Automatización de la recogida de firmas para iniciativas ciudadanas

VNiVERSiDAD D SALAMANCA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Julio de 2016

Autor

Aythami Estévez Olivas

Tutor

Rodrigo Santamaría Vicente

Declaración de autoría

Yo, Aythami Estévez Olivas, declaro que este Trabajo de Fin de Grado que lleva por título *"Automatización de la recogida de firmas para iniciativas ciudadanas"* ha sido realizado por mi persona, bajo el tutelaje de Rodrigo Santamaría Vicente, dentro del marco de mis estudios de Grado en Ingeniería Informática en la Universidad de Salamanca. Confirmando que aquellas partes que hayan sido fruto del trabajo de otros han sido debidamente citadas.

En Salamanca, a ___ de julio de 2016

Firma del alumno: Firma del tutor:

Aythami Estévez Olivas Rodrigo Santamaría Vicente

Resumen

Los procesos de recogida de firmas con identificación (con DNI) requieren un enorme esfuerzo por parte de los organizadores en términos logísticos, económicos o de recursos humanos. Se requiere gente para recoger las firmas, para clasificarlas,… Esfuerzos que con frecuencia, no llegan a ningún puerto por no alcanzar los apoyos necesarios. Esto principalmente se debe a los ajustados plazos para recolectar las firmas, a problemas organizativos, a falta de conocimiento sobre estos procesos, etc.

El objetivo de este proyecto es el de automatizar, en la medida de lo posible, los procesos de recogida de firmas, eliminando la carga de clasificar esas identificaciones y ofreciendo una digitalización de los datos de esa firma que permita contrastar con las firmas manuscritas para minimizar las firmas perdidas por falta de legibilidad.

Para ello se ha desarrollado una aplicación para dispositivos móviles que puede ser usada por los organizadores de las recogidas de firmas de modo sencillo y les permite centrarse en lo que de verdad importa, conseguir las firmas que buscan.

Complementariamente se ha desarrollado un servidor donde se concentra la funcionalidad del sistema y se almacenan todos los datos asegurando los requisitos necesarios de privacidad y destrucción posterior de la información. Dicha funcionalidad emplea técnicas de visión artificial o reconocimiento óptico de caracteres para lograr la clasificación de identificaciones así como algoritmos criptográficos y protocolos seguros de primer nivel para asegurar la privacidad.

Este proyecto se ha desarrollado siguiendo los preceptos del software libre, lo que permite a cualquier persona usar, modificar o distribuir este proyecto haciendo que cualquiera que lo necesite pueda utilizarlo.

Palabras clave:

Recogida de firmas, DNI, Rest, Android, Java

Overview

The signature events with identification (DNI) require an enormous effort by the organizing committee in logistical, economic or human resources terms. It require people to retrieve the signs, to classify them,… This efforts, frecuently, are all in vain for not reach the necessary support. This is mainly because of the tight deadlines to recolect the signs, due to organizational problems and/or owing to the lack of knowledge about this events.

The goal of this project is to automate, to the extent possible, the signature events, removing the charge of classify this ID and offering a data digitalization of the sign that allow to compare with the handwritten sign in order to minimize the loss of sign due to the lack of legibility.

To achieve that goal a mobile device application has been developed, the committee could use it in a simply way and allow them to center in the real topic, get the required signs.

Supplementary a server which concentrate the functionality and stores, in a secure way, all the data has been developed. That functionality involves computer vision techniques and optical character recognition algorithms to achieve the classification of the IDs, to guarantee the privacy cryptographic algorithms and secure protocols has been used.

The project has been developed following the precepts of the open source software, which allows the use, redistribution or modification of this software to any person who need it.

Key words:

Signature event, DNI, Rest, Android, Java.

Índice de contenidos

Índice de figuras

Índice de tablas

Lista de abreviaturas

- **DNI:** Documento Nacional de Identidad.
- **ILP:** Iniciativa Legislativa Popular.
- **TFG:** Trabajo de Fin de Grado.
- **SO:** Sistema Operativo.
- **OpenCV:** Open Source Computer Vision.
- **OCR:** Optical Character Recognition.
- **REST:** REpresentational State Transfer.
- **HTTP:** HiperText Transfer Protocol.
- **URI:** Uniform Resource Identifier.
- **URL:** Uniform Resource Locator.
- **XML:** eXtensible Markup Language.
- **JSON:** JavaScript Object Notation.
- **HTML:** HyperText Markup Language.
- **IDE:** Integrated Development Enviroment.
- **API:** Application Programming Interface.
- **MIME:** Multipurpose Internet Mail Extensions.
- **TLS:** Transport Layer Security.
- **JSSE:** Java Secure Sockets Extension.
- **CA:** Certification Authority.
- **USAL:** Unversidad de Salamanca
- **OWASP:** Open Web Application Security Project.
- **JCE:** Java Cryptographic Extensions.
- **AES:** Advanced Encryption Standard
- **PDF:** Portable Document Format

1. Introducción

En España existen distintos mecanismos de democracia participativa que requieren la recogida de firmas con identificación de los firmantes, esta identificación se realiza mediante el Documento Nacional de Identidad[1] o DNI. Algunos ejemplos de estos mecanismos son las Iniciativas Legislativas Populares (ILPs), las cuales se refieren a la posibilidad de que cualquier ciudadano pueda presentar iniciativas de ley sin ser representante popular en los órganos legislativos estatales si presenta un número determinado de apoyos (500.000 firmas si es a nivel nacional). Dichas ILPs están regulada a nivel nacional y provincial en la Constitución (Art. 87.3[2]) y en diversas leyes orgánicas. Otro ejemplo de mecanismo es el apoyo a Agrupaciones de Electores[3] no asociadas a partidos políticos, para poder presentar una candidatura a unas determinadas elecciones se requiere un número mínimo de firmas del censo electoral de las respectivas elecciones, el número de firmas necesario varía en función de las elecciones a las que se presenten así como del tamaño del censo, siendo en general un 1% del censo.

En cualquiera de los sistemas presentados previamente se requiere tomar ciertos datos del firmante, generalmente utilizando hojas en las que cada firmante anota a mano los datos necesarios, así como su DNI, ya sea fotocopiándolo mientras el firmante rellena sus datos o bien mediante una fotografía aportada por este (por las dos caras). Teniendo en cuenta el elevado número de firmas que se pueden llegar a tener que recoger así como la gestión de la privacidad de los datos de acuerdo a la legalidad vigente y los problemas de legibilidad de los datos escritos, la gestión de una recogida de firmas puede suponer un reto. Si además añadimos que los plazos de recogida y presentación de las firmas suelen ser bastante ajustados sería muy útil la automatización de este proceso o parte de él. Para justificar la dificultad de estos procesos basta con citar que a fecha de 2012 solo se ha aprobado una ley mediante ILP en 30 años y 66 propuestas distintas.[4]

Lo que busca el presente TFG es automatizar estos procesos de recogida de firmas. La automatización total del proceso no se contempla, ya que en la sociedad actual existe aún cierta desconfianza en el uso de las nuevas tecnologías para asuntos trascendentes como puede ser la firma electrónica o las compras on-line. Este tipo de iniciativas suelen ir ligadas al contacto con la gente, a que un desconocido te pare por la calle y te convenza con sus argumentos por ello la recogida de firmas en la calle con datos manuscritos continuará siendo así. Lo que se plantea aquí es una herramienta que pueda ayudar a las comisiones encargadas de recoger las firmas facilitando el proceso de almacenar y clasificar los DNI de las firmas recogidas con el objetivo de ahorrar tiempo, recursos e intentar mejorar los cauces de participación ciudadana.

1.1. Contenido y estructura de la memoria

El presente documento se organiza en los siguientes apartados siguiendo las recomendaciones de (García Peñalvo & Moreno García, 2000):

[1] http://www.interior.gob.es/web/servicios-al-ciudadano/dni/concepto-y-validez
[2] http://www.congreso.es/consti/constitucion/indice/sinopsis/sinopsis.jsp?art=87&tipo=2
[3] https://es.wikipedia.org/wiki/Agrupaci%C3%B3n_de_electores
[4] http://www.abc.es/20120606/espana/abci-iniciativa-legislativa-popular-201206052148.html

- **Objetivos:** donde se explica de forma concreta los distintos objetivos perseguidos con este proyecto. Aparecen clasificados en tres subgrupos,
    - *Objetivos funcionales*: especificación de las funcionalidades que ha tener el sistema.
    - *Objetivos no funcionales*: aquellos no relacionados directamente con la funcionalidad como restricciones que ha de cumplir el proyecto o cualidades externas que ha de exhibir.
    - *Objetivos personales*: los perseguidos por el alumno con la realización del proyecto.
- **Conceptos teóricos:** en este apartado se aclararán algunos aspectos teóricos necesarios para comprender el desarrollo del proyecto.
- **Técnicas y Herramientas:** esta parte de la memoria tiene como objetivo presentar las técnicas y herramientas utilizadas a lo largo del desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** aquí se recogen los aspectos más interesantes del desarrollo del sistema articulados entorno al modelo de proceso utilizado en este proyecto.
- **Trabajos relacionados:** este apartado comenta los aspectos principales de algunos proyectos similares al aquí desarrollado, realizando una comparativa final entre dichos proyectos.
- **Conclusiones y líneas de trabajo futuras:** como conclusión de esta memoria se hacen algunas reflexiones sobre los resultados obtenidos así como algunas propuestas de mejora en el futuro.
- **Bibliografía:** que contiene las referencias a artículos o documentos utilizados además de los distintos enlaces web que se encuentran repartidos por el documento.
- **Apéndice A: Fondo de fotografía recomendado:** este último apartado muestra la plantilla diseñada como fondo sobre el que hacer las fotografías de los DNI. Se recomienda su uso para asegurar unos mejores resultados en el funcionamiento y rendimiento del sistema.

Adicionalmente a este documento se pueden encontrar en formato electrónico los siguientes anexos:

- **Anexo I: Plan del Proyecto Software:** en el que se incluye una planificación temporal detallada.
- **Anexo II: Especificación de Requisitos del Software:** donde se detallan los requisitos del sistema usando representaciones textuales y diagramática.
- **Anexo III: Modelo de Análisis:** que contendrá la especificación del dominio del problema.
- **Anexo IV: Modelo de Diseño:** que contendrá la especificación del dominio de la solución.
- **Anexo V: Documentación Técnica:** este anexo contendrá la documentación para el programador.
- **Anexo VI: Manual de Despliegue:** con las instrucciones necesarias para la instalación y despliegue del servidor.

2. Objetivos

El objetivo principal de este proyecto es desarrollar un sistema distribuido siguiendo un modelo cliente-servidor en el que los clientes sean dispositivos móviles a través de los cuales se realicen fotografías de los DNI y dichas fotografías sean enviadas al servidor en el que se procesarán, clasificarán y almacenarán convenientemente para que puedan ser recuperadas posteriormente.

2.1. Objetivos funcionales

El objetivo principal se puede descomponer en los siguientes objetivos funcionales:

2.1.1. Obtención de imágenes de DNI

Obtención de imágenes de DNI en buena calidad e integración/asociación con datos personales relacionados (Nombre y Apellidos, Nº de identificación). Para ello se usarán técnicas de visión artificial.

2.1.2. Almacenamiento de información

Almacenamiento de las imágenes tomadas así como la información sobre campañas o firmas de modo transparente para el usuario.

2.1.3. Digitalización de datos

Los datos personales a menudo se recogen mediante hojas escritas, por falta de recursos o desconocimiento del uso de medios digitales. Se estudiará la posibilidad de realizar reconocimiento óptico de caracteres sobre las imágenes de los DNI para recuperar datos del mismo tales como nombre, apellidos o número de DNI.

2.1.4. Gestión de campañas

Han de ofrecerse mecanismos para poder dar de alta y baja campañas de recogida de firmas de manera sencilla.

2.1.5. Recuperación de firmas

El sistema tiene que proporcionar un mecanismo para recuperar todos los DNIs asociados a firmas almacenadas de una campaña de cara a la presentación de estas, se buscará optimizar el número de firmas por página con el objetivo de minimizar el gasto de papel que puede ser muy significativo, por ejemplo en los casos tradicionales de presentación de firmas que se fotocopien los DNI se gasta 1 folio por cada DNI.

2.2. Objetivos no funcionales

Además de los objetivos funcionales existen otros externos a la funcionalidad, pero que son necesarios para la viabilidad del proyecto como:

2.2.1. Simplicidad

La herramienta desarrollada ha de ser muy sencilla de utilizar y abstraer del cliente móvil toda la complejidad y procesamiento que conlleva el sistema. Ya que el objetivo de este es automatizar los procesos de recogida de firmas quitándole la mayor carga cognitiva posible al usuario.

2.2.2. Seguridad

El almacenamiento debe asegurar los requisitos necesarios de privacidad, y destrucción posterior de la información teniendo en cuenta el carácter especialmente sensible de la información manejada por el sistema.

2.2.3. Plataforma

Aunque la plataforma del cliente es libre, esta, ha de ser ejecutable en un cliente móvil por lo que se estima que Android sería la mejor opción debido al gran número de dispositivos que cuentan con este SO en nuestro país, pero sin descartar la opción de aplicación web o versiones iOS.

2.2.4. Rendimiento

Será necesario ofrecer una buena experiencia de usuario minimizando en la medida de lo posible los tiempos de espera en el cliente.

2.3. Objetivos personales

Los objetivos personales buscados con este proyecto se pueden resumir en:

- Experimentar el desarrollo de un proyecto real empleando los conocimientos adquiridos a lo largo de la titulación.
- Realizar un proyecto de software libre que, con suerte, pueda ser aprovechado por alguien.
- Profundizar y ampliar los conocimientos que se tienen mediante la resolución de autónoma de los problemas que vayan surgiendo a lo largo del desarrollo.

3. Conceptos teóricos

En este apartado se aclararán algunos conceptos teóricos que pueden ser necesarios para entender ciertos aspectos del proyecto, En concreto se hablará de la visión artificial y alguna de sus técnicas empleadas, del reconocimiento óptico de caracteres, de la arquitectura REST para el diseño de servicios web, de algunos conceptos básicos sobre el desarrollo de aplicaciones en Android y de Scrum, el modelo de proceso empleado.

3.1. Visión artificial

Una de las partes sustanciales del proyecto es la de visión por computador, ya que se emplean técnicas de esta rama para detectar los DNI de las fotografías así como hacer diversos procesos de corrección de imagen como la detección de ángulo y la corrección de este, recortes de imagen o cierto pretratamiento antes de pasar las imágenes por él-sistema de detección de caracteres.

Podemos definir la visión artificial como:

> *Ciencia que desarrolla la base teórica y algorítmica mediante la que se extrae y analiza información útil sobre el mundo a partir de imágenes aisladas, secuencias o conjuntos de estas*[5].

