red social y el servicio de r

3.2.1. Servicio de gestión de la red social



En esta subsección se va a entrar en sobre la API REST desarrollada con ayuda de Spring Framework. Como introducción podemos ver en la Figura 3.2.1 la estructura del proyecto, organizada por paquetes para mantener la mantenibilidad del código. Como ya se dijo en capítulos anteriores junto al nombre del proyecto se indica el repositorio y la rama actual. En un proyecto en Spring Boot, se distinguen tres carpetas principales “src/main/java”, “src/main/resources” y “src/test/java”. En la primera carpeta se encuentra el código fuente de la aplicación, eso es, la funcionalidad completa de la aplicación. En la segunda carpeta se encuentran los recursos estáticos (ficheros .js, .css, .png, etc) y las plantillas de la aplicación si las hubiese, además del fichero de configuración “application.properties” del que se hablará más adelante.

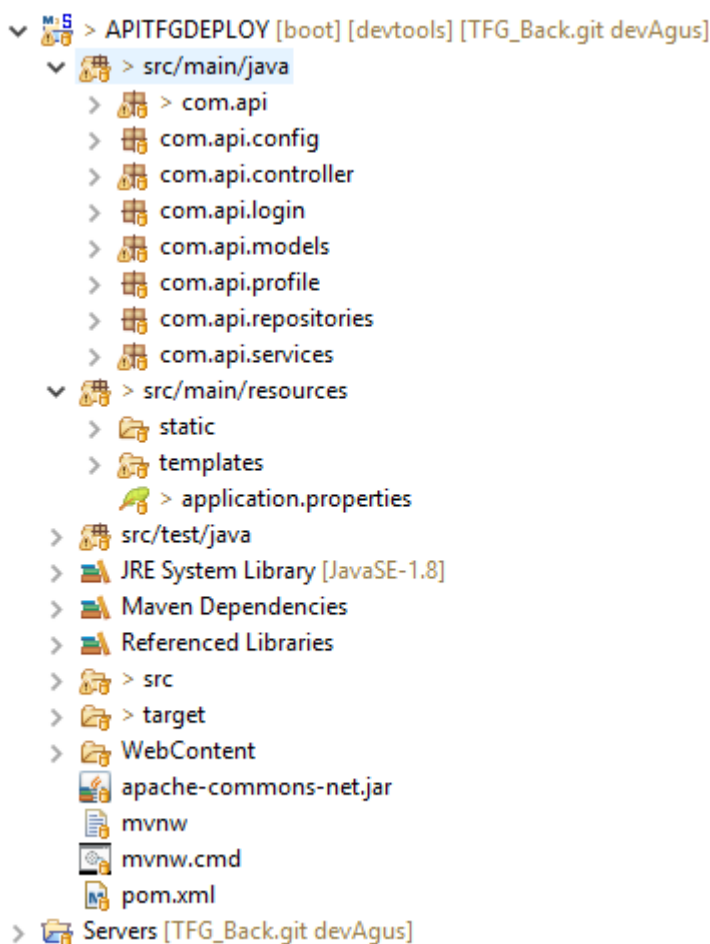


Figura 3.2.1: Estructura del proyecto

La estructura de toda API REST desarrollada en Spring tiene la estructura que se ve en la Figura 3.2.2. A lo largo del capítulo se irá haciendo referencia a dicha figura con el objetivo de ir explicando cada una de las partes.

Spring IoC Container y los beans

Antes de entrar en detalle sobre cada una de las partes de la aplicación, es necesario explicar cómo Spring es capaz de realizar todo lo que se va a ver en futuras secciones. En primer lugar, se tienen los beans. Los beans son los objetos que componen la columna vertebral de la aplicación y son administrados por el contenedor de Spring (Spring IoC Container). Cabe destacar que un bean podría ser un controlador, acceso a la base de datos, configuración de la seguridad de la aplicación, etc. Es decir, los beans son los objetos que contienen la funcionalidad de la aplicación y son configurados por el contenedor de Spring. El contenedor cuenta con el principio de IoC (Inversion of Control) también conocido como Inyección de Dependencias. Es un proceso en el cual los objetos definen sus dependencias, esto es, otros objetos con los que trabajan dichos objetos, los argumentos del constructor, propiedades que se establecen en la instancia del o

El contenedor de Spring sabe qué objetos instanciar, configurar o ensamblar gracias a los metadatos de configuración. Estos metadatos de configuración están representados a su vez por un fichero XML, por anotaciones Java o por código Java. El contenedor de Spring esta representado por la interfaz BeanFactory y ApplicationContext. La interfaz Bean-

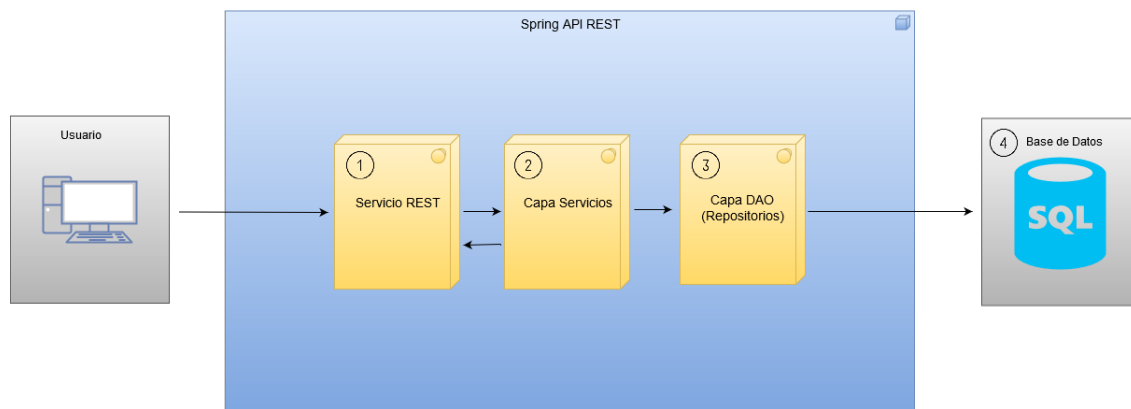


Figura 3.2.2: Diagrama API REST en Spring

Factory provee un mecanismo avanzado de configuración capaz de administrar cualquier objeto, `ApplicationContext` es una interfaz que implementa `BeanFactory` y además de las características propias de la interfaz `BeanFactory`, implementa una serie de características mas avanzadas para el manejo de otro tipo de servicios.

En resumen, el desarrollador provee a través de un fichero XML, mediante anotaciones o mediante código Java, los beans que necesita el contenedor de Spring. Dicho contenedor se encargar de tomar esos beans y configurarlos, ensamblarlos e instanciarlos.

■ la Figura 3.2.2, el contenedor Spring sería el recuadro azul, esto es, la base sobre la construye todo lo demás.

Observando

Modelos

Como ya se ha introducido en la sección de las herramientas utilizadas durante el desarrollo, Spring abstrae al desarrollador de la transcripción de las clases a las tablas en la Base de Datos. Por lo que **tenemos** modelos basados en datos. Suponiendo que se tiene configurada la conexión con la Base de Datos, en cuanto la aplicación es desplegada es posible establecer determinadas opciones para la creación de las tablas correspondientes y los datos contenidos en dichas tablas.

Para poder realizar la creación de estas tablas, es necesario la creación de clases con una serie de anotaciones. Se tienen que anotar las clases con el objetivo de constituyan una entidad y puedan ser traducidas a tablas de la Base de Datos. Además, hace falta especificar cuál será la clave primaria de la entidad. Posteriormente, tanto en desarrollo local como en producción, una vez desplegada la aplicación, Spring usando lo especificado en un fichero de configuración del que se hablará más adelante, establece la conexión con la Base de Datos, traduciendo las relaciones entre clases y las propias clases anotadas pertinentemente en tablas, con sus claves primarias y foráneas en el caso de que existan relaciones.

La estructura de los modelos queda representada en la Figura 3.2.3. Entrando más en detalle sobre dichos modelos tenemos:

- Clase Usuario. Podría decirse que es la entidad central de la aplicación. Un usuario puede poseer:
 - N publicaciones

