



*Facultad
de
Ciencias*

**APLICACIÓN MÓVIL PARA LA VALIDACIÓN DE
DOCUMENTOS DE IDENTIDAD**
(Mobile application for identity card validation)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Marcos López Martínez

Director: Carlos Blanco Bueno

Co-Director: Verónica Román Morante

Junio - 2015

Resumen

En los últimos años el uso de tecnologías móviles se ha disparado. En 2014 se ha llegado a cifras tan desorbitantes como 10 Smartphones o Tablets vendidos por cada equipo tradicional, sumando un total de 1244,89 millones de dispositivos. Siguiendo estas cifras, una aplicación móvil tendría hasta 10 veces más clientes potenciales que una convencional y por este motivo, el desarrollo móvil está creciendo paralelamente a las ventas.

Dentro de este relativamente nuevo mercado surge la idea de este proyecto, que pretende aprovecharse de las características presentes en los dispositivos móviles para ofrecer una aplicación de gran utilidad. La aplicación en cuestión, tiene como objetivo principal facilitar la validación del DNI electrónico español y la extracción de sus datos. Además, se pretende almacenar estos datos después de su extracción, pudiendo ser posteriormente utilizados por aplicaciones de terceros. Este proyecto se ha realizado en colaboración con la empresa “Netkia Soluciones SL.”, que ha ofrecido los medios necesarios para el desarrollo de la aplicación.

Para lograr esto se hace uso de la cámara presente en prácticamente todos los dispositivos móviles y de las tecnologías proporcionadas por las librerías OpenCV y Tesseract OCR, que permiten reconocer objetos en imágenes y reconocer caracteres respectivamente. Además, la aplicación se ha implementado con ayuda del framework de desarrollo móvil Phonegap, que facilita su portabilidad entre las diferentes plataformas.

Palabras clave: Aplicación móvil, Phonegap, Reconocimiento de imágenes, Reconocimiento de caracteres, Validación DNI.

Abstract

In the last years the use of mobile technologies has increased significantly. In 2014 we have reached numbers such as 10 Smartphones or Tablets sold for every traditional PC, with a total of 1244,89 million devices. Following this numbers, a mobile application would have 10 times more potential clients than a conventional app and for this reason, mobile development is rising parallel to sales.

Within this relatively new market arises the idea of this project, which pretends to benefit from the features present in the mobile devices to offer a really useful application. The main objective of the application is to ease the validation of the electronic spanish identity card and the extraction of its data. Also, it is intended to store this data after its extraction, allowing third-party applications to use them. This project has been developed in collaboration with "Netkia Soluciones S.L", which has offered the necessary means to implement the application.

To archive this, the application uses the camera present in practically all mobile devices and the technologies provided by the libraries OpenCV and Tesseract OCR, which permit recognizing objects in pictures and recognizing characters respectively. Also, the application has been implemented using the mobile development framework Phonegap, which ease its portability between different platforms.

Keywords: mobile application, Phonegap, Image recognition, Character recognition, DNI Validation.

Índice

1. Introducción y objetivos.....	8
1.1 Introducción	8
1.2 Objetivos	8
2. Material y métodos utilizados.....	9
2.1 Metodología	9
2.2 Planificación	10
2.3 Tecnologías.....	11
2.3.1 Phonegap.....	11
2.3.2 HTML5	12
2.3.3 CSS3	12
2.3.4 Javascript.....	13
2.3.5 Java	13
2.3.6 Objective-C.....	14
2.3.7 Clasificadores Haar.....	14
2.3.8 OpenCV.....	16
2.3.9 Tesseract OCR.....	16
2.3.10 Android SDK e iOS SDK	17
2.3.11 Node.js	18
2.4 Herramientas.....	18
2.4.1 Eclipse IDE	18
2.4.2 Xcode.....	19
2.4.3 GThumb.....	20
2.4.4 Sublime Text 2	21
2.4.5 MySQL Workbench.....	22
2.4.6 Herramientas utilizadas para las pruebas	22
3. Análisis de requisitos.....	23
3.1 Requisitos funcionales.....	23
3.2 Diagramas de casos de uso y plantillas	23
3.3 Requisitos no funcionales	25
4. Diseño.....	26
4.1 Diseño arquitectónico	26
4.2 Diseño detallado	27
5. Implementación	29