En pocas palabras se basa en la aplicación de algoritmos, como filtros, a imágenes con el objetivo de extraer cierta información de ellas. Esto que para los seres vivos es tan natural y automático supone un desafío para un computador. Cabe destacar que no existe la perfección en estos procesos y pequeños cambios en la iluminación por ejemplo pueden alterar los resultados de modos imprevistos, como yo mismo he podido comprobar con la realización de este proyecto. Otra definición de visión artificial que recoge ese componente de dificultad es:

> *In computer vision, we are trying to […] describe the world that we see in one or more images and to reconstruct its properties, such as shape, illumination, and color distributions. It is amazing that humans and animals do this so effortlessly, while computer vision algorithms are so error prone. (Szeliski, 2010)*

Concretando un poco más en los procesos de visión artificial más importantes empleados para el proyecto:

3.1.1. Umbralización[6]

La umbralización de una imagen o *threshold* como se conoce en inglés es el método de segmentación más simple. Lo explicaré con el siguiente ejemplo: las diferentes regiones de una imagen corresponden con objetos, los cuales se quieren analizar. La

[5] Presentación "Robótica: Visión artificial", Universidad de Salamanca (2014)
[6] http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html

separación entre regiones de una imagen está basada en la variación de intensidad entre los pixeles del objeto y los del fondo.

ILUSTRACIÓN 1: EJEMPLO DE UMBRALIZACIÓN

Para diferenciar los pixeles en los que estamos interesados del resto, se realiza una comparación del valor de intensidad de cada pixel respecto a un *threshold* (umbral) determinado en función del problema que se quiera resolver. Una vez separados de forma adecuada los pixeles importantes podemos fijarles un determinado valor para identificarlos (como 0 (negro) o 255 (blanco) en una imagen en blanco y negro).

Existen varios tipos distintos de *threshold*, en el proyecto se ha empleado el *threshold* binario inverso el cual funciona de la siguiente manera. Sea la siguiente gráfica que representa la intensidad de los pixeles y la línea azul el valor thresh.

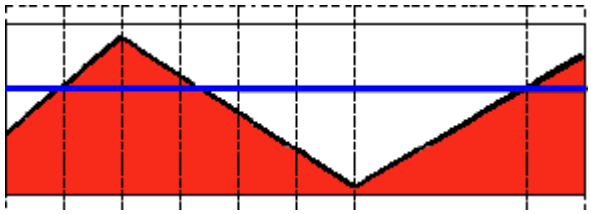

ILUSTRACIÓN 2: UMBRALIZACIÓN: GRÁFICA BASE

El thershold binario se podría expresar como:

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxVal} & \text{otherwise} \end{cases}$$

ILUSTRACIÓN 3: FUNCIÓN THRESHOLD BINARIO INVERSO

Es decir, si la intensidad del pixel **`src(x, y)`** es mayor que el **`thresh`** entonces la intensidad del nuevo pixel es **`0`**, en caso contrario **`maxVal`**. Por tanto la gráfica quedaría de la siguiente manera:

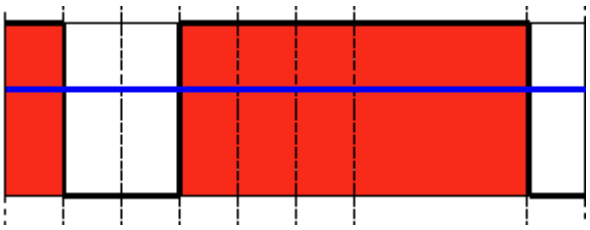

ILUSTRACIÓN 4: UMBRALIZACIÓN: GRÁFICA THRESHOLD BINARIO INVERSO

En el caso concreto de este sistema la fotografía, que idealmente se hace sobre un fondo blanco (un folio con el contorno de un DNI) se pasa se pasa a escala de grises y tras esto, al aplicar una umbralización binaria inversa el fondo blanco pasa a ser negro y el DNI blanco con lo cual se favorece su detección. En la siguiente imagen se puede apreciar un ejemplo de umbralización binaria inversa.

ILUSTRACIÓN 5: EJEMPLO UMBRALIZACIÓN BINARIA INVERSA

3.1.2. Búsqueda de contornos[7]

La imagen binarizada procedente de la umbralización anterior pasa por un algoritmo de detección de contornos (Suzuki & Abe, 1985) que busca detectar el cuadrado blanco en el que se habrá convertido el DNI si todo ha ido bien.

Entendemos como contorno una curva que une todos los puntos continuos a lo largo de un borde teniendo estos el mismo color o intensidad. Los contornos son útiles para la visión artificial en análisis de forma, detección de objetos y reconocimiento.

Teniendo en cuenta la definición previa resulta obvio que será mucho más fácil detectar contornos en una imagen binarizada en la que solo haya pixeles blancos o negros. Por ejemplo la implementación de *OpenCV* (una de las librerías de funciones para visión artificial más utilizadas) de este algoritmo "busca" objetos blancos sobre fondos negros.

Una vez detectados todos los contornos de la imagen habría que filtrarlos por algún criterio como el contorno de mayor área o el que tenga una relación de aspecto (proporción entre su anchura y altura) coincidente con la del objeto que buscamos.

En la siguiente imagen podemos apreciar el resultado de aplicar la búsqueda de contornos a la Ilustración 5 y seleccionar aquél con mayor área, pintado en la imagen de color azul.

[7] http://www.docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html#gsc.tab=0

ILUSTRACIÓN 6: EJEMPLO DETECCIÓN DE CONTORNOS

3.2. Reconocimiento óptico de caracteres

Otra de las partes sustanciales del motor del sistema es el reconocimiento óptico de caracteres, el cual se emplea sobre los DNI recortados para extraer ciertos datos que se detallaran posteriormente.

Se puede definir el reconocimiento óptico de caracteres (Optical Character Recognition, OCR) como la conversión mecánica o electrónica de imágenes de texto impreso o escrito a mano a texto que una máquina puede entender[8]. El reconocimiento de texto impreso resulta bastante más sencillo, como puede deducirse, debido a las fuentes de texto fijas, tamaños de línea constantes…Es una técnica íntimamente relacionada con la visión artificial y emplea numerosas técnicas de esta como la umbralización binaria (véase 3.1.1.).

3.2.1. Caso de estudio: Tesseract

Para explicar cómo funciona un proceso de OCR real me basaré en *Tesseract*[9] (Smith, 2007), (Smith, 2007b) un motor OCR que empezó a ser desarrollado por HP en 1984 y actualmente es software libre desarrollado por Google. Debido a su larga trayectoria y a su licencia de software libre está bastante documentado y su código fuente se puede encontrar en *GitHub*. El procesamiento que sigue *Tesseract* es un pipeline paso a paso. Antes de entrar al procesamiento de OCR propiamente dicho realiza una umbralización adaptativa que a partir de imágenes en escala de grises o en color genera una imagen binaria en la cual puede detectar texto negro sobre blanco o viceversa.

[8] https://en.wikipedia.org/wiki/Optical_character_recognition
[9] https://github.com/tesseract-ocr

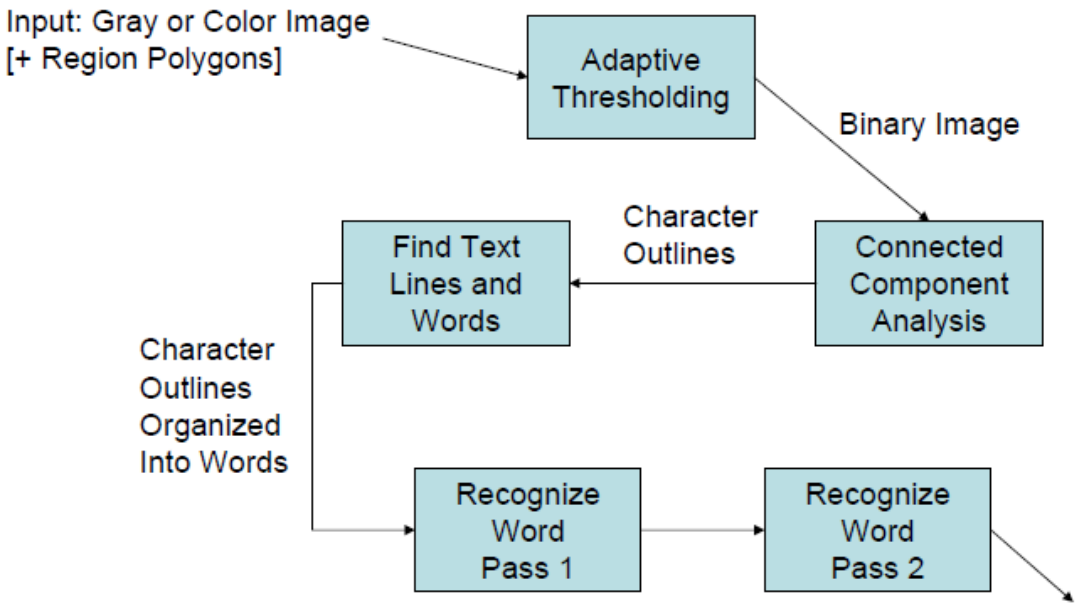

ILUSTRACIÓN 7: TESSERACT: ARQUITECTURA (SMITH, 2007B)

El primer paso está relacionado con el **análisis de componentes**, en el cual los contornos de los componentes (letras) se almacenan para su identificación posterior, dichos contornos sufren una aproximación poligonal con la que se elimina ruido pero también información.

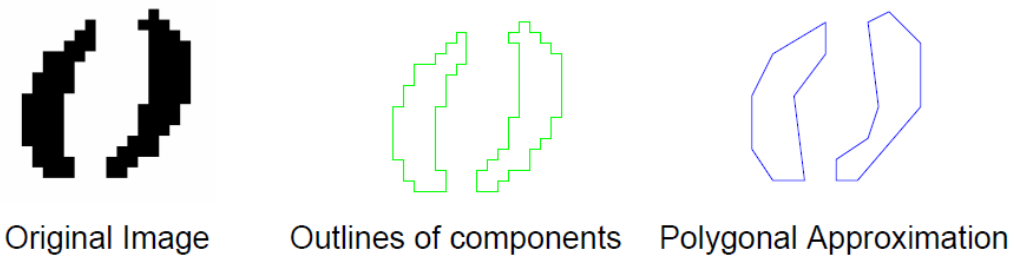

ILUSTRACIÓN 8: TESSERACT: ANÁLISIS DE COMPONENTES (SMITH, 2007B)

A continuación se ejecuta un algoritmo que **busca líneas,** diseñado para encontrar líneas aunque la imagen de origen esté girada. Este se basa en un filtrado de contornos por su altura. Se calcula la mediana de la altura en una región de texto y en función de esto permite eliminar los contornos menores que cierta fracción de dicha mediana, ya que entiende que son signos de puntuación o ruido, así como separar los caracteres que se toquen verticalmente, ya que ordena y procesa los contornos por la coordenada X y hace que se pueda asignar cada uno a una única línea de texto. Finalmente se fusionan los contornos que se superpongan horizontalmente uniendo con ello las marcas diacríticas con sus letras y asociando partes de algunos caracteres "rotos".

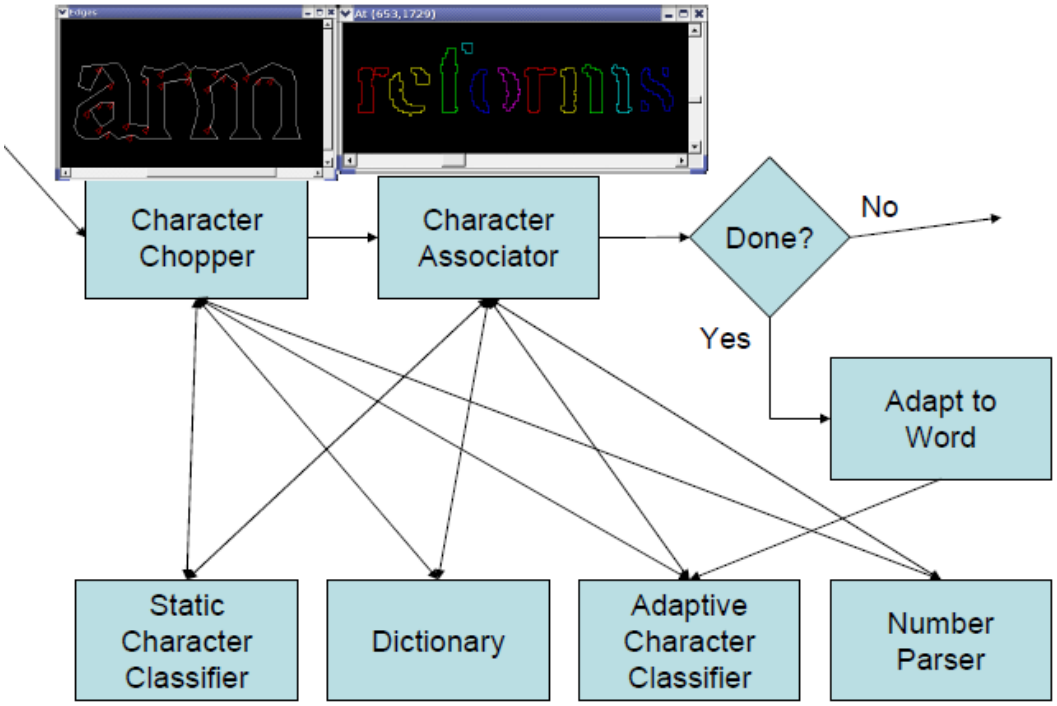

ILUSTRACIÓN 9: TESSERACT: MÓDULO DE RECONOCIMIENTO DE PALABRAS (SMITH, 2007B)

Tras encontrar las líneas se busca **reconocer las palabras** que las componen, inicialmente se determina si la línea está escrita con una tipografía de espaciado fijo, en cuyo caso divide las palabras en letras asignando ese espacio fijo a cada una. En caso contrario el proceso de **separación de las palabras en letras** se complica.

Mientras el resultado de esta separación sea insatisfactorio, *Tesseract* busca mejorarlo. Si la palabra sigue sin ser adecuada se pasa a un **asociador de caracteres** el cual busca en un grafo de segmentación utilizando una estrategia A* (la mejor solución primero) las combinaciones que maximicen la aproximación de contornos a caracteres candidatos. Tanto el separador de caracteres como el asociador se emplean de diversas herramientas para llevar a cabo su función como es un **clasificador de caracteres estático** el cual es entrenado mediante un **conjunto de entrenamiento** formado por 20 muestras de 94 caracteres en 8 fuentes variando la negrita y la cursiva.

Cada vez que el módulo de reconocimiento de palabras está considerando una segmentación, el **módulo lingüístico** elige la palabra que mejor coincida en diferentes categorías: palabra más frecuente, palabra más parecida del diccionario, "palabra" numérica más similar, palabra en mayúsculas más similar… La decisión final se toma seleccionando la palabra con la menor distancia ponderada en función de la categoría a la segmentación que se está evaluando.

Por último el módulo de reconocimiento de palabras emplea también un **clasificador adaptativo** muy similar al estático pero este tiene en cuenta la fuente concreta y es entrenado con la salida del clasificador estático. Debido a este clasificador adaptativo el proceso de detección de palabras se ejecuta dos veces para aprovecharse de lo que haya podido aprender este clasificador.