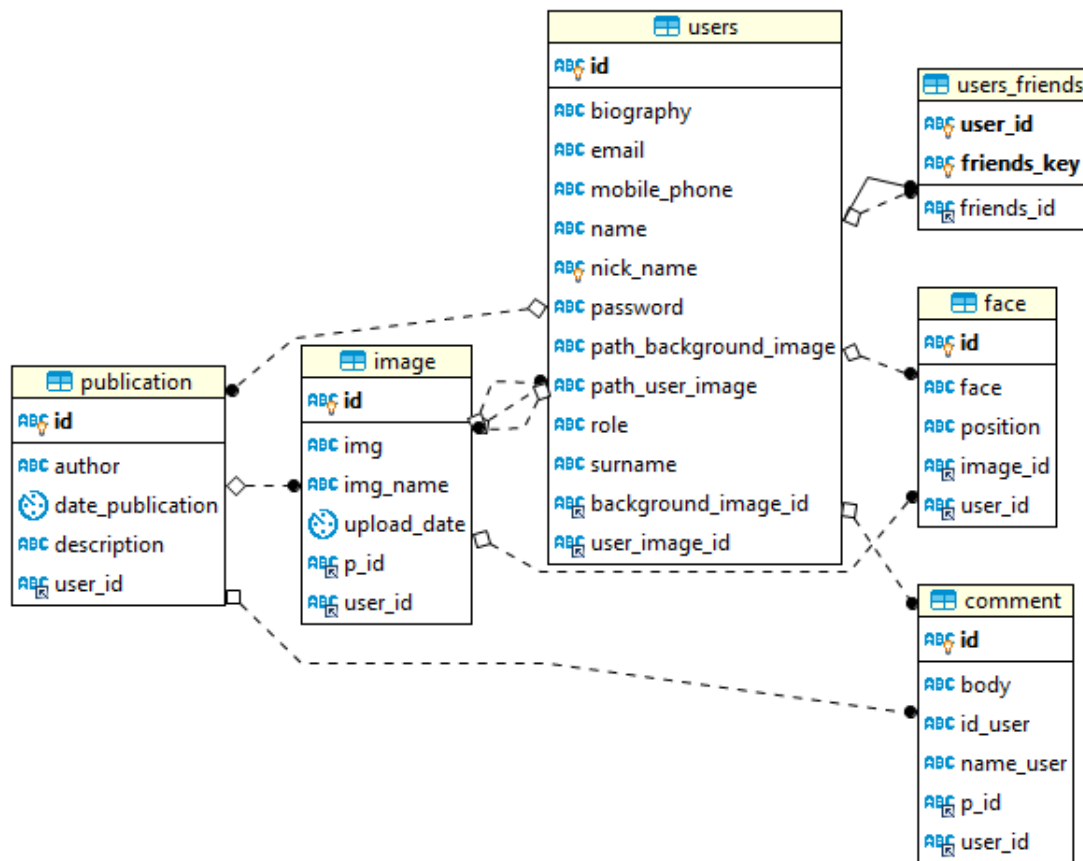


Figura 3.2.3: Diagrama Entidad-Relación

- N comentarios
 - N imágenes provenientes de las sus propias publicaciones
 - N amigos
 - N caras procedentes de sus imágenes.
- Clase Publicación. Una publicación se compone de imágenes y comentarios.
 - Clase Comentario. Un comentario solamente puede pertenecer a un usuario y a una publicación.
 - Clase Cara. Las caras procedentes de las fotos, solo puede provenir de una sola foto y pueden pertenecer a un solo usuario. Cabe destacar que, estas caras que pertenecen al usuario son caras que se han extraído de imágenes contenidas en publicación que el propio usuario ha subido. Está claro que, solamente serán caras de amistades del usuario.
 - Clase Imagen. La clase Imagen solo puede pertenecer a una publicación y a un usuario. Además puede poseer N caras extraídas de la propia foto.

Controladores

Los controladores de la aplicación contienen toda la funcionalidad de la aplicación. Son el punto de acceso al que el cliente puede acceder para obtener los recursos que necesita. En este caso, pasa parecido que con los modelos. Es necesario anotar la clase de cierta manera, para que Spring sepa que una clase es un controlador que tiene como objetivo atender la solicitud de recursos que hace el lado cliente. Si nos fijamos en la Figura 3.2.2 se tiene que los controladores serían el Servicio REST (1). Se trata del punto de acceso que tiene el lado cliente para poder acceder a los recursos que necesita.

De este modo, una clase anotada como controlador, contendrá en su interior una serie de métodos que se encargará de proveer recursos al lado cliente. Cada método estará anotado pertinentemente. Por cada método, es necesario decir qué tipo de método es (GET, POST, DELETE, PATCH, etc) y qué ruta atiende. Además, si la petición HTTP envía parámetros en la URL, Spring provee de funcionalidad para poder recoger estos parámetros y utilizarlos para los fines que se crean necesarios.

Spring tiene la capacidad de serializar la información que se devuelve al lado cliente. Siempre y cuando esta información sea serializable. Por lo que, basta con crear una clase con atributos que sean serializables, al crear una instancia de ese objeto y devolverlo, Spring haciendo uso del serializador Jackson devolverá el objeto en formato JSON al lado cliente. De esta forma, si tuviésemos un método GET que se encargase de devolver un usuario determinado del sistema y suponiendo que la clase Usuario contiene atributos serializables, la instancia de la clase Usuario que se debe devolver, será serializada y será devuelta en formato JSON al lado cliente.

Por otra parte, la información que proviene del lado cliente puede ser interpretada por Spring. Si un método de tipo POST determinado recibiese un objeto de la clase Usuario, bastaría con añadir un argumento al método anotado debidamente, de esta forma, Spring se encargará de formar el objeto de dicha clase con la información en formato JSON proveniente del lado cliente.

Ahora bien, los controladores que existen en la aplicación son:

- Controlador de usuarios. Este controlador se encarga de proveer recursos relacionados con los usuarios. Y, en concordancia con los requisitos funcionales especificados en el capítulo de Análisis, tenemos:
 - Obtener todos los usuarios.
 - Obtener a un usuario dado su identificador.
 - Editar a un usuario.
 - Eliminar a un usuario.
 - Creación de un usuario.
- Controlador de publicaciones
 - Obtener todas las publicaciones
 - Obtener una publicación dado su identificador
 - Editar una publicación
 - Eliminar una publicación

- Añadir una publicación
- Controlador de comentarios
 - Obtener todos los comentarios
 - Obtener un comentario dado su identificador
 - Editar un comentario
 - Eliminar un comentario
 - Añadir un comentario
- Controlador encargado de la página principal de la aplicación.
 - Carga de la página principal del lado servidor.
- Controlador encargado del registro y del inicio de sesión
 - Registrar a un usuario
 - Iniciar sesión de un determinado usuario
 - Obtener el usuario que ha iniciado sesión
- Controlador encargado de los amigos
 - Obtener todos los amigos
 - Añadir a un amigo
 - Eliminar a un amigo
 - Obtener las publicaciones de tus amigos.
 - Obtener sugerencias de usuarios que podrían ser amistades
- Controlador de utilidades
 - Búsqueda de un usuario dado su nombre de usuario
 - Búsqueda de usuarios dado su nombre
- Controlador del perfil de usuario
 - Obtener un determinado perfil de usuario
 - Editar perfil de usuario
- Controlador de la descarga de imágenes
 - Obtener una imagen determinada

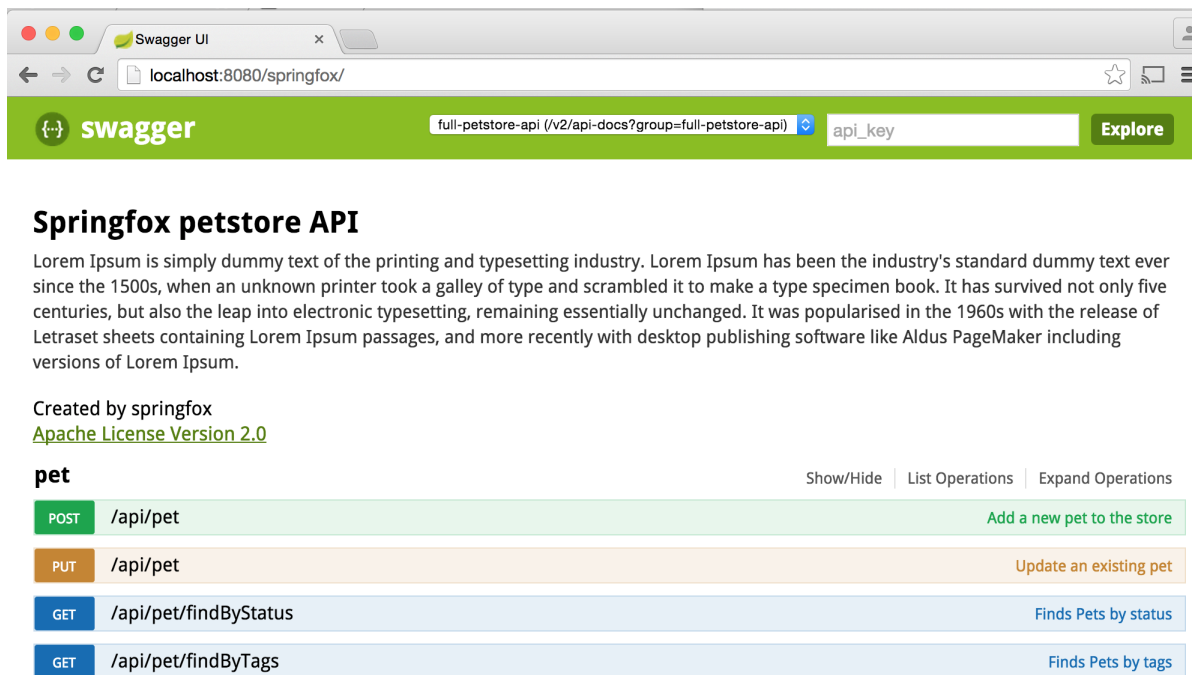


Figura 3.2.4: Swagger UI

Documentación

Para la documentación, se usa un módulo de Spring llamado Springfox Swagger2. Este módulo da la capacidad de realizar la documentación completa de la API en formato JSON de forma automatizada, basta con decirle en que paquete están alojadas los controladores de la aplicación. En cuanto se configura el módulo, indicando dónde se encuentran los controladores y se despliega la aplicación, se habilita un endpoint donde poder acceder y poder descargar la documentación en formato JSON.

Aparte de esto, existe otro modulo derivado del anterior llamado Springfox Swagger UI. Este módulo sigue el mismo proceso de generación de la documentación que el modulo anterior, la diferencia es que habilita un endpoint donde es posible ver la documentación gráficamente en formato HTML y da la capacidad de probar toda la funcionalidad desde dicha página. Además, como se especificará más adelante, hay endpoints en los cuales hace falta iniciar sesión previamente, dicho modulo permite simular dicho inicio de sesión y poder probar todos los endpoints.

Si se accede a la dirección base de la aplicación seguido de “/swagger-ui.html” se puede ver la documentación al completo, ordenando los métodos HTTP por controlador. Una vez se tiene todo configurado, se despliega la aplicación, y se accede al endpoint habilitado por el propio modulo, el resultado se puede ver en la Figura 3.2.4, extraída de la documentación de SpringFox [20]

Configuración y seguridad de la aplicación web

Ya se introdujo en la sección de las herramientas utilizadas el uso de Spring Security para asegurar la seguridad de la aplicación y además es necesaria la configuración de la documentación, como ya se entró en detalle anteriormente.