5.1 Entrenamiento de la Cascada Haar	29
5.1.1 Preparación de las muestras negativas.....	29
5.1.2 Preparación de las muestras positivas.....	30
5.1.3 Entrenamiento de la cascada	32
5.2 Implementación de la capa de presentación	34
5.2.1 Interfaz móvil	34
5.2.2 Interfaz web	38
5.3 Implementación de la capa de negocio	41
5.3.1 Parte Javascript	41
5.3.2 Parte nativa	42
5.4 Implementación de la capa de acceso a datos.....	45
5.4.1 Base de datos MySQL.....	45
5.4.2 API REST.....	45
5.4.3 Aplicación web	46
6. Pruebas.....	48
6.1 Pruebas unitarias.....	48
6.2 Pruebas de integración	49
6.3 Pruebas de sistema	49
6.3.1 Pruebas de rendimiento.....	49
6.3.2 Pruebas de seguridad	50
6.5 Pruebas de aceptación	50
7. Conclusiones y trabajos futuros	51
7.1 Conclusiones.....	51
7.2 Trabajos futuros	51
7.2.1 Mejora del reconocimiento de caracteres	51
7.2.2 Implementación de la versión de Windows Phone.....	52
7.2.3 Mejora de la interfaz gráfica	52
8. Bibliografía	53

Índice de ilustraciones

Ilustración 1: Método iterativo e incremental	9
Ilustración 2: Logotipos de Apache Cordova, Phonegap y Phonegap Build.....	11
Ilustración 3: Logotipos de HTML5, CSS3 y Javascript	12
Ilustración 4: Logotipos de Java y Oracle	14
Ilustración 5: Logotipo de Objective-C	14
Ilustración 6: Filtros Haar	15
Ilustración 7: Ejemplo de uso de los clasificadores Haar	15
Ilustración 8: Logotipo de OpenCV.....	16
Ilustración 9: Reconocimiento de caracteres con Tesseract OCR.....	17
Ilustración 10: Logotipo de Node.js	18
Ilustración 11: Interfaz de Eclipse	19
Ilustración 12: Interfaz de Xcode	20
Ilustración 13: Interfaz de GThumb	21
Ilustración 14: Interfaz de Sublime Text 2.....	21
Ilustración 15: Diagrama de casos de uso	24
Ilustración 16: Arquitectura en 3 capas	26
Ilustración 17: Diagrama de componentes del sistema.....	27
Ilustración 18: Diagrama de navegación	27
Ilustración 19: Definición de interfaces	28
Ilustración 20: Clase Documento	28
Ilustración 21: Una de las imágenes utilizadas para las muestras negativas.....	30
Ilustración 22: Una de las imágenes utilizadas para las muestras positivas.....	30
Ilustración 23: Ejemplo de uso de la utilidad createsamples.....	31
Ilustración 24: Ejemplo de uso de la utilidad traincascade.....	33
Ilustración 25: Vista del menú principal de la aplicación Android	35
Ilustración 26: Vista del menú principal de la aplicación iOS.....	35
Ilustración 27: Vista del inicio de la validación en Android.....	35
Ilustración 28: Vista del inicio de la validación en iOS	35
Ilustración 29: Vista de los datos capturados del frente en Android.....	36
Ilustración 30: Vista de los datos capturados del frente en iOS	36
Ilustración 31: Vista del resultado de la validación en Android.....	36
Ilustración 32: Vista del resultado de la validación en iOS	36
Ilustración 33: Vista del historial en Android	37
Ilustración 34: Vista del historial en iOS.....	37
Ilustración 35: Vista de una entrada el historial en Android	37
Ilustración 36: Vista de una entrada el historial en iOS	37
Ilustración 37: Vista de las instrucciones en Android	38
Ilustración 38: Vista de las instrucciones en iOS	38
Ilustración 39: Vista principal de la aplicación web	39
Ilustración 40: Vista de una entrada de la lista	39
Ilustración 41: Vista de los gráficos de estadísticas	40
Ilustración 42: Vista la lista de elementos filtrados	40
Ilustración 43: Parte del código de extracción del DNI y reconocimiento de los caracteres.....	41
Ilustración 44: Parte del código de validación	42
Ilustración 45: Parte del código de la clase OpenCVPlugin en Java	43