3.3. REST

REST o Representational State Transfer (Fielding, 2000)[10] es un estilo arquitectónico para el desarrollo de aplicaciones web enfocado a datos en lugar de a detalles de implementación. Tiene como propósito inducir rendimiento, escalabilidad, simplicidad, modificabilidad, visibilidad, portabilidad y fiabilidad. Es el estilo arquitectónico de la World Wide Web (WWW).

REST también es el estilo arquitectónico que se ha utilizado para este proyecto para desarrollar el servicio web.

Se fundamenta en el protocolo HTTP para la comunicación y en las URI como método de direccionamiento y localización de recursos. REST distingue entre recurso y representación, un recurso es una fuente de información específica referenciada por un identificador global (URI), mientras que una representación de un recurso es la forma en la que se intercambia dicho recurso.

Un servicio web RESTful es aquel que cumple las siguientes restricciones:

- La URI del servicio es de la forma http://example.com/resources/
- Los datos están en un formato estándar como XML o JSON.
- Solo hay operaciones HTTP: GET, PUT, POST, DELETE.
- API accesible a través de un navegador (uso de hipertexto).
- No mantiene estado del cliente en el servidor, cada petición de un cliente contiene toda la información necesaria para ejecutar la petición.

3.4. Android

Android es un sistema operativo móvil, actualmente desarrollado por Google, basado en núcleo de Linux y diseñado principalmente para dispositivos con pantallas táctiles como móviles o tablets[11]. Es el SO móvil con mayor cuota de mercado mundial[12], llegando en 2014 a estar presente en 81,5% de los dispositivos, esto ha hecho que sea el entorno de desarrollo móvil elegido para este proyecto.

Soporta Java como lenguaje de programación de forma nativa, pero con alguna particularidad, no se basa en un método *main()* como lo hace Java, si no que se ejecuta por medio de instancias de **Actividades[13].** Una actividad es un componente de una aplicación que provee una pantalla con la que los usuarios pueden interactuar para realizar alguna tarea. Durante el ciclo de vida de una actividad, esta puede pasar por varios estados, cada uno de ellos con métodos concretos que se ejecutan al entrar en dicho estado. Es responsabilidad del programador controlar esos cambios de estado para asegurarse de no consumir recursos del sistema si ya no se está usando de forma activa la actividad o no perder el progreso del usuario si abandona momentáneamente la

[10] https://en.wikipedia.org/wiki/Representational_state_transfer
[11] https://en.wikipedia.org/wiki/Android_(operating_system)
[12] http://www.xatakandroid.com/sistema-operativo/android-consolido-su-liderazgo-en-2014-alcanzando-el-81-5-de-cuota-de-mercado
[13] https://developer.android.com/guide/components/activities.html

aplicación. En el siguiente diagrama se pueden observar los estados así como los métodos invocados para los cambios de estados (en las flechas).

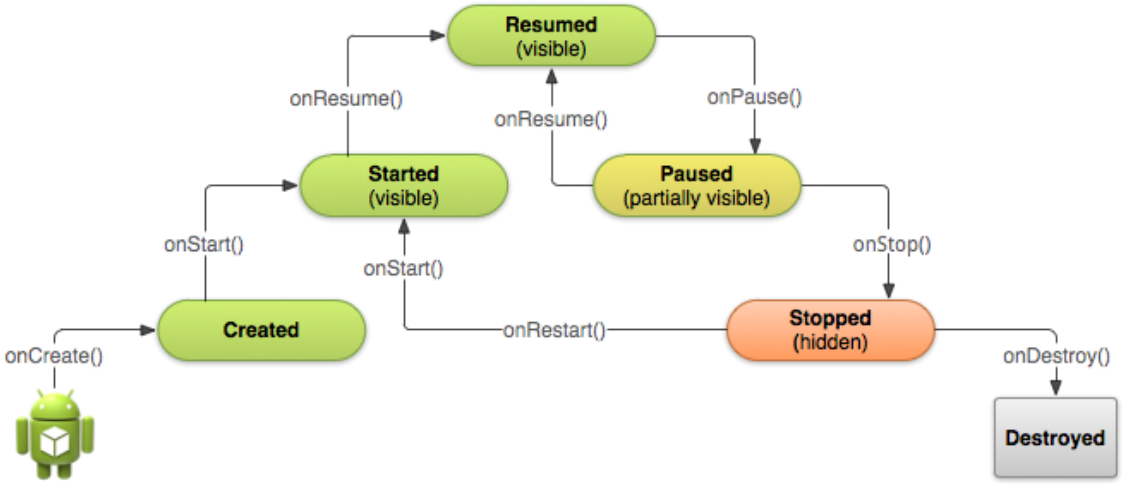

ILUSTRACIÓN 10: CICLO DE VIDA DE UNA ACTIVIDAD ANDROID[14]

Una de las principales características de Android es la habilidad de una aplicación de derivar al usuario hacia otra aplicación de acuerdo a la acción que desee realizar. Por ejemplo si una aplicación necesita realizar fotografías no es necesario crear una actividad en la aplicación que haga fotografías, en su lugar se puede crear una petición para realizar una fotografía mediante el uso de un **Intent**[15], el propio sistema inicia una aplicación capaz de realizar la tarea requerida, tomar una fotografía en este ejemplo.

3.5. Modelo de proceso software: Scrum

Podemos definir un **modelo de proceso** de software como una representación abstracta de un proceso software el cuál a su vez es el conjunto de actividades y resultados asociados necesarios para producir un producto software (Sommerville, 2005), es decir el modo de trabajo para el desarrollo de software. El modelo **Scrum** en concreto pertenece a la familia de **procesos ágiles** los cuales se caracterizan por (García Peñalvo, 2014):

- Poner menos énfasis en el análisis, diseño y documentación, priorizando la implementación.
- Tener equipos de desarrollo pequeños.
- Seguir un desarrollo incremental.
- Ser adecuados para entornos con requisitos cambiantes o poco definidos.

En Scrum[16] estos principios se cumplen gracias a que para desarrollar un producto software divide su desarrollo en periodos cortos de tiempo llamados **sprints** los cuales

[14] https://developer.android.com/training/basics/activity-lifecycle/starting.html
[15] https://developer.android.com/training/basics/intents/sending.html
[16] Scrum Alliance, Core Scrum, (2014) https://www.scrumalliance.org/scrum/media/ScrumAllianceMedia/Files%20and%20PDFs/Learn%20About%20Scrum/Core-Scrum.pdf

tienen una longitud definida (normalmente entre 1-4 semanas) y durante cada uno se construye y entrega un incremento del producto. Scrum define una serie de **stakeholders** o roles:

- Cliente o **product owner**: es el encargado de maximizar el valor del trabajo que realiza el equipo, mantiene la visión general del proyecto y asegura que se esté construyendo el producto requerido. Algunas de sus funciones son decidir que tareas realizar en cada sprint o cuales son los artefactos a producir en cada uno.
- **Scrum master**: es el líder del equipo, encargado de organizar al equipo de desarrollo para que sean productivos. Mantiene reuniones periódicas con el equipo durante los sprints para comprobar el trabajo realizado, verificando si se están cumpliendo los objetivos planificados y en caso de que no, replanificar para lograrlo.
- Equipo de desarrollo o **development team**: Es el conjunto de personas encargadas de desarrollar el producto, suele ser un equipo multidisciplinar de profesionales quien, entre todos, tiene las habilidades necesarias para desarrollar cada incremento.

Cada sprint produce una serie de artefactos y lleva a cabo una serie de actividades. Todos empiezan con una lista de tareas a realizar o **product backlog** priorizada por el cliente, dichas tareas pueden ser extensas y generales por lo que el equipo de desarrollo en conjunto con el cliente, puede descomponerlas y reordenarlas en lo que se conoce como refinamiento del "retraso" o **product backlog refinement**. Con esta lista de tareas se **planea el sprint**, se prevé las tareas que se realizarán en este sprint (que no tienen por qué ser todas) y se planifica su realización. Tras esto comienza el sprint, durante el cual el equipo se auto-organiza para producir el incremento acordado en la planificación, habitualmente se realizan reuniones internas diarias o **daily scrum** con el objetivo de asegurar que se esté cumpliendo la planificación. Al final de cada sprint se produce un **incremento del producto** que debe de ser una implementación de las tareas acordadas en la planificación. Sobre este incremento se produce una **revisión** con el cliente en el que se comprueba si se ha cumplido con lo esperado y este actualiza el backlog como considere para el siguiente sprint. Además de esta revisión también se produce una **retrospectiva** en la que se analiza cómo ha funcionado el sprint, las cosas que han ido bien y las que no, con el objetivo de identificar problemas e intentar remediarlos en siguientes sprints. En el siguiente diagrama se puede contemplar el proceso entero de un sprint.

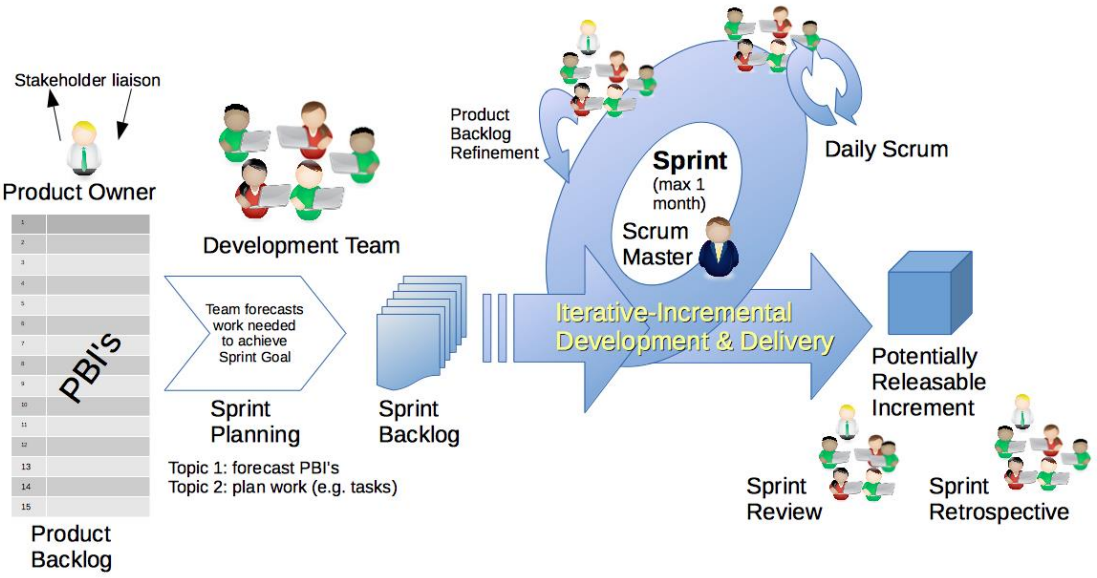

ILUSTRACIÓN 11: SCRUM DIAGRAMA[17]

[17] By Dr ian mitchell - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=44894952

4. Técnicas y Herramientas

En el desarrollo de este proyecto se han utilizado más de 20 técnicas y herramientas, distintas. A continuación se destacan las más destacadas agrupadas por tipos:

4.1. Lenguajes de programación y herramientas de desarrollo: Java y Android SDK

El lenguaje de programación usado para el desarrollo del servidor y la aplicación ha sido **Java**[18] debido a que es el lenguaje de programación de uso más extendido en el mundo[19][20], a la experiencia previa con él y al estar soportado de forma nativa en Android. En concreto se ha usado la versión Java SE 8 para el servidor pero en no para la aplicación debido a que esta versión aun no es soportada por Android.

Android SDK[21] es un conjunto de herramientas de desarrollo escrito en Java, que contiene todas las utilidades necesarias para el desarrollo de aplicaciones en Android. Ha sido utilizado en su versión 23.0.2 para crear la aplicación Android, la cual es compatible con dispositivos Android 2.3.4 o superiores.

4.2. Entornos de Desarrollo Integrado: Eclipse y Android Studio

Para desarrollar el servidor se ha usado **Eclipse**[22] como IDE en su versión Mars 4.5 que se encuentra bajo la licencia de software libre Eclipse Public License 1.0, mientras que para la aplicación Android se ha empleado **Android Studio**[23] que tiene una licencia Apache 2.0, aunque se inició el desarrollo en la versión 1.5.1 se ha finalizado en la versión 2.1.2.

4.3. Servidor de aplicaciones web: Apache Tomcat

Para ejecutar el servlet (la parte del servidor del proyecto) se ha elegido **Apache Tomcat**[24] en su versión 8.0.14 ya que se puede descargar en Debian vía apt. Tomcat es una implementación de diversas especificaciones de Java EE mantenida por Apache Software Foundation[25] bajo una licencia Apache 2.0.

[18] http://docs.oracle.com/javase/8/docs/
[19] http://spectrum.ieee.org/computing/software/top-10-programming-languages/
[20] http://www.tiobe.com/index.php/tiobe_index
[21] https://developer.android.com/studio/releases/build-tools.html
[22] https://www.eclipse.org/
[23] https://developer.android.com/studio/intro/index.html
[24] http://tomcat.apache.org/
[25] https://www.apache.org/

4.4. Formatos de texto: JSON y HTML

HTML[26] es el lenguaje de marcado estándar para la creación de páginas web. En este proyecto se ha empleado para crear algunas páginas web necesarias para recuperar los datos subidos al servidor.

El volcado de datos en disco se ha realizado mediante la creación de ficheros **JSON**[27] un formato ligero de intercambio de datos independiente de lenguaje.

4.5. Algoritmos criptográficos: AES-128 y PBKDF2

Con el objetivo de no almacenar las contraseñas de los usuarios en texto plano estas se han almacenado tras pasar por la función hash **PBKDF2**[28].

Para añadir más seguridad también se han encriptado los ficheros usando el algoritmo de encriptado simétrico **AES-128**[29] donde el 128 hace referencia al tamaño en bits de la clave que utiliza.

4.6. Utilidades para la generación de certificados: OpenSSL y Keytool

Para la generación del certificado raíz de una autoridad de certificación ficticia firmado con mis propios datos se utilizó la herramienta **OpenSSL**[30] que provee un conjunto de herramientas para el protocolo TLS usado para establecer comunicaciones seguras por HTTPS.

También se ha usado la herramienta **Keytool**[31] para la generación de un almacén de certificados requerido para poder establecer conexiones seguras con Tomcat y para generar una petición de firma de certificado, la cual fue firmada con el certificado raíz anterior e introducida en el almacén de certificados de Tomcat.

4.7. Bibliotecas

4.7.1. Jersey

Jersey[32] constituye la implementación de referencia de la API de Java JAX-RS[33] para el desarrollo de servicios web RESTful. En concreto se ha usado su versión 2.22.2 con licencia CDDL.