Ahora bien, para poder hacer que todo esto funcione, es necesario la configuración a través de clases anotadas pertinentemente. En total, se tienen 3 clases que se encargan de la configuración de la seguridad de la aplicación:

- Clase relacionada con la configuración de la seguridad de la web. En esta clase, se establecen configuraciones básicas sobre la seguridad de la aplicación. Se establece la encriptación de la contraseña de los usuarios, se establece la configuración CORS de la aplicación con el fin de una comunicación exitosa del lado cliente.
- Clase relacionada con el acceso a los recursos de la aplicación. En esta clase, se establece una lista blanca y una lista negra de puntos de acceso para el lado cliente. Básicamente, se establecen los recursos a los que el lado del cliente puede acceder y a los que necesita antes autenticación para poder acceder.
- Clase relacionada con la autenticación basada en un token JWT. En esta clase, se establece la configuración de la autorización con token.
 - El tipo de autenticación que se usa en la aplicación es OAuth2.0 y JWT (JSON Web Tokens). Estos servicios proveen al lado cliente el acceso a los recursos de forma limitada, siempre y cuando se proporcionen los datos de autenticación pertinentes. Sabiendo el flujo del token JWT usando OAuth2.0 [7], el lado cliente puede acceder a los recursos con la debida autenticación. Concretamente se usa OAuth2.0 Password Grant Type. Como se ve en el flujo de datos mostrado en la Figura 3.2.5 proporcionado por la documentación de Oracle [2], se habilita un endpoint, dado por la propia dependencia de OAuth, al cual es necesario hacer una petición POST con una serie de parámetros que necesita OAuth para realizar la autenticación. Estos parámetros los establece OAuth, pero los valores los pone el lado del servidor y los comparte con el lado del cliente.
 - El usuario introduce su usuario y contraseña (previo registro), el lado del cliente realiza una petición a dicho endpoint con los parámetros pertinentes para obtener el token y a partir de ahí, el cliente debe de mandar el token en cada petición que se haga para confirmar que el usuario sigue autenticado. El token tiene un tiempo de expiración de 12 horas establecido por la propia API. Ahora bien, dentro de la aplicación existe un fichero de configuración, del cual se entrara en detalle más adelante. En dicho fichero se establecen los valores para los parámetros que necesita OAuth. En esta clase, se toman esos valores dados en el fichero y se establece la autenticación OAuth con la que el cliente debe acceder.

Además, Spring permite configurar la aplicación web desde un fichero dentro del propio proyecto. En **mi** caso el fichero se denomina “application.properties”, en dicho fichero van parámetros de configuración globales. En dicho fichero, se especifican los parámetros que necesita Spring para la autenticación mediante OAuth2.0.

Repositorios

Los repositorios o DAOs (Data Access Object) son las herramientas que se encargan de la persistencia de la aplicación. Proporcionan una capa de abstracción para el desarrollador,

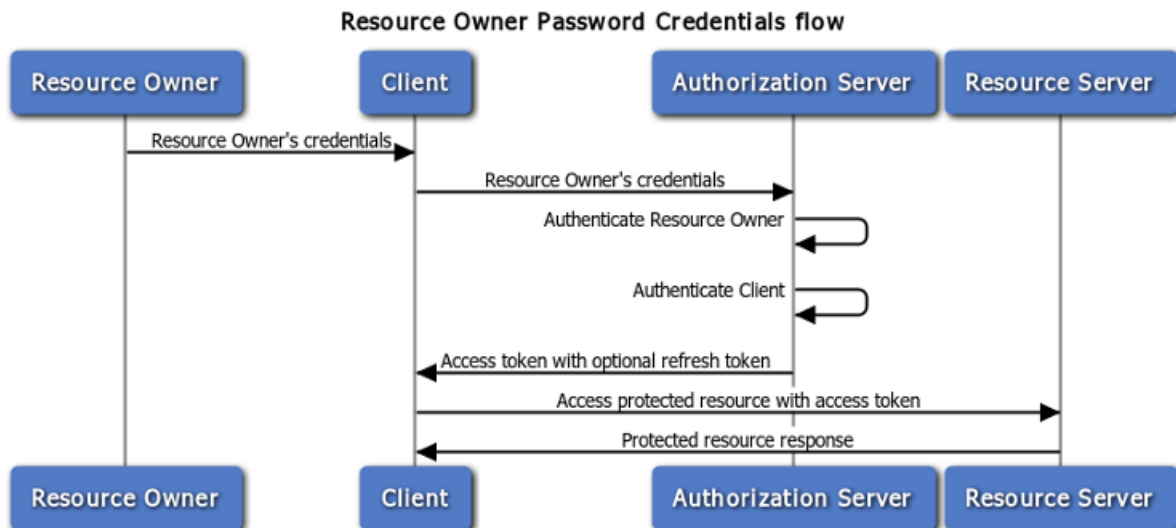


Figura 3.2.5: Flujo de datos de OAuth2.0 con token JWT

abstrayendolo de las consultas SQL requeridas para poder añadir, consultar o eliminar datos de la Base de Datos.

La interfaz JpaRepository contiene una serie de métodos básicos que implementan todas las operaciones que se pueden realizar (crear, leer, actualizar y eliminar) con respecto a la Base de Datos. Además es posible que el desarrollador pretenda realizar consultas cambiando el criterio de búsqueda, para ello, simplemente se crea la cabecera del método en la interfaz. El nombre del método debe de seguir un formato concreto: Debe empezar por “findBy” seguido del atributo de la clase por el que se quiere buscar. Debe pasarse por parámetro el atributo por el que se quiere buscar, con el tipo adecuado. Si el objeto que se devuelve no es una lista, quiere decir que el método encontrará uno y lo devolverá. En cambio si se devuelve una lista, el método buscara todas las ocurrencias del valor del parámetro que se le pase.

Un solo repositorio no va a poder hacerse cargo de todos los modelos de la aplicación, ya que cada repositorio se asocia al tipo del modelo sobre el que se quieren realizar una serie de operaciones.

Haciendo referencia a la Figura 3.2.2, se trata de los Repositorios (3) o DAOs. Es la ultima puerta que se cruza para almacenar la información en la Base de Datos.

Servicios

sino que se

son

A diferencia del apartado anterior, los servicios no son una herramienta proporcionada por Spring. ■ implementa ■ por parte del desarrollador. En mi caso, he implementado un servicio por cada uno de los repositorios que existen. Dichos servicios deben ir anotados pertinentemente y tendrán métodos que implementaran los mismos métodos que el DAO.

El objetivo de la creación de Servicios es proporcionar mantenibilidad al código. La creación de servicios implican proporcionar una capa de abstracción para el desarrollador, de manera que si es necesario cambiar la implementación de algunas de las operaciones de los repositorio, bastaría con hacer las modificaciones pertinentes en los servicios sin que el código de los controladores quedase afectado.

Los servicios existentes en la aplicación son los siguientes:

- Servicio dedicado a los usuarios
- Servicio dedicado a las publicaciones
- Servicio dedicado a los comentarios
- Servicio dedicado a las imágenes
- Servicio dedicado al servidor donde se aloja la aplicación
- Servicio dedicado a las caras extraídas de las imágenes

son

Dado que los servicios son una capa que se encuentra entre los controladores y los repositorios, en la Figura 3.2.2, se trata de la Capa Servicios (2) y se encarga de mediar entre el Servicio Rest (1) y los DAOs (3) proporcionando la capa de abstracción de la que ya se ha hablado.

Base de Datos

En el capítulo de Análisis se introdujo un poco a la Base de Datos dado es uno de los recursos externos utilizados. Se utiliza un sistema de gestión de Bases de Datos derivado de SQL: MariaDB. Una vez se configura debidamente en el VPS, en Spring se establece la configuración global en el fichero “application.properties” para que la aplicación pueda conectarse a la Base de Datos. Además, es necesario realizar una integración del tipo de Base de Datos en Spring. Para ello existe una dependencia que se encarga de integrar MariaDB en Spring.

Cabe destacar que se tienen dos bases de datos, una usada para el desarrollo en local, para pruebas. Y otra para producción, usada en el despliegue de la aplicación. Una vez más, para cambiar una Base de Datos por otra basta con indicarlo en el fichero de configuración nombrado anteriormente

Y por último, haciendo referencia a la Figura 3.2.2, la Base de Datos (4) es el último destino para mantener la persistencia de la información. Siendo los DAOs (3) los encargados de administrar el almacenamiento de dicha información.

3.2.2. [REDACTED] Servicio de reconocimiento facial

[REDACTED]

de reconocimiento facial

Antes de proceder a la descripción de cada una de las partes del algoritmo, es necesario aclarar una serie de cuestiones. La [REDACTED] usada para el reconocimiento facial (face_recognition) se basa en el deep learning, concretamente mediante el uso de redes neuronales, para el reconocimiento facial. Dicha [REDACTED] no necesita que el desarrollador le provea un conjunto de test para el correcto reconocimiento de caras en una foto.

biblioteca

En un primer momento, [REDACTED] se pretendía [REDACTED] que un controlador de la API REST de Spring se ejecutase un script de Python que realizase la funcionalidad. [REDACTED] e Esto no fue posible en un entorno web, por lo que fue necesario la realización de una segunda API REST hecha en Python. Por otra parte, si se presta atención al repositorio de dicha librería en GitHub [6] se puede ver que puede usarse junto con un framework web, en este caso se usa Flask, el cual es muy parecido al usado en este caso.

de gestión de los servicios de la red social

Algoritmo

biblioteca Face Recognition

para los servicios de reconocimiento facial.

Como ya se ha dicho, la [REDACTED] no necesita ser entrenada para realizar su cometido. Por lo que basta con proporcionarle una foto y esta nos devolverá las posiciones de la cara (en píxeles) en la foto. En la Figura 3.2.6 podemos ver el algoritmo que se sigue a la hora de subir una foto, representado mediante un diagrama de actividad se ha incluido el lado cliente para poder explicar mejor la utilidad de la API en una aplicación web. El algoritmo que se sigue es el siguiente, teniendo en cuenta la Figura 3.2.6:

1. El usuario se dispone a subir una foto
2. Se muestra una vista previa de la foto
3. Se piden las caras al lado servidor. y si es posible, se piden también sugerencias
4. El lado servidor devuelve caras y puede (o no) devolver sugerencias
5. El lado cliente dibuja las caras en la vista previa, y sugerirá al usuario si es posible
6. El usuario etiqueta las caras con los usuarios y se lo comunica al lado servidor a través del lado cliente
7. El usuario, si está conforme, confirmará la subida de la foto
8. Vuelta al paso 1.