Ilustración 46: Parte del código de la clase OpenCVPlugin en Objective-C	43
Ilustración 47: Parte del código de la clase TesseractPlugin en Java	44
Ilustración 48: Parte del código de la clase TesseractPlugin en Objective-C	44
Ilustración 49: Parte del código de la API REST	46
Ilustración 50: Parte del código de la aplicación web	47
Ilustración 51: Parte del código de las pruebas unitarias	48

1. Introducción y objetivos

1.1 Introducción

Gracias a la rápida y enorme expansión de las plataformas móviles como complemento o sustituto de los equipos tradicionales, las aplicaciones para Smartphones están poco a poco estableciéndose como uno de los campos más importantes dentro del desarrollo de software. Partiendo de esto y teniendo muy en cuenta los avances en las tecnologías de reconocimiento de objetos y caracteres, se puede dar lugar a muchas ideas interesantes.

En concreto, la idea de este proyecto pretende aprovecharse de lo anteriormente mencionado para tratar de ofrecer una solución al siguiente problema: se ha observado que existen diversas situaciones en las que se presenta la necesidad de validar documentos de identidad y extraer los datos contenidos en los mismos.

Por ejemplo, es común validar el DNI en la fase previa a contrataciones de algunos servicios o tomar los datos de los clientes después de ello. Gracias a esta aplicación, un empleado de un comercio podría validar el DNI de un cliente y a su vez guardar sus datos en una base de datos propia simplemente sacando dos fotos con su Smartphone.

1.2 Objetivos

Teniendo en cuenta lo anteriormente expuesto, el principal objetivo del proyecto es el de desarrollar una aplicación funcional en las plataformas móviles más importantes que ofrezca la posibilidad de validar documentos de identidad y obtener los datos que contienen. Como objetivos específicos de este principal tendríamos:

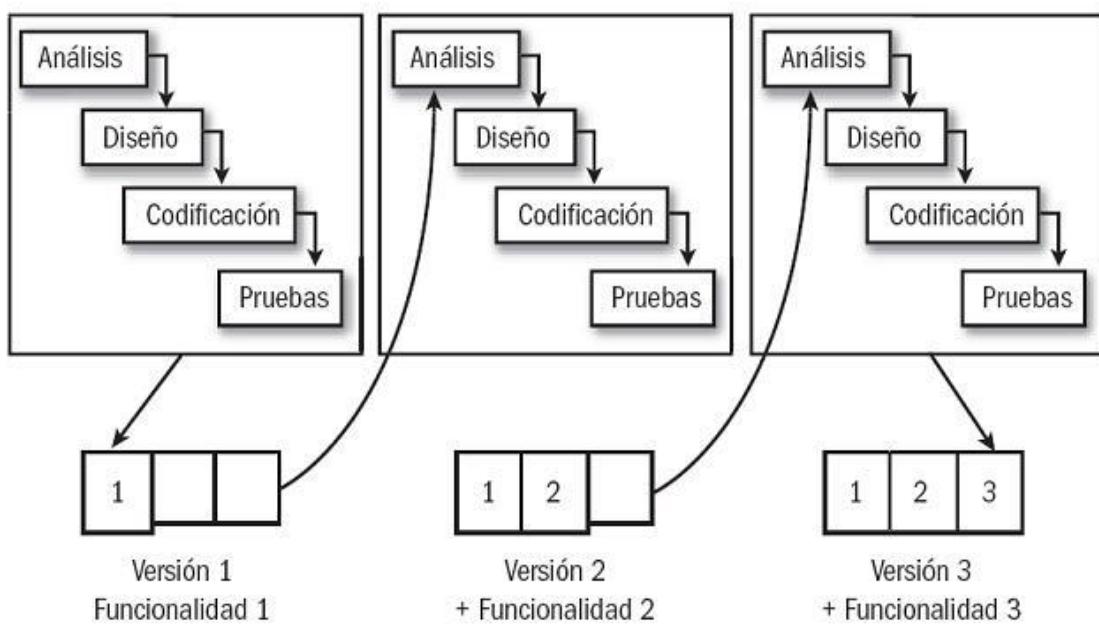
1. Diseñar e implementar la aplicación para las dos principales plataformas móviles: Android e iOS.
2. Lograr una precisión considerable tanto en el reconocimiento del DNI como en la posterior lectura de caracteres.
3. Diseñar una interfaz sencilla e intuitiva que facilite el uso de la aplicación.
4. Ofrecer la posibilidad de utilizar los datos extraídos por aplicaciones de terceros.
5. Realizar una validación lo más precisa posible, que en ningún caso dé como válido un documento que no lo sea.
6. Almacenar los datos extraídos en una base de datos de manera que estén disponibles siempre que se requiera.
7. Diseñar e implementar una aplicación web que permita ver los datos almacenados en la base de datos.