---

[26] https://www.w3.org/html/
[27] http://json.org/
[28] https://tools.ietf.org/html/rfc2898
[29] http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
[30] https://www.openssl.org/
[31] https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html
[32] https://jersey.java.net/
[33] https://jax-rs-spec.java.net/

4.7.2. OpenCV[34]

Librería de visión artificial liberada bajo licencia BSD, implementa un gran número de algoritmos de visión artificial como los explicados en el apartado 3.1 de esta memoria. Esta librería es multilenguaje contando con interfaces para C, C++, Python y Java, siendo esta ultima la utilizada. En concreto se ha utilizado su versión 3.1.

4.7.3. Tesseract y Tess4J

Tesseract (véase 3.2.1) es uno de los principales motores OCR en el mundo del software libre (cuenta con una licencia Apache 2.0), dispone de una API en C y C++, por ello se ha utilizado **Tess4J**[35], un wrapper que permite ejecutar la API de Tesseract en Java.

4.7.4. Ion

Para establecer las conexiones desde la aplicación Android al servidor se ha utilizado la librería **Ion**[36] bajo licencia Apache 2.0, esta permite establecer conexiones asíncronas de un modo muy simple

.

4.7.5. PDFBox

Apache **PDFBox**[37] es una librería de software libre (licencia Apache 2.0) para trabajar con archivos PDF. Se ha empleado su versión 2.0.2 para construir documentos PDF con los DNI y el contenido de la base de datos de campañas para que los usuarios puedan recuperar sus subidas.

4.7.6. Boxable

Para poder construir tablas en los documentos PDF en las que volcar el contenido de la base de datos de una campaña se emplea **Boxable**[38] 1.4 (licencia Apache 2.0), esta librería usa PDFBox para crear tablas de forma fácil sin tener que preocuparse de aspectos a bajo nivel.

4.7.7. Gson

La conversión de objetos Java a su representación en JSON se ha realizado con **Gson**[39] en su versión 2.6.2 bajo licencia Apache 2.0.

---

[34] http://opencv.org/
[35] http://tess4j.sourceforge.net/
[36] http://koush.com/ion
[37] https://pdfbox.apache.org/
[38] http://dhorions.github.io/boxable/
[39] https://github.com/google/gson

4.7.8. Secure Password Storage v2.0

Como implementación del algoritmo PBKDF2 se ha elegido la librería **Secure Password Storage v2.0**[40], que utiliza la implementación interna de Java de dicho algoritmo y permite la generación de hashes a partir de contraseñas así como la verificación de estos hashes.

4.8. Sistema de control de versiones: Git, GitHub y Bitbucket

Para gestionar las distintas versiones de código producidas a lo largo del proceso de desarrollo de este proyecto se ha utilizado la herramienta **Git**[41], en concreto su versión de terminal descargada vía apt.

Se han usado dos servidores Git distintos para alojar los artefectos producidos en el desarrollo, el código correspondiente a la aplicación Android y al servidor se encuentra en repositorios de **GitHub**[42] y para incrementos del producto o pruebas intermedias se han generado proyectos en **BitBucket**[43], se han usado estos dos servidores debido a que no se pueden crear repositorios privados en GitHub de manera gratuita mientras que en BitBucket sí.

4.9. Documentación del código: JavaDoc y JAutodoc

JavaDoc[44] ha sido utilizado para generar una documentación de programador en HTML a partir de comentarios con determinado formato en los ficheros de código Java.

También se ha empleado el plugin de Eclipse **JAutoDoc**[45] como ayuda a la generación de comentarios en el código con el formato de JavaDoc.

4.10. Herramientas para la documentación del proyecto

4.10.1. Visual paradigm

Los distintos diagramas UML que forman parte de la documentación técnica han sido desarrollados utilizando la herramienta **Visual Paradigm**[46] en su versión 13.1 con la licencia académica de la Universidad de la Salamanca.

40 https://github.com/defuse/password-hashing
41 https://git-scm.com/
42 https://github.com/
43 https://bitbucket.org/
44 http://www.oracle.com/technetwork/articles/java/index-jsp-135444.html
45 http://jautodoc.sourceforge.net/
46 https://www.visual-paradigm.com/

4.10.2. REM

REM[47] es una herramienta CASE de gestión de requisitos con la que se han descrito los requisitos, objetivos y casos de uso de la documentación técnica, en concreto se ha utilizado la versión 1.2.2 de la herramienta.

4.10.3. Microsoft Project

La planificación temporal del proyecto y sus respectivos diagramas se ha servido de la herramienta **Microsoft Project 2013**[48] descargada gracias a la licencia académica de la Universidad de Salamanca.

4.10.4. Microsoft Office Word

Para redactar tanto el presente documento como los distintos anexos se ha usado el procesador de textos **Microsoft Word 2013**[49] también gracias a la licencia académica de Universidad de Salamanca.

[47] https://www.lsi.us.es/descargas/descarga_programas.php?id=3
[48] https://products.office.com/es-es/project/project-and-portfolio-management-software
[49] https://products.office.com/es-es/word

5. Aspectos relevantes del desarrollo

En este apartado se detallara los puntos principales del proceso de desarrollo, problemas encontrados, alternativas valoradas… Todo ello organizado entorno al modelo de proceso software seguido, Scrum (véase 3.5).

Como introducción decir que además del título oficial de este proyecto *"Automatización de la recogida de firmas para iniciativas ciudadanas"* se ha decidido llamarle **Demos**. Este nombre está inspirado en la raíz griega del prefijo demo que significa pueblo[50] y se debe a que el objetivo de este proyecto no es otro que el de ayudar a llevar la democracia a la gente.

El Scrum seguido en este proyecto tiene algunas particularidades derivadas de las especificaciones del proyecto. No existe un cliente o product owner, sí que hay un scrum master (el tutor, Rodrigo Santamaría Vicente) y el equipo de desarrollo está formado por un solo individuo (yo mismo). Esto tiene dos consecuencias sustanciales: primero que las tareas del cliente han tenido que ser realizadas por los otros dos stakeholders y segundo que las reuniones scrum diarias para coordinar carecen de sentido al ser una única persona el desarrollador. A pesar de todo ello ha sido elegido como el modelo de proceso más adecuado debido a los requisito poco definidos del proyecto, al escaso equipo de desarrollo y a creer que se adapta mejor a un proyecto pequeño como este que un modelo más tradicional como el proceso unificado ya que prima mucho más fases formales y de documentación.

Ahora pasaré a detallar los distintos sprints en los que se ha dividido la realización del proyecto así como sus aspectos más destacados de cada uno.

5.1. Primer sprint

Este sprint inicial comenzó con una reunión de planificación el día 29/01/2016 y tuvo una duración de un mes, acabando el 29/02/2016. A pesar de que el TFG se adjudicara el 16/12/2016 antes de este primer sprint solo hubo una reunión inicial para aclarar los objetivos del proyecto.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Hacer fotos con una aplicación Android.
- Subir fotos a un servidor.
- Manejo de Git y la plataforma GitHub.

Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Hacer fotos con una aplicación Android controlando el manejo de cámara y el enfoque a corta distancia.
- Redimensionado de estas fotos buscando alguna librería de imágenes, presumiblemente OpenCV, la cual estaba empezando a conocer en la asignatura de Robótica.

[50] http://dle.rae.es/?id=C9MVDo9

- Subir fotos usando un servicio web de arquitectura REST, usando peticiones POST para la subida de archivos.
- Manejo de Git, creando una cuenta en GitHub donde se alojara el proyecto con alguna licencia de software libre para que pueda ser continuado o aprovechado por otras personas y manejo del comando Git.

En este punto se empezaron a tomar decisiones de tecnologías o arquitecturas a usar como se puede ver. Se intentó aprovechar las asignaturas que estaba cursando en aquel momento, Robótica en la cual se comenzó a aprender a manejar **OpenCV** (véase 4.7.2) o Sistemas Distribuidos en la que se utilizan servicios web con arquitectura **REST** (véase 3.3) corriendo en servidores **Tomcat** (véase 4.3) en las prácticas, para poder consultar en caso de necesidad a los respectivos profesores de dichas asignaturas (Jesús F. Rodríguez Aragón y Rodrigo Santamaría Vicente). Se decidió decantarse por el lenguaje de programación **Java** (véase 4.1).

5.1.1. Hacer fotos con una aplicación Android.

Cabe destacar que el desarrollador tenía alguna experiencia previa en Android, así que estaba familiarizado con los conceptos básicos de la plataforma. Cuando se inició el desarrollo llamó la atención que Android había cambiado su IDE recomendado de **Eclipse** (véase 4.2) con el plugin Android Development Tools (ADT)[51] a un IDE propio llamado **Android Studio** (véase 4.2) basado en IntelliJ IDEA[52].

Tras familiarizarse con el nuevo IDE gracias a la fantástica documentación para desarrolladores disponible en https://developer.android.com/index.html se comenzó a analizar las distintas alternativas para hacer fotografías con un dispositivo Android. Una opción era usar la aplicación de cámara del propio dispositivo[53] mediante el uso de *Intents* (véase 3.4) y la otra era construir una aplicación de cámara propia[54].usando la API de cámara android.hardware.camera2[55] de Android.

En la siguiente tabla se analizan las características de ambas alternativas.

[51] https://marketplace.eclipse.org/content/android-development-tools-eclipse
[52] https://www.jetbrains.com/idea/
[53] https://developer.android.com/training/camera/photobasics.html?hl=es
[54] https://developer.android.com/guide/topics/media/camera.html?hl=es#custom-camera
[55] https://developer.android.com/reference/android/hardware/camera2/package-summary.html?hl=es

TABLA 1: COMPARACIÓN ENTRE ALTERNATIVAS DE CÁMARA

	Usar aplicación nativa	**Construir aplicación**
Permite recuperar las fotografías tomadas y manejarlas al gusto	Sí	Sí
Permite controlar características de la cámara como el flash o el enfoque	No	Sí
Esfuerzo requerido	No demasiado	Cierto esfuerzo de programación de interfaz y funcionalidad

Se comprobó que no resulta imprescindible controlar características de la cámara porque las aplicaciones nativas de cámara suelen ser lo bastante buenas para tomar las fotografías que se requieren en modo automático, que el usuario estará mucho más acostumbrado a realizar fotografías con su aplicación nativa que ya conoce y que supondría un esfuerzo adicional el crear una nueva aplicación. Por ello se toma la decisión de **usar la propia aplicación de cámara del dispositivo** para tomar las fotografías.

5.1.2. Redimensionado de imágenes con alguna librería.

Como ya se ha dicho, se estaba aprendiendo en la asignatura de Robótica algún uso esencial de la librería de visión artificial OpenCV escrita en C++, la cual se adaptaba a la perfección a las necesidades del proyecto, pero el equipo de desarrollo carecía de experiencia en ese lenguaje y tanto la aplicación como el servicio web se iban a programar en Java por lo que se comenzó a buscar una opción similar en dicho lenguaje. Nada más ponerse a investigar se descubrió que los propios desarrolladores de OpenCV ofrecían una versión para Java de su librería, aunque la documentación o ejemplos de OpenCV en Java son bastante escasos se instaló dicha librería y se probó su método *resize*[56] la cual permite redimensionar imágenes al tamaño deseado de forma simple.

5.1.3. Subir fotos a un servidor.

Para la tarea de subir las fotos mediante un servicio web con arquitectura REST se usará la API de **Java JAX-RS** (véase 4.7), que se sirve de anotaciones (sentencias comenzadas con el carácter @ como *@Overwrite*) para convertir un objeto (instancia de una clase en Java) en un recurso web, esta API se introdujo en el estándar Java desde la versión EE 6. En concreto se utilizara la implementación de referencia de

[56] http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#resize

Oracle, el proyecto **Jersey** (véase 4.7). Cuando se comenzó a investigar cómo subir archivos usando dicha implementación en la mayoría de sitios se coincide que la forma de hacerlo es posteando un formulario de tipo *multipart/form-data* [RFC1867][57] el cual permite transmitir flujos de datos o data streams directamente y añadiendo como alguno de los campos de dicho formulario la propia fotografía. Se siguió el siguiente ejemplo como guía para esta parte del desarrollo [58]

Cabe destacar que para utilizar las funcionalidades multipart de Jersey es necesario incluir una librería adicional a las habituales de Jersey, *jersey-multipart.jar*. En todos los sitios consultados se coincide en usar un gestor de dependencias llamado Maven[59] desarrollado por Apache, el cual se encarga de descargar y gestionar las diversas librerías necesarias por un proyecto, pero también se pueden descargar estas librerías directamente de Internet. Esto último es lo que se hizo. Surgieron numerosos problemas con la subida de ficheros y no fue posible cumplir el objetivo de subir ficheros debido a múltiples excepciones producidas al efectuar la subida por lo que se dejó esta tarea para el siguiente sprint.

5.1.4. Manejo de Git y la plataforma GitHub.

Respecto a la última tarea del sprint el manejo de **Git** (véase 4.8) y la creación de una cuenta y repositorios en **GitHub** (véase 4.8) para los distintos artefactos del proyecto se crearon dos repositorios distintos, uno para la aplicación Android el cual es accesible desde la siguiente URL: https://github.com/AythaE/Demos_Android y otro para el servidor Rest: https://github.com/AythaE/Demos_Rest

5.2. Segundo sprint

Este sprint comenzó con una reunión de planificación y revisión del sprint anterior el día 29/02/2016 y tuvo una duración de siete semanas debido a problemas para cumplir las tareas planificadas, acabando el 19/04/2016.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Solucionar el problema con la subida de ficheros (del sprint previo).
- Detectar los DNI en las fotografías y recortarlos.

Realizar un proceso de OCR sobre los DNI en el servidor Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Solucionar el problema con la subida de ficheros (del sprint previo).
- Detectar el DNI en las fotografías investigando los diversos métodos de la librería OpenCV.
- Corregir problemas en la fotografía como en ángulo.

[57] http://www.rfc-base.org/txt/rfc-1867.txt
[58] https://examples.javacodegeeks.com/enterprise-java/rest/jersey/jersey-file-upload-example/
[59] https://maven.apache.org/

- Enviar las fotografías tomadas con la aplicación al servidor.
- Realizar OCR de los DNI recortados

5.2.1. Solucionar el problema con la subida de ficheros.

Explorando en mayor profundidad el error que se producía en la subida de archivos al servidor se descubrió que la librería *jersey-multipart.jar,* que era necesaria añadir para utilizar las características multipart de jersey, utilizaba internamente otra librería llamada *mimepull.jar*[60] para acceder a las partes adjuntas de un mensaje MIME[61]: Este es un estándar creado para extender el formato de los emails (que no ha de ser confundido con MIME Type) y soportar caracteres no ASCII o envío de ficheros, pero se ha usado en múltiples protocolos además de para el envío de emails como es en protocolo HTTP para el envío de *multipart/form-data.* Una vez incluida dicha librería funcionó el servicio web de subida de ficheros.