Ahora bien, cabe decir, que el lado servidor devolverá caras siempre y cuando existan caras en la foto. Si no existen caras en la foto, no se devolverán, además de no devolver sugerencias [REDACTED]. Si es posible devolver caras, el servidor devolverá caras además de sugerencias dependiendo de las fotos subidas anteriormente por el usuario. El algoritmo aprende [REDACTED] las caras que el usuario ha registrado anteriormente. Por lo que, si el usuario sube su primera foto a la aplicación, es posible que el servidor devuelva caras, pero no devolverá en ningún caso sugerencias [REDACTED]. El usuario solo podrá etiquetar caras con otros usuarios si y solo si estos usuarios pertenecen a su lista de amigos. En otro caso, no será posible [REDACTED], etiquetarlas.

Además, el algoritmo es capaz de mejorar en función de la actividad del usuario en la aplicación, es decir, [REDACTED] más amigos tenga el usuario y más fotos suba con etiquetas, el sistema irá aprendiendo a realizar predicciones más fieles con respecto a sus amigos.

cuantos

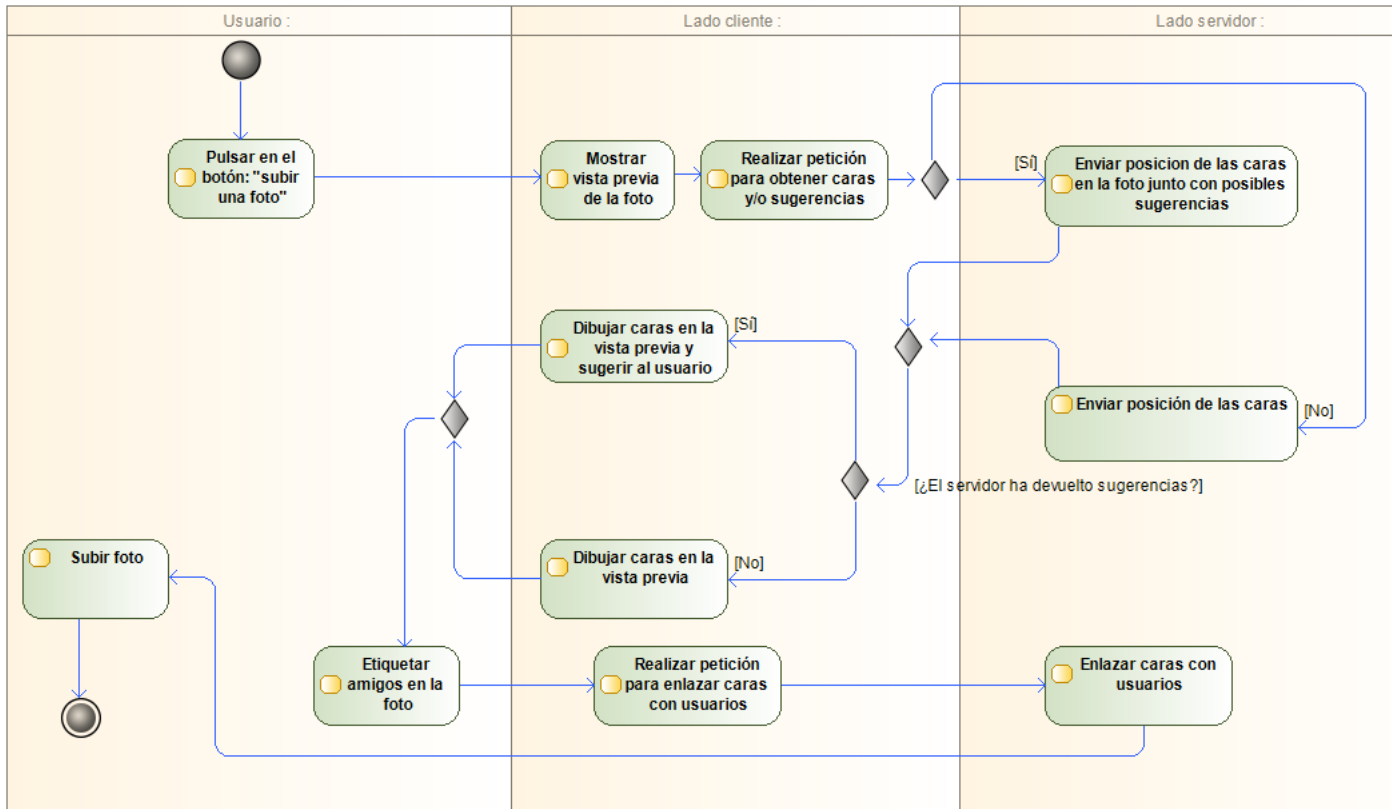


Figura 3.2.6: Diagrama de actividad de la subida de fotos

Controladores

Una vez se ha hablado sobre cómo funciona el algoritmo de reconocimiento, se procede a hablar de la funcionalidad que hace posible que todo funcione correctamente. Se tienen dos controladores principales:

- El controlador encargado de proporcionar caras y sugerencias si es posible. Haciendo referencia a la Figura 3.2.6, este método se corresponde con las actividades (4) y (6). Al principio, se encarga de extraer las posiciones de las caras, pero antes de devolver las posiciones correspondientes, comprueba si hay sugerencias. Por lo que, por cada cara que se encuentra en la foto, se busca en el directorio de caras de las amistades del usuario con el objetivo de encontrar alguna coincidencia y así poder hacer sugerencias. Por lo que a la vuelta, se devuelve en formato JSON, las posiciones de las caras (si se encuentran la foto enviada) y las sugerencias de dicha cara (si se se han encontrado sugerencias de la cara).
- El controlador encargado de enlazar caras con usuarios. Una vez que las posiciones de las caras llegan al lado cliente y son pintadas para que el usuario se encargue de etiquetar las caras con los nombres de usuario de sus amistades (actividades (5), (6) y (7) en la Figura 3.2.6), es necesario que el sistema sepa qué cara se corresponde con qué usuario. Por lo que una vez se ha etiquetado (actividad (9) en la Figura 3.2.6), se realiza la petición para enlazar las caras con usuario, enviando al lado servidor desde el lado cliente, la posición de la cara en la foto, la foto en cuestión y el usuario que se encuentra en esa cara (actividades (10) y (11) en la Figura 3.2.6).

CORS

de gestión de la red social

Al igual que ocurría en la API REST de Spring es necesario realizar la configuración pertinente del mecanismo CORS (Intercambio de Recursos de Origen Cruzado) para ello se desarrolla lo que se denomina un decorador de Python. Un decorador de Python es básicamente una función que es usada para extender el comportamiento de otra función, sin modificar en nada a dicha función. Por lo que, se realiza un decorador para los endpoints que se tienen en la API REST que realice la configuración adecuada, para así poder realizar una comunicación exitosa con el lado cliente

Verificación de JWT

En la sección anterior, se habló de OAuth2.0 y de los JWT que se usan para la autenticación del usuario en el sistema. Dado que esta API REST no pertenece a Spring, es necesario comprobar que el usuario que accede a los recursos esta autenticado. Por lo que, haciendo uso de la librería PyJWT se realiza una validación del token que llega desde el lado cliente. Y se comprueba que el usuario exista en el sistema. Si todo va bien, se le permite al lado cliente acceder a los recursos del lado servidor.

Atendiendo a la Figura 3.2.5, una vez se ha completado el flujo, el lado cliente usa este token para comunicarse directamente con la API REST de Python, esta API se encarga de validar el token gracias a la librería que se ha especificado anteriormente. Una vez se ha autenticado correctamente el usuario podrá hacer uso de los recursos prestados por la API y se procedería al intercambio de datos visto en la Figura 3.2.6.

Servidor multi-hilo

Como se ha visto, el framework Bottle provee toda la funcionalidad necesaria para la construcción de una API REST en Python. Pero como se puede comprobar en la documentación de dicha herramienta, no provee la funcionalidad necesaria para hacer que la aplicación atienda a más de una petición concurrentemente. Por lo que, fue necesaria otra librería que le provea a Bottle esa funcionalidad de la que carece. En la documentación de Bottle se recomiendan varias librerías capaces de realizar dicha funcionalidad como se ve en la Figura 3.2.7, se eligió CherryPy como librería para realizar esta función. Como se pueden ver en la Figura 3.2.7, dicha librería es bastante estable y realiza la función que necesitamos. Además, en un primer momento se optó por el uso de paste, dado que además de la estabilidad, ya sido testado y probado, pero carece de documentación consistente, por lo que se optó por CherryPy.

Bottle ships with a lot of ready-to-use adapters for the most common WSGI servers and automates the setup process. Here is an incomplete list:

Name	Homepage	Description
cgi		Run as CGI script
flup	flup	Run as FastCGI process
gae	gae	Helper for Google App Engine deployments
wsgiref	wsgiref	Single-threaded default server
cherrypy	cherrypy	Multi-threaded and very stable
paste	paste	Multi-threaded, stable, tried and tested
rocket	rocket	Multi-threaded
waitress	waitress	Multi-threaded, powers Pyramid
gunicorn	gunicorn	Pre-forked, partly written in C
eventlet	eventlet	Asynchronous framework with WSGI support.
gevent	gevent	Asynchronous (greenlets)
diesel	diesel	Asynchronous (greenlets)
fapws3	fapws3	Asynchronous (network side only), written in C
tornado	tornado	Asynchronous, powers some parts of Facebook
twisted	twisted	Asynchronous, well tested but... twisted
meinheld	meinheld	Asynchronous, partly written in C
bjoern	bjoern	Asynchronous, very fast and written in C
auto		Automatically selects an available server adapter

Figura 3.2.7: Tabla de adaptadores para Bottle

Capítulo 4

Métricas

En este capítulo se van a explicar las medidas que se han acumulado durante el desarrollo del proyecto. En primer lugar se va a exponer un diagrama de Gantt, para poder reflejar el tiempo empleado en cada una de las fases del desarrollo de este proyecto. Por otra parte, se van a exponer una serie de medidas que se han realizado sobre la aplicación web, relativas a cobertura de código, vulnerabilidades, etc.