5.2.2. Detección de DNIs y corrección de su ángulo

Para la detección del DNI usando OpenCV se creó un proyecto Java aparte del servidor o la aplicación, este proyecto toma imágenes de disco hechas con algún móvil y les aplica todo el procesado de imagen incluido el OCR. Una vez que estuvo acabado se integró en servidor. Dicho proyecto es el incremento de este sprint y el motor principal del sistema, todas las modificaciones relacionadas con esta parte del sistema se han probado antes en dicho proyecto el cual se encuentra alojado en https://bitbucket.org/AythaE/detecciondni/src y al que se puede acceder con la cuenta de visitante habilitada para ello:

- Email: aythae@usal.es
- Contraseña: incrementoSegundoSprint

Antes de comenzar a explicar el proceso hay que destacar que es necesario hacer las fotografías de los DNI usando un folio como fondo para que el proceso funcione, ya que de no ser así no se puede detectar con claridad la diferencia entre el DNI y el fondo (véase Apéndice A: Fondo de fotografía recomendado). Ahora se procederá a explicar el proceso, lo primero que hay que hacer es convertir la imagen a escala de grises usando para ello el método *cvtcolor*[62], además de para la detección permite trabajar con imágenes de un solo canal por lo que las operaciones son más cómodas y rápidas. Tras esto se pasa la imagen por un filtrado gaussiano con el objetivo de mitigar posibles ruidos como brillos, polvo en la cámara…esto se hace mediante el método *GaussianBlur*[63], A continuación sería necesario convertir la imagen en escala de grises a una imagen binaria antes de aplicar el método *findContours*[64] que implementa un algoritmo de detección de contornos (véase 3.1.2). Según la propia documentación de dicho método se pueden usar varios métodos para binarizar una

[60] https://mimepull.java.net/
[61] https://en.wikipedia.org/wiki/MIME#Multipart_messages
[62] http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor
[63] http://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#gaussianblur
[64] http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findcontours

imagen, se probaron dos de ellos, *Canny*[65] y *threshold*[66], con resultados muy distintos. El primero de ellos implementa el detector de bordes Canny (para una explicación detallada véase[67]), este método requiere tres parámetros configurables para variar sus resultados, umbral bajo, umbral alto (que suele ser tres veces el umbral bajo) y tamaño del núcleo (matriz usada internamente para los cálculos).

En la siguiente imagen podemos ver un ejemplo del resultado de este método:

ILUSTRACIÓN 12: EJEMPLO DETECTOR DE BORDES CANNY

El segundo método implementa una umbralización, siendo el tipo elegido binaria inversa (véase 3.1.1 para ejemplo y explicación más detallada) debido a las características del problema. Este método cuenta también con varios parámetros de configuración, en concreto umbral bajo y alto (que suele ser el valor máximo para obtener una binarización inversa, 255), haciendo que todo pixel con una intensidad superior al umbral bajo se convierta en negro y todo aquel que no lo sea en blanco.

Probando ambos métodos y "jugando" con sus parámetros se llegó a la conclusión de que resulta más sencillo detectar los DNIs usando el método *threshold*, imagino que debido a que idealmente el DNI queda como una mancha rectangular blanca que es más fácilmente detectable que la imagen producida por *Canny* aunque esta tenga mayor grado de detalle.

Una vez binarizada la imagen se pasa al método *findContours,* como ya se ha mencionado antes, el cual devuelve un array con los contornos (el DNI entero, el cuadrado de la foto del DNI,…), para saber cuál de estos contornos es el del DNI se ha seguido el sencillo criterio de seleccionar aquel de mayor área usando para ello el método *contourArea*[68].Una vez seleccionado el contorno correcto se calcula el menor rectángulo que lo contiene con *minAreaRect*[69]. En la siguiente imagen se puede observar el contorno en color azul y el rectángulo que lo contiene en color verde.

---

65 http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#canny

66 http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#threshold

67 http://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html

68 http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#contourarea

69 http://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#minarearect

ILUSTRACIÓN 13: EJEMPLO RECTÁNGULO CONTENEDOR DE DNI

A partir de dicho rectángulo extraemos el ángulo que tiene[70] para poder corregirlo más adelante, dicho ángulo no es exacto pero es capaz de minimizar el ángulo de la imagen hasta un rango aceptable. Existe un problema respecto a esto y es que si la imagen del DNI esta girada 90º de la posición esperada no se puede saber si esta girada a la izquierda o a la derecha, por defecto se supone que está girada a la derecha, por lo que se toma el criterio de girarla hacia la izquierda, pero en caso de que esté girada hacia la izquierda, al volverse a girar hacia la izquierda quedaría boca abajo y el proceso de detección de texto mediante OCR fallará.

Con el objetivo de determinar si se ha encontrado el DNI correctamente se realiza una comparación del ratio de aspecto del rectángulo (ancho dividido por el alto) que lo engloba con el que debería tener un DNI en función de su tamaño (+/- un valor de margen). Si el valor entra dentro del margen se toma como correcto y se continua con el proceso, en caso contrario se repite todo el proceso decrementando el valor de umbral bajo del método *threshold*, con lo que se coge "menos parte" del DNI. Esta repetición se realiza un determinado número de veces tras el cual si no se ha detectado nada que pueda ser un DNI se da la detección por fallida. Cabe la posibilidad de que por casualidad se detecte algo que no sea un DNI pero que coincida con su relación de aspecto en cuyo caso presumiblemente el proceso de OCR fallara y también se dará por fallida la detección. En ambos casos se informará al usuario de que repita la fotografía.

Una vez detectado algo que parece ser un DNI se pasa a corregir su ángulo y recortarlo, esto se realiza sobre la imagen en color ya que posteriormente es la que se guardará en el servidor. Para ello primero se calcula la matriz afín de rotación en dos dimensiones en función del ángulo detectado en el paso previo y del centro del DNI detectado mediante el método *getRotationMatrix2D*[71], a continuación se pasa a aplicar esa transformación afín[72] mediante el método *wrapAffine*[73]. Una vez realizada la rotación se corta de la imagen (original en color) el área contenida en el

[70] http://felix.abecassis.me/2011/10/opencv-bounding-box-skew-angle/
[71] http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#getrotationmatrix2d
[72] https://es.wikipedia.org/wiki/Transformaci%C3%B3n_af%C3%ADn
[73] http://docs.opencv.org/2.4/modules/imgproc/doc/geometric_transformations.html#warpaffine

rectángulo de detección de DNI, pero con sus dimensiones algo ampliadas para intentar evitar cortar parte del DNI si el rectángulo es muy justo.

5.2.3. Detección óptica de caracteres en los DNI recortados

En este paso en el que ya se tiene lo que supuestamente es un DNI cortado y con su ángulo corregido se pasa a aplicar el procedimiento de OCR, para ello se emplea el motor **Tesseract** (véase 4.7.3 y 3.2.1) con el binding **Tess4J** (véase 4.7.3) para Java. Como ya se ha dicho Tesseract es probablemente el motor OCR con mayor andadura en el mundo del software libre, motivo por el cual ha sido seleccionado para esta parte del proyecto. Antes de pasar la imagen directamente al proceso de OCR se realiza un pretratamiento con algunos métodos de OpenCV ya explicados (recordemos que el DNI ha sido recortado a partir de la imagen original subida al servidor, sin tener tratamiento de imagen alguno), primero se pasa a escala de grises, se filtra de los posibles ruidos y se binariza con el método *threshold*. Tras eso se pasa al motor OCR al que se le indica que se configure para el lenguaje español. El resultado es una cadena de caracteres con todas las líneas de texto detectadas separadas con un \n.

Como conclusión de estos 2 apartados previos cabe destacar que se han encontrado problemas en la detección de la parte trasera del DNI electrónico 3.0[74] seguramente debido a sus colores más claros lo que hace que produzca menor contraste con el folio de fondo. El rendimiento de todo este proceso deja mucho que desear estando en tiempos de entre 15 segundos detectando una fotografía a la primera y hasta 80s en casos erróneos. La mayor parte de tiempo de este procesamiento se va en el proceso de detección de caracteres, no en la detección del DNI en la imagen. Se planteara mejorar el rendimiento en el siguiente sprint. **A pesar de que la detección de DNIs y el OCR se lograran implementar en el proyecto independiente, se experimentaron problemas con algunas librerías de OpenCV que imposibilitaron su integración en el servidor.**

5.2.4. Enviar las fotografías tomadas con la aplicación al servidor.

El último punto del sprint es el envío de las fotografías desde la aplicación Android al servidor. Como ya se ha dicho al comienzo de este sprint, para el envío de archivos al servidor es necesario enviar un mensaje HTTP *POST*[75] de tipo *multipart/form-data*, investigando como realizar esto en una aplicación Android el equipo se encontró con la librería **Ion** (véase 4.7.4) de koush la cual permite construir peticiones *POST* asíncronas del tipo deseado y añadiendo parámetros textuales o ficheros de un modo realmente simple como se puede ver en el siguiente ejemplo:

[74] http://computerhoy.com/noticias/hardware/nuevo-dni-electronico-30-asi-es-asi-funciona-39025
[75] https://en.wikipedia.org/wiki/POST_%28HTTP%29

```
Ion.with(getContext())
.load("https://koush.clockworkmod.com/test/echo")
.uploadProgressBar(uploadProgressBar)
.setMultipartParameter("goop", "noop")
.setMultipartFile("archive", "application/zip", new File("/sdcard/filename.zip"))
.asJsonObject()
.setCallback(...)
```

ILUSTRACIÓN 14: ION EJEMPLO DE POST MULTIPART/FORM-DATA

Así pues se añadió al proyecto Android y se logró enviar fotografías sin mayor problema.

La larga duración de este sprint está determinada por ser en el que se sientan las bases del núcleo del sistema (la detección de DNIs y el OCR), esto conllevó más tiempo del esperado pero a pesar de ello los resultados obtenidos son bastante satisfactorios.

5.3. Tercer sprint

Este sprint comenzó con una reunión de planificación y revisión del sprint anterior el día 19/04/2016 y tuvo una duración de cuatro semanas, acabando el 20/05/2016.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Solucionar el problema de integración de OpenCV en el servidor (del sprint previo).
- Probar a detectar los DNI electrónicos 3.0 dibujando un recuadro alrededor de estos en el folio para ayudar al proceso de detección a delimitar los bordes.
- Mejora del rendimiento en la detección y OCR.
- Guardar los resultados del OCR en algún formato estándar de intercambio de datos como JSON o XML que servirá de base de datos NOSQL.
- Mejorar usabilidad de la aplicación mostrando mensajes al usuario con los resultados de sus subidas.

Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Solucionar el problema de integración de OpenCV en el servidor (del sprint previo).
- Probar a detectar los DNI electrónicos 3.0 dibujando un recuadro alrededor de estos en el folio para ayudar al proceso de detección a delimitar los bordes.
- Mejora del rendimiento en la detección y OCR probando las siguientes técnicas:
    - Delimitar el área en la que realizar OCR del DNI, en lugar de aplicarlo al DNI completo hacerlo a la parte donde se encuentre el texto únicamente.
    - Entrenar al motor OCR para obtener resultados mejores y más rápidos.
- Guardar los resultados del OCR en algún formato estándar de intercambio de datos como JSON o XML que servirá de base de datos NOSQL.
- Realizar el envío y el proceso de detección con OCR a ambas caras del DNI en el servidor.
- Mejorar usabilidad de la aplicación mostrando mensajes al usuario con los resultados de sus subidas.

5.3.1. Solucionar el problema de integración de OpenCV en el servidor

Al estudiar más en profundidad el problema de la integración de OpenCV en el servidor Tomcat se descubrió que este radicaba en la librería nativa *libopencv_java.so*. Tomcat era incapaz de cargar esta librería a pesar de que se le indicara en Eclipse de donde tenía que tomarla porque al ejecutar Tomcat desde Eclipse este no recibe los parámetros que fijan en Eclipse si no que toma por defecto los del sistema, por lo que estaba buscando la librería en las ruta de librerías de Java (java-library-path), tras la inclusión de dicha librería en las rutas habituales de Java este problema se solventó.

5.3.2. Mejorar la detección de DNI dibujando su contorno en un folio

Se comprobó que simplemente marcando con un bolígrafo el contorno del DNI en un folio antes de hacer la fotografía los resultados de detección mejoraban sustancialmente, por ello se suministra junto con esta memoria el Apéndice A: Fondo de fotografía recomendado con el que se deberán hacer las fotos para asegurar el resultado previsto.

5.3.3. Recepción y procesamiento de fotografías en el servidor

Realizar el envío de las dos fotos en la aplicación resultó bastante simple, bastó con realizar otra fotografía después de tomar la primera y añadir ambas fotografías como *multipartFile* a la petición con Ion (véase 5.2.4). La recepción y tratamiento de dichas fotografías en el servidor resultó más trabajosa. Se modificó el proceso actual de subida para que al guardar las fotografías que le llegan en una petición al servidor se hiciera en paralelo con un hilo por fotografía y se siguió una estrategia idéntica para la detección y el OCR de las fotografías. Cabe mencionar que el nombre y apellidos se extraen de la cara posterior del DNI. Se ha elegido este criterio debido a que en la parte trasera inferior del DNI se encuentra una zona con información impresa OCR-B para la lectura mecanizada[76], es decir que está preparada para el reconocimiento óptico de caracteres (con una tipografía más grande, solo caracteres en mayúsculas…). Mientras que en la cara frontal, a pesar de que también aparezcan el nombre y apellidos, estos lo hacen con un tamaño menor y en medio de múltiples elementos, con lo que resulta más complicado detectarlos. El número del DNI, en cambio, se extrae de la parte frontal. Este número también aparece en la zona especialmente diseñada para OCR de la parte posterior, pero extraerlo de ahí sobrecargaría el procesamiento necesario para la parte trasera del DNI, mientras que para la delantera bastaría con detectarlo y recortarlo, hay que recordar que cómo se ha mencionado antes lo más pesado del procesamiento de los DNI es el OCR. Por ello si se extrajeran todos los caracteres de la parte posterior no se apreciaría la mejora de rendimiento que se busca con el procesamiento paralelo de ambas caras, además el número del DNI de la parte frontal aparece en una esquina y sin elementos textuales cercanos por lo que resulta más fácil de extraer que el nombre y los apellidos.

[76] https://es.wikipedia.org/wiki/DNI_electr%C3%B3nico_en_Espa%C3%B1a

Con el objetivo de mejorar el rendimiento del reconocimiento y el OCR se probó a limitar los caracteres que Tesseract buscaba[77] a letras mayúsculas en la parte trasera (de donde se extrae el nombre y apellidos) y números junto con letras mayúsculas en la delantera (para extraer el número del DNI). Además se limitó el área en la que realizar el procedimiento de OCR al tercio inferior de ambas fotografías recortadas. Con el objetivo de mejorar los resultados del OCR se realizaron intensas comprobaciones recorriendo los resultados del OCR en busca de alguna secuencia de caracteres que coincidiera con expresiones regulares usando el método *matches*[78] de Java tanto para el número del DNI como para el nombre y apellidos. Se adoptó una política de repetición del proceso en caso de error, variando los valores de umbral de la binarización pre OCR, similar a la que se seguía ya para la detección de los DNI, estableciendo también un límite de intentos. Con todas estas medidas se logró mejorar sustancialmente los tiempos de reconocimiento que oscilan entre 3 y los 15 segundos por fotografía en los peores casos (respecto a los 15- 80 segundos previos) además de mejorar notablemente el resultado del OCR, por ejemplo al dejar de confundir las I con 1 o las O con 0, pero dicho resultado nunca será perfecto ya que depende mucho de la calidad de la fotografía (resolución de la cámara, iluminación, enfoque…).

5.3.4. Guardar los resultados del OCR en disco

Como resulta obvio es necesario guardar información sobre los DNI en algún tipo de almacenamiento no volátil para poder recuperarla posteriormente y trabajar con ella o presentársela al usuario. Para ello se introdujo la clase *Firma* pensada para contener toda la información necesaria de cada subida al servidor. Inicialmente contenía los siguientes campos: nombre, apellidos, número de DNI (todos ellos extraídos del OCR), un *File*[79] en Java que apuntaba al fichero donde se encuentra la parte delantera del DNI recortado y otro para la parte trasera. Con el avance del desarrollo se fueron añadiendo otros campos necesarios que se explicaran posteriormente.

Se estimó que no era necesario tener una base de datos relacional tradicional para almacenar las instancias de *Firma*, que era suficiente con almacenarlas en una base de datos NoSQL de tipo documental[80]. Estas bases de datos se caracterizan por estar formadas por un conjunto de documentos independientes, los cuales contienen toda la información de la base de datos, por lo que no existen las tablas típicas de las bases de datos relacionales. Esto les confiere gran flexibilidad al no estar sujetas a esquemas concretos de almacenamiento lo que se traduce, por ejemplo, en que si es necesario incluir un campo nuevo en un documento se puede hacer directamente (sin tener que modificar una tabla, darle valores a todas las filas para ese nuevo campo…). Una de las características más interesantes es que permiten almacenar objetos directamente simplemente serializándolos mediante alguna librería externa o mediante herramientas del propio lenguaje. Los documentos que forman la base de datos suelen estar en un lenguaje de intercambio de datos estándar como JSON o XML. Se decidió elegir **JSON** (véase 4.4) debido a que es menos verboso que XML

[77] https://github.com/tesseract-ocr/tesseract/wiki/FAQ#how-do-i-recognize-only-digits
[78] http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#matches(java.lang.String)
[79] https://docs.oracle.com/javase/7/docs/api/java/io/File.html
[80] http://weblogs.asp.net/britchie/document-databases

por lo que ocupará menos espacio. Respecto a cómo manejar dichos ficheros se conocía con anterioridad un *parser* de Java a JSON conocido como **Gson** (véase 4.7.7), dicha librería es realmente sencilla permitiendo el paso de objetos Java a JSON con el método *toJson* y al contrario con *fromJson*, tiene licencia de software libre y permite el formateo de documentos JSON para mejorar su legibilidad.

Por tanto tras finalizar una detección y aplicarle el OCR se crea una instancia de firma con los datos extraídos y gracias a Gson se escribe en un fichero con extensión .json.

5.3.5. Mostrar mensajes al usuario con los resultados de sus subidas.

Por último para este sprint se hicieron algunas mejoras de usabilidad en la aplicación Android. La barra de progreso asociada a la subida y controlada por Ion se sustituyó por un dialogo de tipo modal (el cual impide que se interactúe con la ventana principal hasta que este desaparezca) que muestra una barra de progreso y desaparece cuando la subida concluye, se incluyeron múltiples mensajes para el usuario mediante el uso de pequeños mensajes emergentes, *Toast*[81], dichos mensajes incluyen por ejemplo un pequeño guiado para el usuario indicándole que haga la foto frontal, la posterior. También muestran el resultado de la subida si ha habido éxito, o si por el contrario tiene que repetir las fotografías debido a algún error.

Como conclusión, tras este sprint ya se tenía todos los mecanismos necesarios para subir fotografías, tratarlas, extraer su información y almacenarlas. Ahora falta montar el sistema de gestión de "usuarios" permitiendo asociar cada subida a cada usuario e implementar unas políticas de seguridad elevadas debido a la naturaleza tan sensible de la información que maneja el sistema.

5.4. Cuarto sprint

Este sprint comenzó con una reunión de planificación y revisión del sprint anterior el día 20/05/2016 y tuvo una duración de tres semanas, acabando el 10/06/2016.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Migración al servidor real en el que se desplegará el proyecto.
- Implementación de mecanismo de comunicación segura entre la aplicación y el servidor.
- Diseñar un sistema de usuarios a los que asociar las firmas subidas.

Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Migración al servidor real en el que se desplegará el proyecto. Lo cual implica:

[81] https://developer.android.com/guide/topics/ui/notifiers/toasts.html

  - Instalación de todas las herramientas necesarias para que funcione el servicio web en entorno con apenas sistema operativo y servicio de SSH.

  - Control de accesos al servidor por SSH

- Implementación de mecanismo de comunicación segura entre la aplicación y el servidor. Para ello se implementara una comunicación por HTTPS mediante el uso de certificados X.509 auto firmados.

- Diseñar un esquema de seguridad más amplio, que involucre no solo la comunicación si no los datos almacenados en el servidor.

- Diseñar un sistema de usuarios a los que asociar las firmas subidas, incluyendo todas sus operaciones. Registro, inicio de sesión, cierre de sesión, borrado de cuenta y su información asociada…

5.4.1. Migración al servidor real en el que se desplegará el proyecto

Hasta ahora el sistema se había desarrollado en un servidor en red local corriendo en mi ordenador, lo cual era suficiente para el desarrollo y prueba de la funcionalidad además de más cómodo por poder acceder directamente a las fotos recortadas o a las bases de datos de firmas, pero ahora la Universidad de Salamanca ha cedido uno de los servidores que tiene para proyectos de alumnos, localizado en la siguiente URL http://prodiasv08.fis.usal.es el cual ejecuta un SO Debian[82] 8.4 solo con un servicio SSH[83] instalado para poder comunicarse con él de forma remota (debido a que no se tiene acceso físico al servidor). Dicho servidor cuenta con 8GB de almacenamiento y 1GB de memoria RAM, lo que implica una limitación considerable de recursos que habrá que tener en cuenta, ya que se requieren instalar bastantes componentes y no son ligeros, OpenCV por ejemplo ocupa 1,7 GB.

Inicialmente el acceso SSH al servidor solo estaba habilitado dentro de la red de la Universidad, por lo que fue necesario manejar el sistema de seguridad **TCP Wrapper**[84] implementado en sistemas Unix que permite filtrar el acceso por red a determinados servicios, entre ellos SSH, para controlar los accesos permitidos o prohibidos basta con retocar los ficheros */etc/hosts.allow* y */etc/hosts.deny*.

Sobre el servidor simplemente mencionar para finalizar que se instaló Tomcat 8 como servicio de modo que tiene su propio usuario en el sistema y se ejecuta solo al arrancar el servidor en lugar de arrancarlo mediante Eclipse o por terminal.

[82] https://www.debian.org/
[83] https://en.wikipedia.org/wiki/Secure_Shell
[84] ftp://ftp.porcupine.org/pub/security/tcp_wrapper.pdf

5.4.2. Implementación de mecanismo de comunicación segura, HTTPS

Para poder establecer conexiones HTTP sobre TLS en Tomcat se intentó seguir el tutorial oficial de Apache[85], creando un certificado auto firmado con la herramienta de java **Keytool** (véase 4.6) y modificando el fichero *server.xml* de Tomcat para habilitar un conector con la implementación **NIO JSSE**[86] en el puerto por defecto de Tomcat, el 8443. Se consiguió hacer funcionar en el servidor local pero no en el servidor remoto. Así que se intentó replicar una práctica similar realizada en la asignatura de Seguridad en Sistemas Informáticos en la cual se habilitaba HTTPS en un servidor web Apache mediante el uso de certificados auto firmados con la utilidad **OpenSSL** (véase 4.6), para esto era necesario crear un certificado auto firmado que actuaría como certificado raíz de una CA propia y con el cual se firma el certificado del servidor. Otra vez el resultado fue el mismo, se consiguió implementar en local pero no en el servidor remoto. Así pues se decidió consultar al tutor sobre esto ya que era probable que fuera algún problema de configuración del servidor desconocido.

Además de esto es recomendable modificar el fichero *web.xml* del proyecto añadiendo una restricción de seguridad para redirigir toda conexión por https, esto se realiza añadiendo el siguiente extracto en dicho fichero.

```
1  <!-- Require HTTPS for everything-->
2  <security-constraint>
3      <web-resource-collection>
4          <web-resource-name>TodoHTTPS</web-resource-name>
5          <url-pattern>/*</url-pattern>
6      </web-resource-collection>
7      <user-data-constraint>
8          <transport-guarantee>CONFIDENTIAL</transport-guarantee>
9      </user-data-constraint>
10 </security-constraint>
```

ILUSTRACIÓN 15: AÑADIDO A WEB.XML PARA HTTPS

Además de lo anterior que es aplicable al servidor debido a que los certificados son auto firmados también es necesario modificar la aplicación Android, debido a que por defecto, al intentar establecer una conexión por HTTPS en el que no se conoce al CA que firma el certificado del servidor se produce una *SSLHanshakeException*[87] esto se puede arreglar "enseñándole" a confiar en los certificados de un determinado CA aunque lo desconozca, para ello es necesario incluir en la aplicación el certificado X509 raíz del CA creado (con el que se ha firmado el certificado del servidor) y seguir las indicaciones dadas en [88], además de ello para utilizar certificados auto firmados con la librería Ion[89] es necesario modificar el final de las indicaciones previas según el siguiente enlace [90]

[85] https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html#Configuration
[86] http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html
[87] https://developer.android.com/reference/javax/net/ssl/SSLHandshakeException.html
[88] https://developer.android.com/training/articles/security-ssl.html#UnknownCa
[89] https://github.com/koush/ion/issues/3
[90] http://stackoverflow.com/a/34627003

5.4.3. Diseñar un sistema de usuarios a los que asociar las firmas subidas.

En lugar de crear un sistema de usuarios con una cuenta por cada individuo se pensó que era mejor crear el concepto de **campaña** de firmas. Todas las firmas recogidas están asociadas a algún fin o campaña de recogida de firmas, por ello en lugar de crear una cuenta de usuario por cada individuo que suba firmas se crea una cuenta de campaña que puede ser compartida en múltiples dispositivos de distintas personas que colaboren con la recogida de firmas de la campaña. De esta manera se tratan por igual todas las firmas asociadas a una campaña, simplificando el proceso y se ahorran recursos al tener que mantener información sobre menos cuentas de usuario.

Será necesario por tanto almacenar de algún modo el nombre de la campaña y su contraseña para poder autenticar a sus usuarios, esto se hará mediante una base de datos documental siguiendo la misma aproximación que para guardar firmas. Cada campaña tendrá asociado su propio directorio (con igual nombre que la campaña) en el que se almacenarán las imágenes de los DNI y el JSON con la información de las firmas.

Para el borrado de cuentas se barajaron dos posibilidades, la primera añadir algún campo de "tiempo de vida" en la campaña el cual se re-estableciera cada vez que alguien accedía a ella y programar su borrado automático tras cierto tiempo sin usarse una cuenta y la segunda que los propios usuarios marcaran una fecha de borrado cuando registraran una campaña y de modo similar esta se borrara automáticamente al llegar dicha fecha. Se pensó que la segunda opción era mejor dado que el usuario tendría un mayor control sobre la campaña, no se añadían responsabilidades en el servidor y la carga adicional para el usuario era prácticamente nula, solo tenía que preocuparse de ello al registrar una campaña. Es necesario puntualizar que cada vez que se habla de borrado de una campaña esto se refiere a borrado de la base de datos de campañas y de su directorio asociado, por ello de todos sus DNI y su base de datos de firmas.

Para almacenar todos los archivos necesarios, la base de datos de campañas, los directorios de las campañas, etc… Se ha creado un directorio general para el sistema en */demos* sobre el cual se almacenaran el resto de archivos y directorios.

5.4.4. Diseñar un esquema de seguridad que proteja los datos del servidor.

Respecto a la seguridad no basta con proteger las comunicaciones ya que aunque los canales sean seguros, si alguien podría acceder al servidor de forma ilícita. Tendría acceso a toda una colección de DNIs, así como a los nombres y contraseñas de todas las campañas, por ello dada la naturaleza de la información almacenada se hace necesario proteger estos datos, para ello el directorio general del proyecto, */demos*, tendrá como dueño al usuario tomcat8 (el usuario del servicio Tomcat) y solo permisos de escritura, lectura y ejecución para él. Dentro de este directorio irán la base de datos de campañas así como los directorios de cada una con sus imágenes de

los DNI y la base de datos de firmas. Con esto se estaría protegido de accesos indebidos siempre y cuando la cuenta root del sistema, alguna con acceso sudo o la cuenta del sistema tomcat8 no quedará comprometida, en cuyo caso estaríamos igual que antes, por ello es necesario aplicar otra medida de seguridad más, encriptar todas las imágenes de los DNI así como las bases de datos. Para ello se usará un cifrador simétrico, **AES128** (véase 4.5), cuyo tamaño de clave es 128 bits el cual debería de ser suficiente para asegurar los datos[91] sin comprometer el rendimiento del sistema. Cada campaña tendrá asociada su propia clave para encriptar sus datos y adicionalmente existirá una clave para encriptar la base de datos de campañas.

Para concluir, relacionado con el tema de la seguridad, también habrá que gestionar las contraseñas de las campañas de modo correcto, ya que estas no se van a almacenar como texto plano, si no que se almacenará un hash[92] de la contraseña concatenada con un conjunto aleatorio y variable en cada hash de caracteres, lo que en criptografía se conoce como como un salt[93]. Este procedimiento se aplica extensamente a la hora de almacenar contraseñas, los sistemas Unix usan dichos elementos. Además habrá que seleccionar la función hash, ya que las más habituales como MD5 o SHA1 son inseguras[94]. Siguiendo las recomendaciones de la Open Web Application Security Project (**OWASP**)[95] sobre el almacenamiento de contraseñas lo recomendable es usar una función hash lenta como Argon2 o **PBKDF2** (véase 4.5) Se elegirá esta última debido a que se encuentra implementada en Java Cryptographic Extensions (**JCE**) que forma parte de JDK desde la versión 1.4.

5.5. Quinto sprint

Este sprint comenzó con una reunión de planificación y revisión del sprint anterior el día 10/06/2016 y tuvo una duración de tres semanas, acabando el 23/06/2016.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Solucionar los problemas con HTTPS en el servidor.
- Implementar los mecanismos de seguridad diseñados en el sprint anterior.
- Implementar el sistema de gestión de campañas diseñado en el sprint anterior.
- Establecer un método de recuperación de DNI.

Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Solucionar los problemas con HTTPS en el servidor ya que efectivamente el puerto usado estaba cerrado
- Implementar los mecanismos de seguridad diseñados en el sprint anterior.
- Implementar el sistema de gestión de campañas diseñado en el sprint anterior.