4.1. Tiempo empleado en el desarrollo del proyecto

Como se puede apreciar en la Figura 4.1.1, se han seguido las fases del desarrollo del software, dividiendo la ultima fase en dos periodos diferentes dentro del diagrama de Gantt. Por lo que tenemos:

- **Análisis.** Fase crítica en el desarrollo del proyecto, esto supuso que se fuese minuciosos a la hora de extraer los requisitos (tanto funcionales como no funcionales), esto justifica que se necesitase casi un mes para poder pasar al diseño de la aplicación web. Se ha decidido dividir en esta fase en dos subtareas:
 - **Análisis de los requisitos.** Dicha subtarea ha tenido una duración de 13 días, en ella se han recogido todos los requisitos funcionales y no funcionales de la aplicación. Era consciente de que en base a estos requisitos se iba a construir la totalidad del proyecto, esto supuso un gasto de tiempo bastante alto, como se puede comprobar en el diagrama. Era necesario tener controlado y bien definidos los requisitos que se deben de cumplir en la aplicación web. Una equivocación en esta fase supondría que tener volver desde una fase mas tardía en el proyecto a solucionar algún requisito mal definido.
 - **Especificación.** Esta subtarea duró el tiempo restante de la fase de análisis: 11 días. Cuando se tuvo una definición clara de los requisitos, se pasó a realizar el documento de especificación de requisitos. Separando entre requisitos funcionales y no funcionales, con el objetivo de realizar un diagrama de casos de uso en función de los requisitos funcionales. Además fue necesario la estructuración de los requisitos con el objetivo de conseguir la comprensión, preparación, modificación y el mantenimiento. El fin era conseguir una primer aproximación al diseño. Cabe destacar que la duración se ve justificada a lo complicado que

resulta en algunas ocasiones la separación de los requisitos funcionales y no funcionales. Resulta complicado la separación entre lo funcional y no funcional e identificar los que se refieren al sistema como un todo.

- **Diseño y arquitectura.** Una vez el análisis se completó, se procedió al diseño de la aplicación. Fase del proyecto que supone la base sobre la cual se va a construir la implementación de la aplicación. He decidido dividir esta fase en tres subtareas:
 - **Elección del hardware.** Esta tarea sin duda ha sido de las mas cortas de todo el proceso de desarrollo: 2 días. Dado que ya se poseían de antemano el hardware necesario, no fue muy difícil la elección del hardware necesario para el desarrollo.
 - **Elección del software.** En cambio, esta tarea necesitó más tiempo: 7 días. El tiempo empleado para seleccionar el software necesario para el desarrollo se ve justificado en el capítulo de diseño e implementación, fueron necesarias multitud de herramientas para la realización exitosa del proyecto.
 - **Arquitectura.** En el caso de la arquitectura, esta fue la subtaska que ha marcado el fin de la fase y sobre la fue necesaria una gran cantidad de tiempo: 13 días. Aunque fue posible solapar esta tarea con la tarea anterior, dado que me fue posible empezar a realizar algunos bocetos sobre la arquitectura del proyecto.
- **Implementación.** Cuando la fase de diseño fue completada y estuvo definida correctamente la arquitectura, se procedió a la realización de la aplicación. Esta fase es la más larga del proceso, más de un mes de duración: 32 días. Se ha dividido esta fase en dos grandes subtareas:
 - **Implementación de la API REST en Spring.** Esta subtaska de la fase de implementación fue la que mas tiempo llevo, dado que la complejidad de esta parte de la aplicación era mucho mayor que la subtaska siguiente. Esta tarea ocupo todo el tiempo que dura la fase de implementación: 32 días. Esto es debido es que esta parte de la aplicación contiene gran parte de la funcionalidad de la aplicación.
 - **Implementación de la API REST en Python.** Subtaska que ha tenido una duración total de 11 días. Es cierto que la duración es un tanto extensa, pero fue posible trabajar paralelamente en ambas partes de la aplicación, como se puede ver en el diagrama. Esto se debe a que las dos partes de la aplicación son prácticamente independientes entre si.
- **Pruebas.** Fase del desarrollo que ha tenido una duración de 17 días. Esta fase no ha sido dividida como las anteriores. Tenemos la siguiente subtaska:
 - **Realización de test unitarios.** Subtaska que ha supuesto la totalidad de la aplicación. Fue necesario emplear mas de dos semanas en esta fase, porque se quería proporcionar una cobertura del código de más del 80 %. Por lo que se procedió a la realización de una serie de test unitarios que pudiesen cubrir la mayor parte del código. Más adelante se entrará más en detalle sobre la realización de las pruebas.

- **Documentación.** El fin de las pruebas dio paso a la documentación de la aplicación. Esta fase tuvo una duración total de 18 días. Dicha fase ha sido dividida en tres subtareas. Entrando más en detalle:
 - **Organización de las clases.** Subtarea con una duración total de 5 días. Tarea con escasa duración debido a que no se fue para nada una dificultad ordenar el código fuente. Dado que se tenían las herramientas necesarias para poder llevar a cabo dicho proceso de organización. Se procedió a la organización de las clases del proyecto en paquetes ordenados por tipos. Esta organización se puede apreciar en la Figura 3.2.1. Esto sería en el caso de la API REST en Spring. En el caso de Python, no fue necesaria prácticamente ninguna organización, dado que la totalidad de la aplicación se encuentra en un solo fichero organizado por controladores.
 - **Configuración de Swagger para la generación de la documentación.** En cuanto a la documentación de la API de Spring, como se ha dicho en capítulos anteriores, se ha utilizado la herramienta Swagger para la generación automática de la documentación ordenada por controladores. Fue necesario la configuración de Swagger para poder realizar dicha generación. Esto tuvo una duración de 5 días.
 - **Incluir comentarios en el código fuente.** Subtarea más importante de esta fase del desarrollo. Sobre todo para futuras fases del desarrollo, para asegurar el mantenimiento de la aplicación. Por ello, se ha dedicado más tiempo en ella que en las demás subtareas de esta fase: 10 días.
- **Despliegue.** Una vez las fases anteriores se dieron por finalizadas, era hora de poner la aplicación en producción con el objetivo de que el lado cliente haga uso de ella. Esta fase tiene una duración total de casi un mes: 26 días. Esto se debe a que fueron necesarias la realización de bastantes subtareas para tener la aplicación preparada para que el lado cliente pueda obtener los servicios que necesita. Dicha subtareas son:
 - **Preparar aplicación para el despliegue.** Como ya se sabe de capítulo anteriores del documento, el proyecto usa Maven para la gestión de las dependencias. Maven da la posibilidad de empaquetar la aplicación en un solo archivo, con el objetivo de proporcionarlo a Tomcat para su posterior despliegue. Tomcat necesita que la aplicación empaquetada esté configurada apropiadamente para que se pueda ser desplegada. Esto tuvo una duración de 2 días. La duración es escasa debido a que Spring Tool Suite permite realizar la configuración a través de la interfaz gráfica de forma sencilla.
 - **Enlazar el dominio con la IP del servidor.** Una vez se contrató el servidor, como se ha especificado en capítulos anteriores, fue necesario la contratación de un dominio con el objetivo de cifrar los datos mediante SSL y poder utilizar el protocolo HTTPS. Cabe destacar que fue necesario esperar entre 24 y 48 horas para que el registro de tipo A se propagase por los servidores DNS. Por todo esto, esta subtarea ha tenido una duración total de 2 días.
 - **Cifrado SSL.** Subtarea directamente relacionada con la subtarea anterior con una importancia significativa. Una vez se enlazó la IP del servidor con el dominio, se pudo proceder al cifrado SSL de los datos. En este caso fue sencillo

gracias a Let's Encrypt, el cual proporciona un bot capaz de realizar el encriptado de la información siguiendo una serie de pasos. Una vez se cifraron los datos mediante SSL, se pudo proceder a utilizar el protocolo HTTPS. Esta tarea tuvo una duración de 7 días.

- **Configuración del proxy de Apache.** Subtarea que se encuentra relacionada con las dos subtareas anteriores. Una vez se realizó la subtarea anterior, con el objetivo de eliminar el puerto donde escuchaba la aplicación por cuestiones de seguridad, se configuró el proxy de Apache para que todo el tráfico de datos fuese redirigido a través del proxy y así poder eliminar el puerto de la URL. Para ello, fue necesario una duración de 8 días.
- **Despliegue de la Base de Datos.** Para el despliegue de la Base de Datos fue necesario instalar MySQL en el servidor y posteriormente crear la Base de Datos junto con los permisos pertinentes con el objetivo de que no fuese accesible desde cualquier máquina. Subtarea que ha durado un total de 13 días. Periodo un tanto extenso debido a que se realizaron una serie de configuración en la Base de Datos para su uso por parte de Spring. Cabe recordar que fue necesario la integración de MariaDB en Spring, lo que retrasó bastante el fin de esta subtarea.
- **Puesta en producción de la aplicación.** Una vez las subtareas anteriores se completaron fue posible poner la aplicación en producción. Esta subtarea ha tenido una duración total de 7 días. Duración un tanto extensa debido a que fueron necesarias comprobaciones por parte del lado cliente de que todo funcionaba correctamente.

■ Mantenimiento

- **Corrección de errores o bugs.** Una vez la aplicación fue desplegada, se reportaron varios errores y bugs por parte del lado cliente que fue necesario solucionar. Es obvio destacar, que dicha aplicación se encuentra en continuo desarrollo a día de hoy en cuestión de solución de errores y bugs pero en el momento del despliegue se detectaron errores críticos que fue necesario solucionar con la mayor brevedad posible, lo cual duró 6 días.
- **Desarrollo de nuevas funcionalidades.** Pasa algo parecido que con la subtarea anterior, la cual se ha realizado paralelamente con ella, la aplicación sigue incluyendo nuevas funcionalidades a día de hoy. Pero el lado cliente, requería de ciertas funcionalidades por temas de comodidad, por lo que fue necesario un gasto de tiempo de 12 días.

4.2. Métricas relativas a la implementación

Como se dijo en el apartado de herramientas utilizadas, se utiliza SonarQube para la realización del análisis del código fuente.