[91] https://www.keylength.com/en/compare/
[92] https://en.wikipedia.org/wiki/Cryptographic_hash_function
[93] https://en.wikipedia.org/wiki/Salt_%28cryptography%29
[94] https://www.owasp.org/index.php/Top_10_2007-Insecure_Cryptographic_Storage
[95] https://www.owasp.org/index.php/Main_Page

- Establecer un método de recuperación de DNI montando las fotos en un PDF aprovechando el espacio el máximo posible.

5.5.1. Solucionar los problemas con HTTPS en el servidor

Tras una llamada del tutor a José Andrés Vicente Lober, administrador de sistema de la USAL y encargado de los servidores, se descubrió que efectivamente el puerto por defecto para las conexiones HTTPS de Tomcat, el 8443, estaba cerrado así que fue necesario mudarlo al 443 estándar de HTTPS. Para ello no basta con cambiarlo sin más debido a que para acceder a los puertos por debajo del 1024 es necesario permisos de root[96] y el usuario tomcat8 no los tiene, por ello fue necesario utilizar la herramienta **authbind**[96] diseñada para permitir a los usuarios normales acceder a los puertos deseados sin tener permisos de superusuario, para ver un ejemplo vaya a[97].

5.5.2. Implementar los mecanismos de seguridad diseñados previamente

La gestión de las diversas contraseñas para el cifrado se ha realizado a través de un almacén de claves o *Keystore*[98], como se recomienda desde Java, dicho almacén de claves se encuentra en el directorio principal del sistema (*/demos*) bajo el nombre *.keystore* y está protegido con una contraseña necesaria para extraer o guardar cualquier clave de él. Esta contraseña "maestra" del Keystore se encuentra en el fichero */etc/tomcat8/.DemosKey* que pertenece a tomcat8 y solo con permisos de lectura para él, su contenido es una cadena de caracteres generada con un generador de contraseñas online[99] En dicho Keystore se almacena la clave de la base de datos de campañas bajo el alias de "master_key" así como una clave por cada campaña que utiliza de alias, el propio nombre de la campaña. Estas claves de campañas son usadas para encriptar cada una de las fotografías subidas así como la base de datos de firmas.

Los hashes de las contraseñas de campañas se generan con el algoritmo *PBKDF2WithHmacSHA256* y la implementación ofrecida por Secure Password Storage (véase 4.7.8), la cual se encuentra en la clase *PasswordStorage* que simplemente utiliza las funcionalidades internas de Java pero permite crear y verificar contraseñas con facilidad.

Para el encriptado y desencriptado simétrico con AES 128 se han seguido las recomendaciones y ejemplos de OWASP[100] creando con todo ello la clase *SymmetricEncription*, con los métodos necesarios para configurar el Keystore, encriptar /desencriptar ficheros usando una contraseña concreta o generando y guardando una y encriptar/desencriptar tokens (se explicarán más adelante).

---

96 https://en.wikipedia.org/wiki/Authbind
97 https://debian-administration.org/article/386/Running_network_services_as_a_non-root_user.
98 https://docs.oracle.com/javase/8/docs/api/java/security/KeyStore.html
99 http://passwordsgenerator.net/
100 https://www.owasp.org/index.php/Using_the_Java_Cryptographic_Extensions

La base de datos de campañas se encuentra en el directorio principal del sistema y se llama *campaigns.json*, contiene instancias de la clase *CampaignCredentials*, las cuales a su vez contienen el nombre, hash de la contraseña y fecha de borrado de las campañas. Estas instancias se convierten a representación textual gracias Gson y dicha representación textual en lenguaje JSON es encriptada con la clave propia de esta base datos antes de almacenarse en disco.

Por último el directorio principal de la aplicación contiene un subdirectorio *campanias* dentro del que se irán creando los diversos directorios de cada campaña.

A modo de resumen en la siguiente imagen se aprecia el contenido del directorio */demos*.

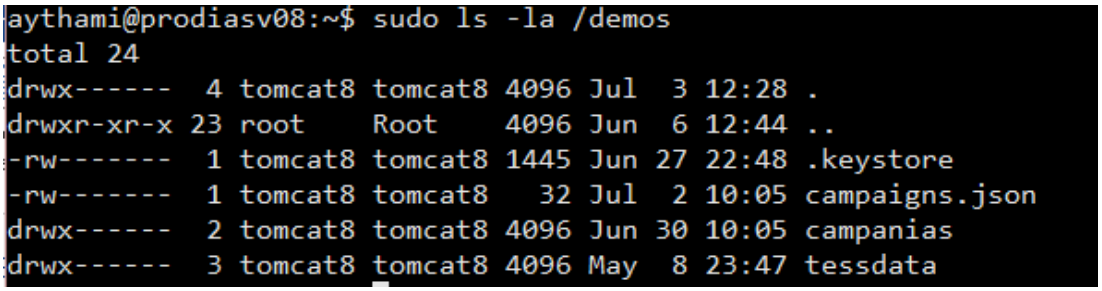

ILUSTRACIÓN 16: CONTENIDO DIRECTORIO /DEMOS

5.5.3. Implementar el sistema de gestión de campañas diseñado previamente

El control de las sesiones de usuario se hace mediante **tokens**, estos se envían al usuario si su login o registro ha sido correcto y se guardarán en su dispositivo enviándose junto con el nombre de la campaña (también almacenado en el dispositivo) en cada subida de fotos o al arrancar la aplicación, para que no tenga que iniciar sesión cada vez. Estos tokens se generan encriptando con el AES128 el nombre de la campaña con su clave de encriptado simétrica y anteponiéndole el vector de inicialización (IV) usado para encriptar dicho token. Este vector de inicialización actúa como un "salt" de la encriptación ya que asegura que encriptando dos veces lo mismo con la misma clave los resultados sean distintos (debido a que es generado con un generador de números pseudo-aleatorios criptográficamente seguro incluido en Java). Por lo tanto:

Token = IV + nombreCampaña_Encriptado

Esta forma de generar los tokens hace que sea posible comprobar a partir de qué campaña han sido generados utilizando el nombre de campaña que se envía junto al token en cada petición, extrayendo el IV (porque se conoce su longitud) y descifrando el resto con la clave de la campaña. Si el nombre de la campaña y el resultado de la desencripción coinciden es que es un token valido. Además se mantiene una estructura de datos de tipo *HashSet*[101] con los tokens activos por motivos de eficiencia.

[101] http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html

La implementación del sistema de gestión de usuarios se ha llevado a cabo en la clase *CampaignManagement* del servidor, para el registro de usuarios se ha creado el método ***register*** el cual recibe el nombre de la campaña, la contraseña y la fecha de borrado codificadas en Base64[102] (para una transmisión de los caracteres sin que se modifiquen por incluir acentos por ejemplo), los decodifica y realiza ciertas comprobaciones sobre dichos campos (que no exista una campaña de igual nombre por ejemplo) si todo es correcto genera el hash de la contraseña. Tras esto se comprueba si existe la base datos de campañas, en caso contrario se entiende que es la primera campaña en registrarse, por lo que el sistema acaba de ser desplegado así pues se crea el directorio principal de la aplicación */demos*, el Keystore y se configura este último incluyendo una contraseña maestra usada para encriptar la base de datos de campañas. Ahora se crea una instancia de la clase *Campaign* que contiene los datos necesarios para manejar una campaña como su nombre o su directorio (no su contraseña ni su fecha de borrado), se crea su directorio, su base de datos de firmas y se genera un token que se devolverá al usuario si el proceso de registro concluye correctamente. A continuación se crea una instancia de *CampaignCredentials* y si todo ha ido bien se escribe esta instancia a la base de datos de campañas (creándola si no existe por ser la primera campaña). A continuación se inserta el token generado en el *HashSet* de activos, se inserta la instancia de *Campaign* creada en una estructura de datos de tipo *HashMap*[103] cuyas entradas son de la forma <nombreCampaña, instanciaCampaign> para que se pueda acceder a la instancia *Campaign* a partir del nombre de la campaña y por último se le envía el token codificado en Base64 al usuario como respuesta.

El proceso de login también se lleva a cabo en la clase *CampaignMangement,* en el método ***login*** este recibe como parámetros el nombre y contraseña de la campaña en Base64, tras decodificar estos parámetros abre la base datos de campañas y va desencriptando campaña a campaña comprobando si coincide el nombre de la campaña desencriptada con el que busca, en caso de ser así se invoca al método *verifyPassword* de *PasswordStorage* que comprueba si la contraseña enviada al método login coincide con el hash almacenado, de ser así se genera un token como se ha explicado previamente, se añade al *HashSet* de tokens y se le envía al usuario, en caso incorrecto se le envía un mensaje de error el cual se le mostrará en la aplicación Android.

La clase *CampaignManagement* también cuenta con un método para autenticar tokens llamado ***authenticateToken*** el cual es invocado al arrancar la aplicación Android si el dispositivo dispone de token almacenado, este método comprueba si el token se encuentra en el *HashSet* de tokens activos, si es así responde a la aplicación y el usuario iniciaría sesión sin tener que introducir ningún credencial. Si el token no está en activos se comprueba su origen desencriptándolo como se ha dicho previamente, si coincide se añade a los tokens activos y se le responde afirmativamente al usuario, si no se le responde negativamente lo que borra el token del dispositivo Android.

Para la gestión de campañas en la aplicación Android se han incluido las actividades *LoginActivity* y *RegisterActivity* que son formularios de login y registro

[102] https://www.ietf.org/rfc/rfc4648.txt
[103] https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

respectivamente con los campos textuales necesarios para estos procesos y las cuales envían peticiones a los correspondientes métodos del servicio web. Como ya se ha dicho si el login o el registro finalizan exitosamente se le envía un token al dispositivo Android el cual estas actividades almacenan en un fichero *.token* contenido en un directorio privado de la aplicación dentro de la memoria interna del dispositivo. Este fichero contiene dos líneas de texto, la primera con el nombre de la campaña y la segunda con el token recibido. Cuando se inicia la aplicación se lanza inicialmente la actividad *CheckSessionActivity* la cual accede a este fichero (si existe) y lanza una petición de autenticación de token al servicio web, si esta falla o no existe el fichero se pasa a *LoginActivity* y si por el contrario tiene éxito se accede directamente a *MainActivity*, la actividad encargada de hacer y enviar las fotografías de los DNI.

Ahora que se dispone de múltiples actividades cada una con su pantalla se hace necesario un método para navegar entre ellas, para ello se ha pensado en un *Navigation Drawer*[104] siguiendo las guías de diseño de Google. En la siguiente imagen se aprecia el aspecto del formulario de inicio de sesión. A la izquierda del nombre de la aplicación se observa el botón para desplegar el Navigation Drawer.

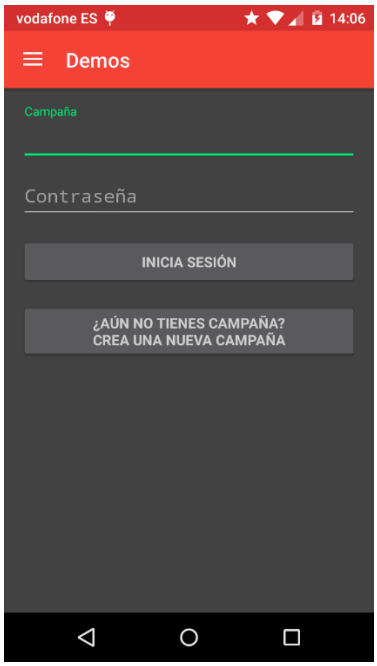

ILUSTRACIÓN 17: LOGINACTIVITY INTERFAZ

5.5.4. Establecer un método de recuperación de DNI en un PDF

Para finalizar en la recuperación de DNIs es necesario construir un PDF[105] por lo que se comenzó a buscar librerías que permitieran hacer esto en Java. La mayor parte de ellas son software privativo como iText[106], en el ámbito del software libre destaca **PDFBox** (véase 4.7.5), esta librería desarrollada y mantenida por Apache

104 https://material.google.com/patterns/navigation-drawer.html#navigation-drawer-behavior
105 https://en.wikipedia.org/wiki/Portable_Document_Format
106 http://itextpdf.com/

permite crear o manipular documentos PDF extrayendo texto por ejemplo. En el caso concreto que nos interesa permite crear PDF con imágenes incluyendo y redimensionando imágenes a placer, usando el repositorio oficial de ejemplos[107] se logró crear PDFs con cinco DNI por página como se puede apreciar en la siguiente imagen:

ILUSTRACIÓN 18: EJEMPLO DE PDF MONTADO

El principal problema con esta librería es el desorbitado tamaño que adquieren los PDF generados, en el caso del ejemplo superior por ejemplo, a pesar de que todas las imágenes pesaban unos 5MB aproximadamente el PDF alcanza un peso de 32 MB y dicho peso se incrementa linealmente con el número de DNIs, (2 páginas con 10 DNIs pesan 64 MB). También puede ser destacable el tiempo que se tarda en generar los PDFs el cual es de unos 4 segundos por página.

5.6. Sexto sprint

Este sprint comenzó con una reunión de planificación y revisión del sprint anterior el día 23/06/2016 y tuvo una duración de una semana y media, acabando el 05/07/2016.

El backlog o retraso de este sprint estaba formado por las siguientes tareas:

- Integrar la recuperación de DNIs en el servicio web.
- Incluir ayudas para el usuario en la aplicación.
- Implementar el borrado de campañas.
- Volcar la base de datos de firmas de la campaña al final del PDF generado.

Tras un refinamiento se llegó al siguiente backlog a desarrollar en el sprint:

- Integrar la recuperación de DNIs en el servicio web creando para ello un formulario web para descargar el PDF.
- Incluir ayudas para el usuario en la aplicación.
- Implementar el borrado de campañas creando comprobando periódicamente la fecha de borrado de estas.

[107] https://svn.apache.org/viewvc/pdfbox/trunk/examples/src/main/java/org/apache/pdfbox/examples/

- Realizar tareas de mantenimiento periódicas en el servidor entre las que se incluirían el borrado de PDFs generados para no llenar el almacenamiento del servidor o borrando los tokens de sesión activos.
- Volcar la base de datos de firmas de la campaña al final del PDF generado incluyendo un identificador de hoja de firmas y los campos adicionales que se consideren necesarios una buena presentación de la información.