En la Figura 4.2.1 se muestra el primer análisis realizado por SonarQube a pesar de que en la Figura se puede ver como no es el primer análisis que se realiza, esto se debe a que se realizó un primer análisis de prueba para comprobar que la herramienta analizaba los

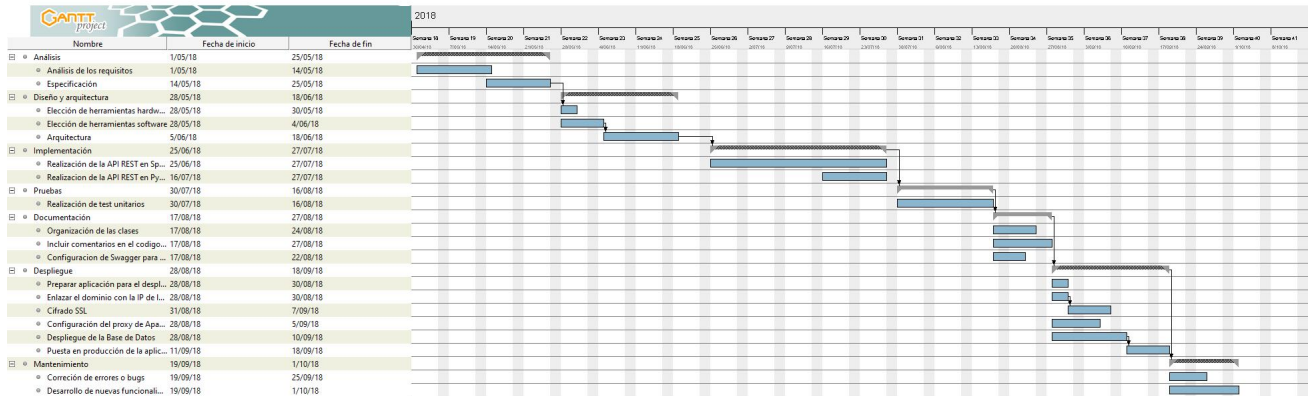


Figura 4.1.1: Diagrama de Gantt sobre las fases del desarrollo.

ficheros que se debían de analizar. Tenemos cuatro puntos a analizar con respecto a la Figura 4.2.1:

1. Vulnerabilidades. Las vulnerabilidades presentes en el código fuente son:
 - a) Uso de funciones que devuelven un valor y no hacer nada con ese valor devuelto
 - b) Uso de un logger en el caso de que se produzca una excepción
 - c) Utilización de objetos POJO o DTO en lugar que los modelos definidos en el proyecto
2. Code Smells. Este punto hace referencia a fallos en el código que actualmente no suponen un problema, pero que podría serlo en un futuro. En este primer análisis, se detectaron estos errores en el código fuente:
 - a) Eliminar importaciones de librerías que no se usan
 - b) Definición de constantes para literales que se repiten N veces
 - c) Eliminar código fuente comentado
 - d) Reemplazo de algunas anotaciones por otras que realizan la misma función en menos código
 - e) Eliminar el tipado en la notación de diamante en algunos métodos.
 - f) Reemplazar los System.out por el uso de un Logger
3. Cobertura de código. A pesar de ser el primer análisis realizado en la aplicación, cabe destacar que este análisis fue realizado después de la realización de las pruebas unitarias de la aplicación y a pesar de que no pasa el control de calidad de Sonar-Qube, se tiene un 78 % cobertura de código por parte de los test unitarios. Además, en este punto del análisis, no se habían excluido los ficheros de configuración de la aplicación, el fichero referente a desplegar la aplicación en local, los ficheros referentes a los repositorios que no dejan de ser interfaces vacías, dado que los repositorios implementan la interfaz encargada de realizar las operaciones básicas contra la Base de Datos.

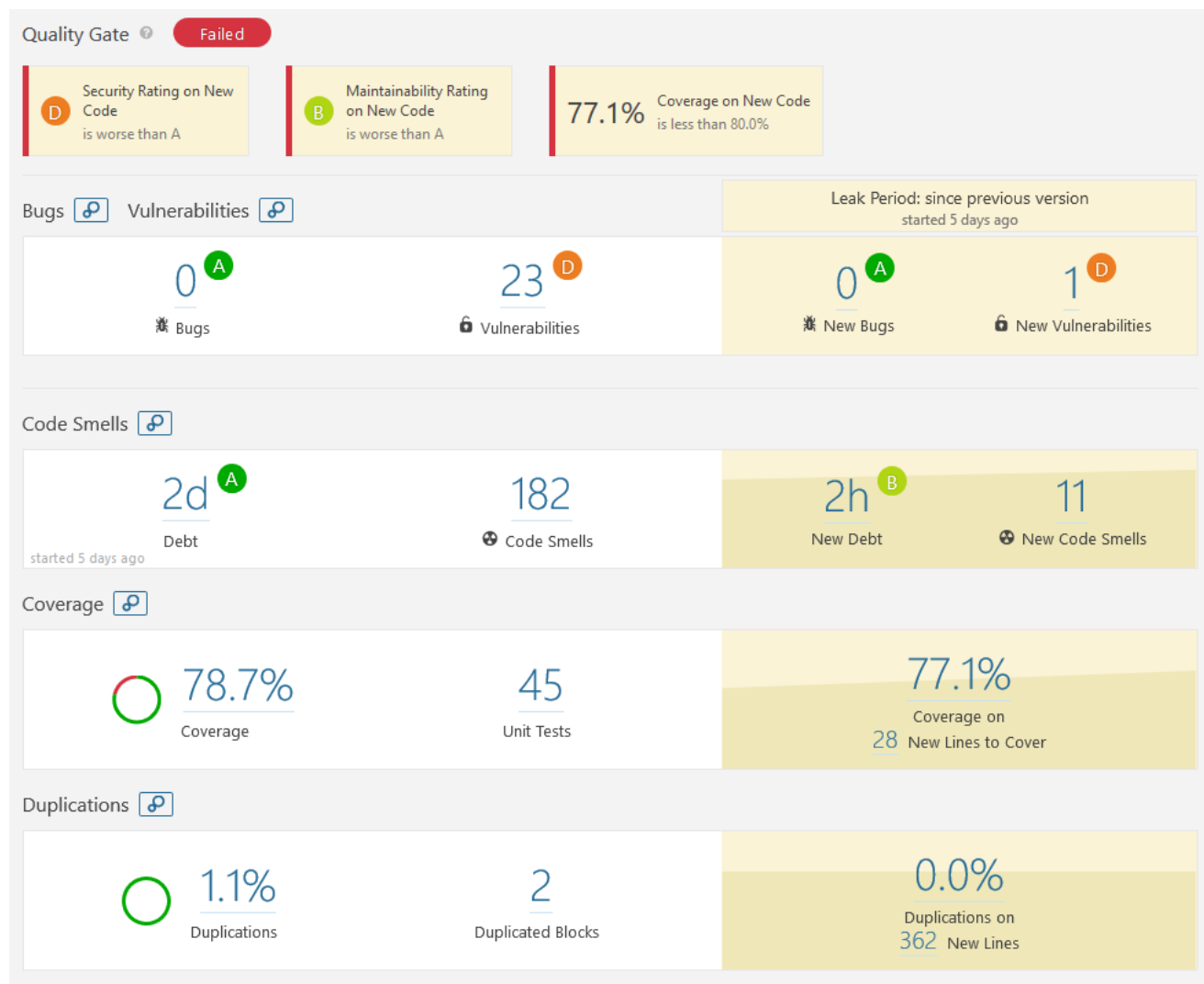


Figura 4.2.1: Primer análisis de SonarQube sobre la aplicación

4. Duplicación de código. En el primer análisis ya se tuvo un tanto por ciento de publicación bastante bajo. Este resultado no es novedad en una aplicación de este tipo, resulta bastante complicado que dos controladores cualquiera que responden a dos peticiones HTTP cualquiera se parezcan. Aún así, en la aplicación existe un pequeño índice de duplicación correspondiente a dos bloques de código correspondientes a controladores referentes a la búsqueda de información, dado que el código es idéntico hasta que se procede a la búsqueda por un determinado atributo.

Una vez se realizó el primer análisis con SonarQube, se procedió a solucionar los fallos en el código. Y se realizó un segundo análisis en el que se tuvo una mejora considerable en todos los puntos descritos anteriormente.

El segundo análisis se tiene en la Figura 4.2.2. En este segundo análisis, me centré más en mejorar la cobertura de código de los test unitarios, dado que existían bastantes porciones de código que realizaban funcionalidades de mucha importancia en la aplicación. Además, dado que se han reducido las vulnerabilidades del código y los Code Smells, la aplicación ha pasado el control de calidad de SonarQube. Como se puede apreciar, no ha habido mejora alguna en los Code Smells, dado que de cierta forma, son los menos urgentes a solucionar.

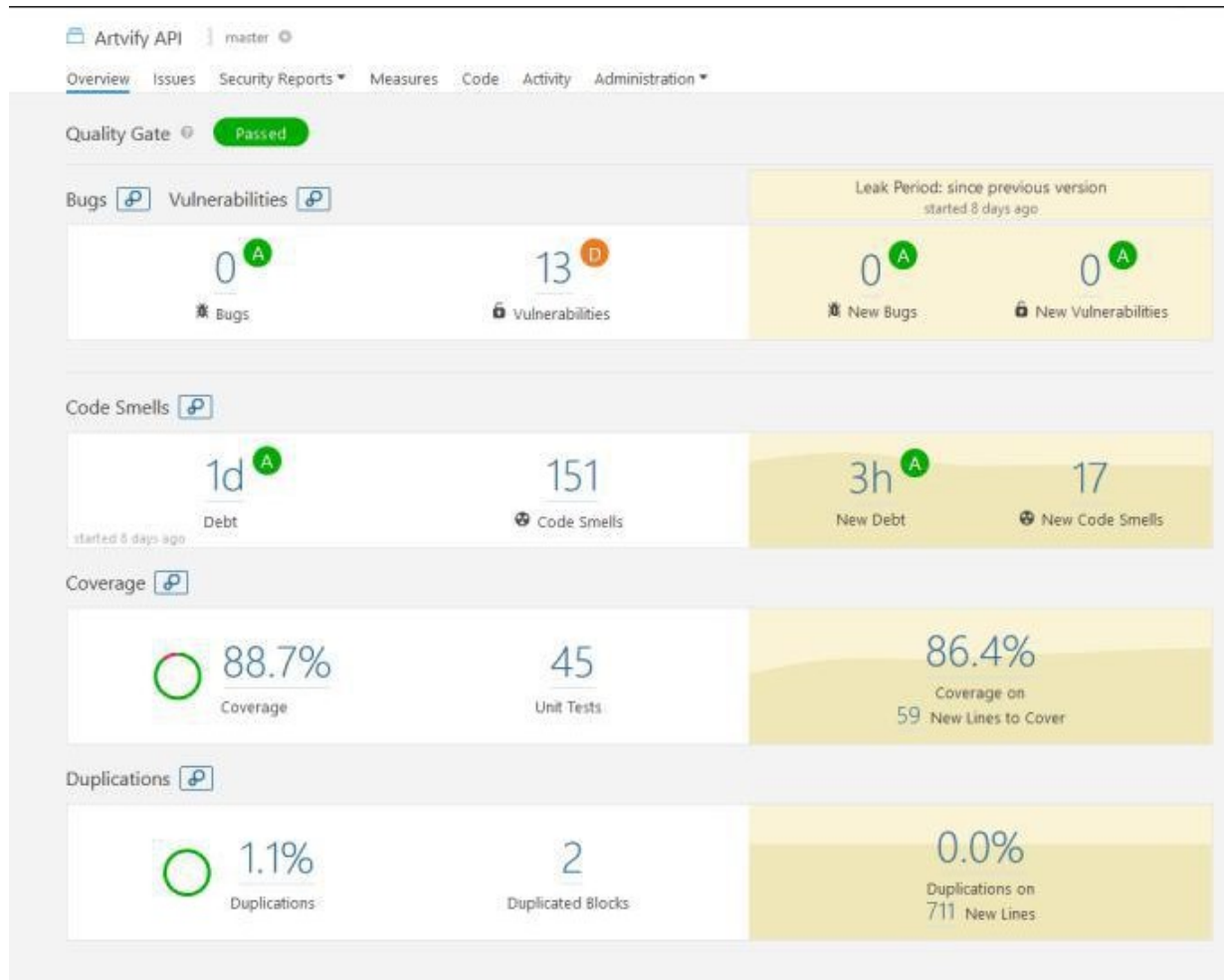


Figura 4.2.2: Segundo análisis de SonarQube sobre la aplicación

Tampoco en el caso de la duplicación, de momento se ha prescindido de mejorar el tanto por ciento de duplicación, dado que se da concretamente en el controlador encargado de la búsqueda de Usuarios. Esto último se verá en el siguiente y último análisis.

En el tercer análisis visto en la Figura 4.2.3. Este último análisis trae consigo las mejoras definitivas con respecto al código fuente. Se han realizado las siguiente mejoras:

- Se han solucionado todas las vulnerabilidades y los bugs en el código. Como se precisó anteriormente, se ha incluido un logger en la aplicación para mostrar excepciones. Se han creado DTO o POJO en el proyecto, con el objetivo de no utilizar los modelos que se han definido en el proyecto. Además, existían una serie de funciones que devolvían un valor y no se hacía nada con ello. En el caso de este código, se trataba de funciones que daban permisos a determinados ficheros y que devolvían un valor booleano indicando si el permisos pudo ser modificado.
- Se han solucionado la gran parte de los Code Smells. En el caso de los Code Smells, se han eliminado las importaciones de librerías que no tienen un uso en el código. Se ha creado una clase de constantes con el objetivo de la utilización reiterada de literales. Se han eliminado grandes cantidades de código comentado que podía suponer un problema. Se han reemplazado las anotaciones “@RequestMapping” por anotaciones propias de cada método HTTP, por ejemplo “@GetMapping”. Se ha

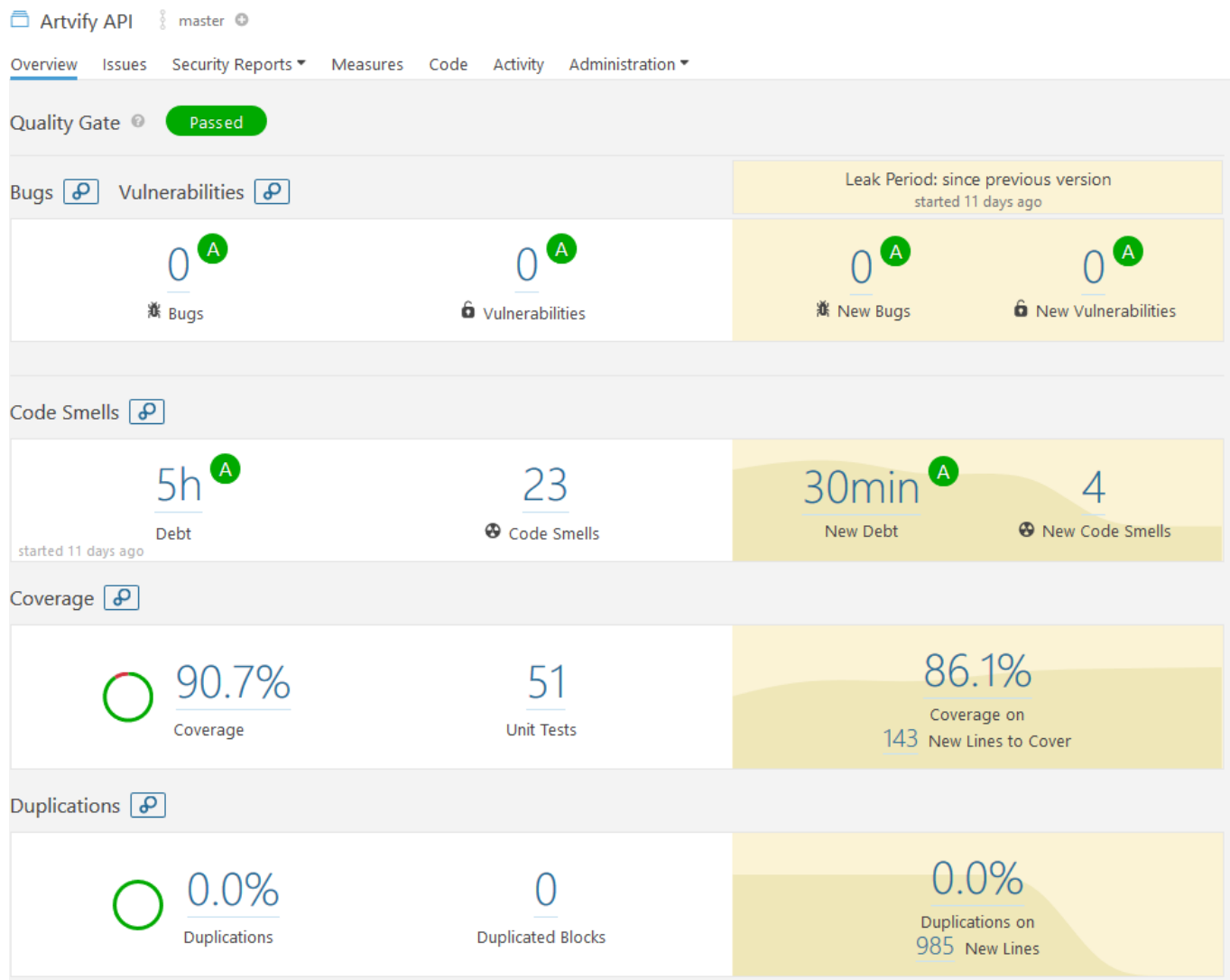


Figura 4.2.3: Tercer análisis de SonarQube sobre la aplicación.

eliminado el tipo en la notación de diamante por no ser necesario en uno de los lados de la asignación para la declaración de objetos. Se ha usado, al igual que con las excepciones, con objetivo de depurar el código el uso de un logger en lugar de mostrar por salida estándar la información haciendo uso de “System.out”

- Se ha conseguido una cobertura de casi el 90%. Se han añadido test unitarios para aumentar la cobertura del código teniendo el tanto por ciento que se ve en la Figura 4.2.3.
- Se ha conseguido una duplicación del 0%. Con objetivo de llegar al 0% de duplicación en comparación con los análisis anteriores, se realizaron cambios en los controladores para que no se repitiesen los dos bloques de código.

Capítulo 5

En geenral se puede concluir que se han cumplido todos los objetivo

Conclusiones

En este último capítulo se van a presentar las conclusiones del Trabajo de Fin de Grado junto con las posibles continuaciones del mismo. Se expondrán dos secciones donde se explicarán estos temas.