5.6.1. Integrar la recuperación de DNIs en el servicio web

Debido al tiempo tardado en generar el PDF es necesario hacer este proceso de forma asíncrona. El proceso de descarga de las firmas de una campaña se inicia accediendo a la URL https://prodiasv08.fis.usal.es/Demos_Rest/formDownloadQuery.html donde se presenta un formulario de inicio de sesión, tras introducir los campos estos se codifican en Base64 y se lanza una petición al método *downloadQuery*, si se le autentica correctamente y la campaña tiene firmas para poder crear un PDF se le muestra una página informándole que su descarga estará lista en una determinada URL en un determinado tiempo (calculado empíricamente) y pesará aproximadamente un tamaño. Esta URL de descarga se genera con la dirección del método *download* del servicio web y dos parámetros, el nombre de la campaña y un token de descarga generado de igual manera que los tokens de sesión. Estos tokens de descargas son almacenados en la clase *CampaignManagement* en una estructura de datos de tipo *HashMap* cuyas entradas son de la forma <tokenDescarga, hiloQueGeneraPDF>. Cuando se intenta descargar un PDF con un token de descarga concreto este se valida y si es correcto se comprueba si el hilo ha acabado en cuyo caso se inicia la descarga, si se han producido errores o no se ha acabado se informa al usuario

Dar las estimaciones de tamaño o tiempo para generar el fichero PDF requiere conocer el número de firmas total que tiene una campaña, motivo por el cual se ha creado un contador en la clase *Campaign,* además se requiere almacenar su valor en un fichero llamado *.numFirmas* en el directorio de la campaña. Si no se almacenara y solo llevara el contador de número de firmas su valor se perdería al reiniciar el servidor.

5.6.2. Volcar la base de datos de firmas de la campaña en el PDF generado.

Es necesario asociar las firmas manuscritas en papel con las imágenes de los DNI almacenadas en el servidor, por ello hay que guardar un identificador hoja de firmas para cada firma de la base de datos de la campaña (en la clase *Firma* por tanto) para que posteriormente al volcar la base de datos en el PDF este número de hoja de firmas pueda servir para ayudar a los organizadores de la campaña a encontrar en sus hojas manuscritas los datos del firmante. Este identificador de la hoja de firmas es en un campo de texto por el usuario en cada subida de DNIs. Además se ha añadido un campo fecha de firma que se genera automáticamente al crear cada instancia de *Firma* que tiene el mismo propósito que el anterior, un campo hoja de DNI que representa la hoja del PDF generado en la que están las fotografías de los DNI asociadas a esta firma y un simple contador de firmas.

Para incluir la base de datos de la campaña en el PDF se pensó crear una tabla en el PDF, el problema es que PDFBox está diseñado para el manejo de PDF a muy bajo nivel, por ello para escribir texto hay que decirle las coordenadas exactas de cada sentencia, algo que se hace insostenible para este uso. Investigando como generar tablas con PDFBox se encontró **Boxable** (véase 4.7.6) una librería montada sobre PDFBox que facilita enormemente la creación de tablas, basta con ir creando filas y celdas dentro de cada fila con su contenido. La propia librería se encarga de dibujar las líneas, cambiar de página cuando se llena... En la siguiente imagen se puede ver cómo quedan las tablas generadas.

Nº	NÚMERO DNI	NOMBRE	APELLIDOS	FECHA	Nº HOJA DE FIRMAS	Nº HOJA DE DNI
1	70918176E	AYTHAMI	ESTEVEZ OLIVAS	04-07-2016 19:04	1	1
2	70918176E	AYTHAMI	ESTEVEZ OLIVAS	04-07-2016 19:09	1	1
3	07817153M	ESPER	OLIVAS SANCHEZ VIZCAINO	04-07-2016 19:23	1	1
4	07817153M	ESPER	OLIVAS SANCHEZ VIZCAINO	04-07-2016 19:25	1	1
5	70918176E	AYTHAMI	IVAS	04-07-2016 19:26	1	1

ILUSTRACIÓN 19: EJEMPLO DE TABLA EN EL PDF

5.6.3. Incluir ayudas para el usuario en la aplicación

Con el objetivo de ayudar al uso de la aplicación Android se ha creado la actividad *InstructionsActivity* la cual está formada por textos desplegables en los que se le indica al usuario como recuperar las firmas enviadas, que hacer si sus fotos son rechazadas porque falla su procesamiento o que sucede cuando manda las fotografías.

5.6.4. Implementar el borrado de campañas y tareas de mantenimiento del servidor

Con el objetivo de no agotar los recursos del servidor sería recomendable realizar tareas de mantenimiento encargadas de liberar recursos de este. Por ello se ha creado la clase *MaintenanceService* encargada de borrar las estructuras de datos de tokens activos y de descarga (para liberar memoria RAM), así como borrar los PDF que se han generado ya que estos ocupan un tamaño considerable como se ha mencionado y llenarían el disco del servidor rápidamente.

Para el borrado de campañas y las tareas de mantenimiento se pensó que lo ideal sería ejecutar un hilo periódicamente, cada día por ejemplo, y que este se encargue de realizar dichas tareas. Así se encontró la clase de Java *ScheduledThreadPoolExecutor*[108] que permite realizar la programación de hilos de forma periódica. Pero surge el problema de que para ejecutar un hilo a una hora determinada no basta con esto, esencialmente debido a los cambios hora con el horario de verano y de invierno, por ello es necesario hacer una tarea que envuelva a la tarea periódica y se auto programe al finalizar para ejecutarse al siguiente día[109].

[108] http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledThreadPoolExecutor.html

[109] http://stackoverflow.com/a/20388073/6441806

Esta funcionalidad de borrado de campaña también se ha implementado en la clase *MaintenanceService*, para ello se crea el fichero *campaigns.tmp.json,* se va leyendo la base de datos de campañas desencriptando campaña a campaña, se comprueba su fecha de borrado y si aún no ha llegado se escribe en el nuevo fichero temporal (encriptada), si por el contrario su fecha de borrado es igual o menor a la actual esta campaña no se escribe en el nuevo fichero y además se borra su directorio. Al finalizar el proceso se sustituye la base de datos de firmas original por la recién creada que contiene solo las campañas válidas. Todo este proceso de mantenimiento y borrado de campañas se ejecuta diariamente a las 5 AM.

6. Trabajos relacionados

En este apartado se analizarán varios sistemas de recogida de firmas comparándolos con el desarrollado en este proyecto

6.1. Change.org

El sitio web de peticiones online **Change.org**[110] es el mayor portal de este tipo en España, llegando a tener más de 10 millones de usuarios activos (el 27% de los usuarios de Internet en España)[111] y 150 millones en todo el mundo según proclaman ellos mismos. Fue lanzado en 2006 y tiene sede en Delaware, Estados Unidos[112]. El sitio permite crear peticiones a cualquier usuario con cuenta (se puede usar la cuenta de Facebook para entrar), basta darle un título, una descripción detallada y dirigirla a alguna persona u organización. Para firmar estas peticiones no se requiere cuenta, basta con dar un nombre, apellidos, correo, país y código postal. Ninguno de estos datos es comprobado, ni siquiera con un correo de confirmación.

Tras crear una petición esta se va compartiendo en redes sociales, mediante el boca a boca, etc. Si todo va bien esta petición puede llegar a los medios de comunicación y gracias a la presión social los destinatarios de esa petición actuaran resolviendo el problema para no ver dañada su reputación.

El sitio ha sufrido numerosas críticas por estar dirigido por una compañía Change.org,Inc. con ánimo de lucro cuyo modelo de negocio se basa en que empresas u organizaciones les retribuyan a cambio de promoción de sus campañas.

6.2. Avaaz

Avaaz[113] es un sitio de recogida de firmas fundado en 2007 en Nueva York[114], entre sus fundadores se incluyen un ex congresista americano o miembros de diversas organizaciones similares. Cuenta con unos 44 millones de usuarios en el mundo. Permite crear campañas mediante cuentas de usuario y firmarlas sin necesidad de crear una, pero requiere introducir datos personales: un correo electrónico, nombre, país, código postal y país.

El principio de funcionamiento es el mismo que el de Change, se basa en la presión social.

Se mantienen económicamente por medio de donaciones de sus miembros, rechazan donantes corporativos o gubernamentales y cuelgan sus cuentas en su web.[115]

---

110 https://www.change.org/
111 http://www.itespresso.es/change-org-llega-10-millones-usuarios-espana-152893.html
112 https://en.wikipedia.org/wiki/Change.org
113 https://secure.avaaz.org/es/
114 https://en.wikipedia.org/wiki/Avaaz
115 https://secure.avaaz.org/es/about.php

6.3. MiFirma.com

MiFirma.com[116] es un sitio web con base en Madrid, creado en el 2011, se definen como "una plataforma para desarrollar Iniciativas Legislativas Populares a través de la firma digital"[117]. No tienen cuentas de usuarios, solo es necesario registrarse como Comisión Promotora para sacar adelante una ILP o campaña similar, además requiere que la IPL haya sido aprobada ya por la Mesa del Congreso, no se puede iniciar una campaña sin una ILP en trámite.

Se presenta como algo complementario a la recogida de firmas tradicional, ya que como ellos mismos afirman "conseguir 500.000 firmas electrónicas [...] puede resultar extremadamente difícil con el nivel de despliegue (DNIe, lectores, instalaciones) actual.".

Para el proceso de firma se requiere el nombre, apellidos, NIF, fecha de nacimiento y un email de contacto. Con estos datos se genera un fichero XML como el siguiente:

```
<ilp>
    <firmante>
        <nomb>Aythami</nomb>
        <ape1>Estévez</ape1>
        <ape2>Olivas</ape2>
        <fnac>19940601</fnac>
        <tipoid>1</tipoid>
        <id>70918176E</id>
    </firmante>
    <datosilp>
        <tituloilp>
            Proposición de Ley de responsabilidad parental y de relaciones familiares
        </tituloilp>
        <codigoilp>ILP2015125</codigoilp>
    </datosilp>
</ilp>
```

ILUSTRACIÓN 20: MIFIRMA.COM FICHERO XML GENERADO

A continuación si el navegador soporta el plugin de Java para exploradores web[118] se lanza un applet que permite seleccionar un certificado oficial de la Fábrica Nacional de Moneda y Timbre asociado a un DNI instalado en el cliente, tras esto se introduce el pin del DNI electrónico y con esto se firma la propuesta. En caso contrario es necesario usar la aplicación AutoFirma ofrecida en [119], al ir a firmar una petición se despliega dicha aplicación que permite seleccionar un certificado y firmar con él el documento de igual modo que con el applet.

Respecto a su modelo de financiación, MiFirma.com es una asociación sin ánimo de lucro cuyo uso es gratuito. Se mantienen a base de donativos y si alguna campaña tiene éxito cobrarían entre 0,20€ y 0,025€ por firma según su número, este dinero se cobraría

116 https://www.mifirma.com/
117 http://blog.mifirma.com/blog/2011/05/25/lanzamiento-de-mifirma-firma-iniciativas-legislativas-populares-con-tu-dnie/
118 https://www.java.com/es/download/faq/chrome.xml
119 http://firmaelectronica.gob.es/Home/Descargas.html

a la administración, que subsana los gastos de una ILP o recogida de avales que llegue al mínimo de apoyos necesarios.

6.4. Comparativa de sistemas de recogida de firmas.

TABLA 2: COMPARATIVA DE SISTEMAS DE RECOGIDA DE FIRMAS

	Change	**Avaaz**	**MiFirma**	**Demos**
Permiten la creación de campañas a cualquier persona	Sí	Sí	Sí, pero se requiere contactar con los administradores y una campaña legalmente en curso	Sí
Permiten firmar a cualquier persona	Sí	Sí	Sí, pero ha de poder firmar documentos con el certificado de su DNIe	Sí, pero la subida de DNIs ha de ser realizada por un organizador de la campaña
Verifican la identidad de los firmantes	No	No	Sí, mediante certificados electrónicos	No, es tarea del organizador de la campaña que sube la firma
Tienen alguna validez legal	No	No	Sí	Sí, pero han de entregarse las fotografías junto con las hojas manuscritas
Tiene ánimo de lucro	Sí	No	No	No

7. Conclusiones y líneas de trabajo futuras

Si repasamos los objetivos incluidos en este documento se puede ver que la consecución de los objetivos funcionales ha sido lograda. También se ha realizado un esfuerzo por cumplir los objetivos no funcionales, pero esta valoración es subjetiva, ya que algunos como la simplicidad o el rendimiento no se pueden apreciar realmente hasta probar este sistema en un entorno real. Los objetivos personales se han cumplido con creces, se ha podido experimentar el desarrollo de un proyecto siguiendo un modelo de proceso ágil, algo que solo se había estudiado en teoría y que es radicalmente diferente al proceso unificado (modelo de referencia a lo largo de titulación). En relación al mundo del software libre, este proyecto ha requerido explorar múltiples repositorios de proyectos reales, adquiriéndose con ello algunas capacidades que serán muy útiles en el futuro como la capacidad de usar y entender librerías extensas, la capacidad de buscar con soltura en la documentación de APIs públicas o la capacidad de manejar sistemas de control de versiones.

Este proyecto ha servido para estructurar mejor los conocimientos adquiridos a lo largo de la carrera en las diversas asignaturas individuales, ha sido necesario integrar conocimientos de asignaturas tan dispares como la ingeniería del software, la robótica o los sistemas distribuidos o la seguridad en sistemas informáticos.

Las líneas de trabajo futuras que se podrían seguir en este proyecto pasan por la mejora de alguno de sus aspectos finales, para lo que no ha habido tiempo, como la recuperación de DNIs principalmente que es la parte más ineficiente del proceso, también se podría optimizar el proceso OCR. Ambos procesos dependen de sus respectivas librerías (PDFBox y Tessteract respectivamente) y su optimización incluiría la búsqueda de otras alternativas prestando especial atención a aquellas de software privativo, ya que aunque no se ha verificado, se ha encontrado información en múltiples sitios recomendando alternativas privativas para entornos reales de explotación debido a sus mejores rendimientos. También convendría acompañar todo el proyecto de una página web que informe del proyecto y permita a los usuarios informarse de la utilidad del sistema o que pasa con sus imágenes.

Ha resultado especialmente interesante el proyecto MiFirma, que al igual que este, pretende ayudar y complementar la recogida de firmas con identificación, siendo ambos proyectos integrables entre sí y permitiendo ofrecer a los organizadores de una recogida de firmas un conjunto de herramientas para su automatización. Estas herramientas van desde una plataforma para las firmas electrónicas como es MiFirma, a una herramienta para el apoyo de la recogida de firmas en la calle como es este proyecto, pasando por asesoría técnica y jurídica sobre estos procesos.

Bibliografía

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine.

García Peñalvo, F. J. (2014). Modelos de proceso, Ingeniería del Software I. *Universidad de Salamanca*, 64.

García Peñalvo, F. J., & Moreno García, M. N. (2000). Proyecto de FInal de Carrera en la Ingeniería Técnica en Informática: Guía de Realización y Documentación.

Smith, R. (2007). An Overview of the Tesseract OCR Engine. *IEEE 0-7695-2822-8/07*, 629-633.

Smith, R. (2007b). Tesseract OCR Engine. What it is, where it came from, where it is going. *OSCON*.

Sommerville, I. (2005). *Software Engineering*.

Suzuki, S., & Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *CVGIP 30 1*, 32-46.

Szeliski, R. (2010). En *Computer Vision: Algorithms and Applications* (pág. 5). Springer.

Apéndice A: Fondo de fotografía recomendado

Por favor, sitúe el DNI dentro del siguiente recuadro para realizar las fotografías.