5.1. Objetivos cumplidos

- **Proporcionar al lado cliente una serie de servicios que le provea de los recursos que necesita.** Se han construido dos API REST. Una de ellas desarrollada en Java con ayuda de Spring Framework. Dicha API contiene el 80 % de la funcionalidad y es capaz de proporcionar al lado cliente la gran parte de los recursos que necesita. La otra API, desarrollada en Python mediante un framework web llamado Bottle, contiene una mejora relacionada con al Visión Artificial, de la cual se hablará en objetivos siguientes.
- **Proporcionar al lado cliente una documentación consistente acerca de la aplicación.** [REDACTED] se ha proporcionado al lado cliente una documentación consistente. En el caso de la API desarrollada con Spring, se ha proporcionado una documentación generada por Swagger. Por otra parte, en el caso de Python se ha generado la documentación gracias a Sphinx. En ambos casos, se ha cumplido el objetivo de proporcionar la documentación al lado cliente.
- **Incluir una mejora a la aplicación que le de cierto atractivo al proyecto.** Objetivo cumplido teniendo la API REST desarrollada en Python que contiene una mejora para la aplicación con relación a la Visión Artificial. Dicha mejora está basada en el reconocimiento facial de fotos que publica un determinado usuario. Una vez el usuario tiene cierta actividad en la aplicación, se es capaz de hacer al usuario sugerencias sobre quién aparece en sus publicaciones de entre sus amistades.
- **Entender en profundidad lo que supone desarrollar un software desde cero.** Como se dijo en el capítulo de Introducción, concretamente en la sección de Motivación, se pretende comprender lo que es el desarrollo de un software desde la fase de análisis hasta el despliegue de la aplicación. Este objetivo se ha cumplido como se puede comprobar en este documento, en el que se ha desarrollado un software que ha pasado por todas las fases de desarrollo de software

- **Tener un entorno en local para el desarrollo y un entorno de producción para probar lo desarrollo de cara al cliente.** En el caso de la API desarrollada en Spring, se utiliza Spring Tool Suite para el desarrollo en local y para el empaquetamiento de la aplicación con el objetivo de poner en producción la aplicación. Se tiene una Base de Datos que tendrá información relativa al desarrollo en local. Por otra parte, el entorno en producción se tiene un VPS el cual tiene instalado Apache Tomcat para el despliegue de la aplicación, aparte de una Base de Datos para almacenar la información en producción. En otro orden de cosas, en el caso de la API [REDACTED] se tiene un desarrollo en local en PyCharm usando una Base de Datos para el desarrollo en local. Al igual que la API en Spring, se despliega la aplicación en el VPS con ayuda del framework web Bottle para la puesta en producción. Además de esto, se utiliza CherryPy para que el servidor desplegado por Bottle sea capaz de atender mas de una petición concurrentemente.
- **Búsqueda y uso de herramientas que hagan posible la realización de los objetivos nombrados hasta ahora.** Como ya se [REDACTED] en el capítulo de Diseño e Implementación, concretamente en la sección de Herramientas utilizadas, se han usado varias herramientas para el correcto desarrollo de la aplicación. [REDACTED]
[REDACTED]
[REDACTED] Entre ellas se pueden destacar: Java, Spring, Python, OpenCV, MariaDB [REDACTED]

5.2. Futuros trabajos

Una vez se ha terminado el desarrollo de proyecto se han tenido una serie de consideraciones en relación a incluir una serie de funcionalidades en la aplicación en el futuro. Dichas funcionalidades se van a detallar a continuación.

En primer lugar, una mejora urgente que se debería [REDACTED] realizar en la aplicación es un aumento de los recursos en términos de GPU del servidor. La API desarrollada en Python hace un uso intensivo de la GPU del servidor para el reconocimiento facial, y ya se ha tenido que realizar una reducción de la resolución de las fotos para poder realizar el reconocimiento.

Seria una buena práctica incluir test de integración con el objetivo de comprobar con exactitud si la capa de los repositorios o DAOs junto con la capa de los servicios están debidamente integradas con la totalidad de la aplicación, ya que actualmente solo se tienen test unitarios comprobando la cobertura del código.

Haciendo hincapié en la API desarrollada en Spring, sería de una gran ayuda para el lado cliente que se incluyese la posibilidad de que, una vez registrado en el servidor, fuese posible realizar una confirmación vía email o mediante SMS. Además también se debería de incluir la funcionalidad referida a la recuperación de credenciales, lo cual está directamente relacionada con la funcionalidad que se ha mencionado anteriormente.

Dado que el proyecto está diseñado para ser usado desde una aplicación cliente de carácter social, una funcionalidad que estaría bien incluir sería la posibilidad de tener un servicio de mensajería instantánea. Con la posibilidad de que los usuarios fuesen capaces de tener

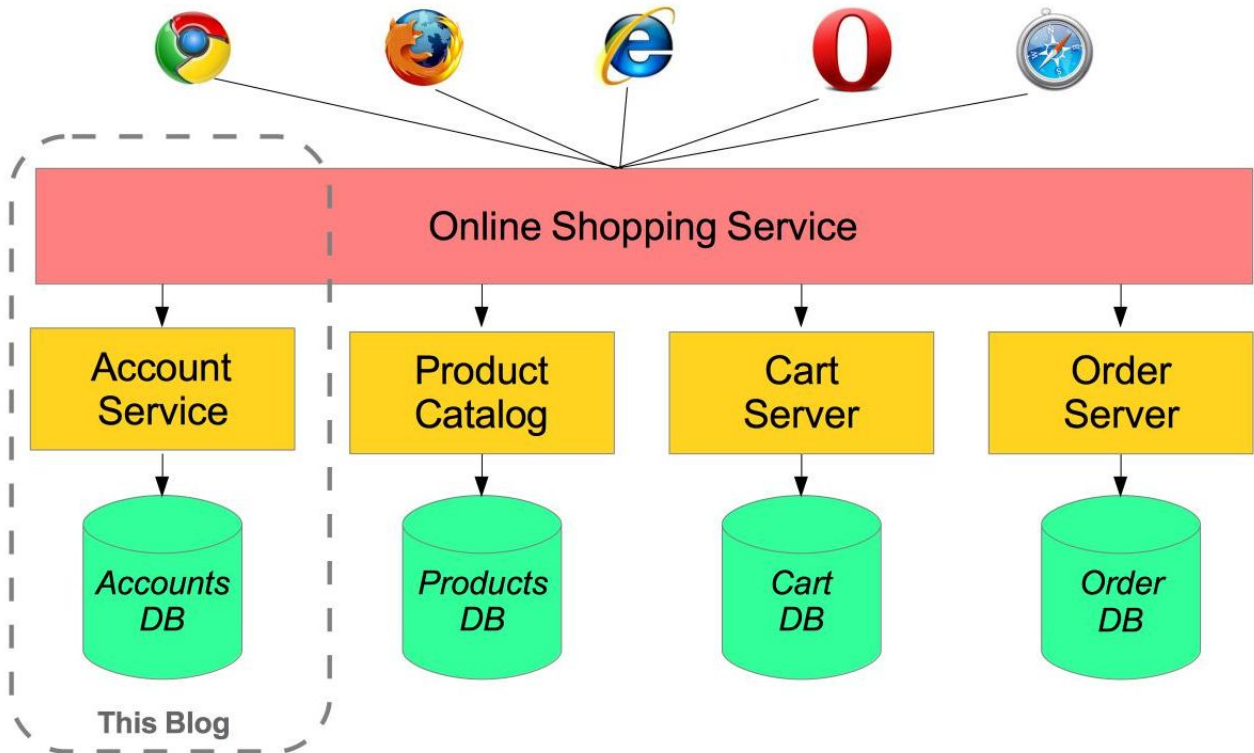


Figura 5.2.1: Microservicios en Spring

un canal privado para hablar unicamente con una persona o ser capaces de crear grupos donde poder hablar con varias personas a la vez.

Actualmente, se tiene todo el proyecto alojado en el mismo servidor, la Base de Datos usada para desarrollo y producción, la API de Python y de Spring están alojadas en el mismo servidor. Por cuestiones de seguridad, una buena mejora sería alojar las aplicaciones en distintos servidores.

Una vez más, hablando de la API desarrollada en Spring Framework, se puede incluir una mejora considerable a la aplicación. Actualmente, la API está construida como un monolito, donde conviven todos los controladores que se encargan de atender las peticiones relacionadas con los diferentes modelos de la aplicación (Usuarios, comentarios, publicaciones, imágenes y caras). Teniendo en cuenta esto, sería una buena práctica crear microservicios en el cada uno se encargue de atender las peticiones de cada modelo. Por lo que si esta funcionalidad se pusiese en practica, se tendrían 5 microservicios, uno por cada modelo de la aplicación, cada uno con su propia Base de Datos, sus repositorios y sus servicios. En la Figura 5.2.1 extraída de un blog de Paul Chapman, un consultor senior del equipo de Spring [5], se ve como el lado servidor tiene modularizado cada uno de los modelos, separandolos por microservicios sin que eso afecte al lado cliente, que va a seguir accediendo a los servicios que provee el lado servidor de la misma forma. De esta forma, si se incluyese esta mejora en la aplicación, aumentaría la escalabilidad y la mantenibilidad de la aplicación sin que el usuario se viese afectado.

Continuando con la funcionalidad anterior y en relación con la API desarrollada en Python. Actualmente, para poder acceder a esta API es necesario utilizar un endpoint diferente, esto significa que las APIs no están relacionadas entre sí de ninguna forma, simplemente



se cambia la ruta para acceder a una u otra. Una funcionalidad a incluir sería, utilizando los microservicios explicados en el apartado anterior, realizar las llamadas referentes a la API de Python desde la aplicación en Spring teniendo obviamente un microservicio dedicado a las llamadas referentes a la API de Python, de manera que todo fuese homogéneo de cara al lado cliente.

Bibliografía

1. *¿Qué es Spring Boot?* <https://www.arquitecturajava.com/que-es-spring-boot/>.
2. *API Gateway OAuth 2.0 Authentication Flows*
3. BBVAOPEN4U. *API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos* <https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos>.
4. *Bottle* <https://bottlepy.org/docs/dev/>.
5. Chapman, P. *Microservices with Spring. Engineering* (2015).
6. *Face Recognition* https://github.com/ageitgey/face_recognition.
7. *Flujo del token de soporte JWT de OAuth 2.0* https://help.salesforce.com/articleView?id=remoteaccess_oauth_jwt_flow.htm&type=5.
8. *GoDaddy* <https://es.godaddy.com/>.
9. *JWT Python Library* <https://pyjwt.readthedocs.io/en/latest/>.
10. *Let's Encrypt* Let's Encrypt. <https://letsencrypt.org/>.
11. *Maven vs Gradle* <https://gradle.org/maven-vs-gradle/>.
12. *OpenCV 3.4.1* <https://opencv.org/opencv-3-4-1.html>.
13. *OVH* <https://www.ovh.es/>.
14. *Python MySQL Connector* <https://dev.mysql.com/downloads/connector/python/>.
15. *Servidores VPS y dedicados* <https://es.godaddy.com/help/que-es-un-servidor-virtual-privado-154>.
16. *Spring Boot* <https://spring.io/projects/spring-boot>.
17. *Spring Data* <https://spring.io/projects/spring-data>.
18. *Spring Framework* <https://spring.io/>.
19. *Spring MVC* <https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html>.
20. *Springfox Swagger UI* <https://springfox.github.io/springfox/docs/current/>.



Índice alfabético