2. Material y métodos utilizados

En este apartado se detallan las tecnologías, herramientas y lenguajes de programación utilizados en el desarrollo de este proyecto, así como la metodología seguida y la planificación del proyecto.

2.1 Metodología

Para el desarrollo de este proyecto se ha optado por seguir una metodología iterativa e incremental. Este modelo se basa en dividir el proyecto en varios bloques temporales o iteraciones. En cada iteración se repiten las fases de análisis de requisitos, diseño, implementación y pruebas. El objetivo principal de este modelo es ofrecer al cliente entregas funcionales del producto de manera constante, lo que permite que este lo pruebe y proporcione retroalimentación.



Esta metodología soporta mucho mejor los cambios y reduce considerablemente los riesgos con respecto al modelo en cascada tradicional. En las primeras iteraciones del desarrollo, se implementan las funcionalidades de mayor importancia, mientras que los detalles se dejan para las últimas. Cada iteración permite ofrecer un producto con mayor funcionalidad que la anterior, cada vez más cercano a la versión final. A continuación se provee un resumen de cada iteración:

- 1) En la primera iteración del proyecto se ha desarrollado una aplicación que permite sacar una fotografía de la parte frontal del DNI, detecta el documento en ella y muestra el resultado recortado. Después se repite el proceso para el reverso del DNI. Esta versión no tiene capa de acceso a datos y posee una interfaz muy primitiva y simple, además de funcionar solo en Android.

- 2) En la segunda iteración se ha añadido la funcionalidad de reconocimiento de texto. Después de detectar el DNI en la imagen, lee los diferentes campos en él y muestra el resultado.
- 3) En la tercera iteración se ha añadido la validación del DNI. Se realiza después del reconocimiento de texto y comprueba la letra del DNI, los números de control, la fecha de vencimiento y que los campos del frente sean iguales a los del reverso.
- 4) En la cuarta iteración se ha añadido la versión de iOS, que contiene la misma funcionalidad que la tercera iteración.
- 5) En la quinta iteración se ha añadido la interfaz de usuario.
- 6) En la sexta iteración se ha añadido la capa de acceso a datos. La aplicación interactúa con la base de datos a través de un servicio REST.
- 7) En la séptima iteración se ha desarrollado una simple aplicación web que permite ver los datos alojados en la base de datos desde un navegador web.

2.2 Planificación

Tarea	Duración	Comienzo	Fin
Instalación y configuración de las herramientas necesarias para desarrollar la aplicación.	3 días	05/02/15	09/02/15
Aprendizaje de entrenamiento de cascadas y lectura de caracteres.	4 días	10/02/15	13/02/15
Entrenamiento del Clasificador de Cascada.	7 días	16/02/15	24/02/15
Aprendizaje de desarrollo en Phonegap.	2 días	25/02/15	26/02/15
Desarrollo de la aplicación Android.	9 días	27/02/15	11/03/15
Aprendizaje de Objective-C y desarrollo en iOS.	2 días	12/03/15	13/03/15
Desarrollo de los plugins nativos para iOS de OpenCV y Tesseract.	6 días	16/03/15	23/03/15
Realización de las pruebas.	2 días	24/03/15	25/03/15
Realización de la interfaz de la aplicación.	4 días	26/03/15	31/03/15
Desarrollo del backend de la aplicación.	11 días	01/04/15	15/04/15
Optimización de la aplicación.	11 días	16/04/15	29/04/15
Realización de la memoria del proyecto.	13 días	30/04/15	18/05/15
Total	74 días	592 horas	

2.3 Tecnologías

2.3.1 Phonegap

Phonegap^{[2][3]} es un framework de código abierto para el desarrollo de aplicaciones móviles multiplataforma que actualmente es propiedad de Adobe Systems. Es una distribución de Apache Cordova, un framework gratuito y de código abierto bajo la Licencia Apache 2.0.

Su principal ventaja es que las aplicaciones desarrolladas con Phonegap pueden ser portadas a diversas plataformas con cambios mínimos o nulos entre ellas. Esto es gracias a que se crean utilizando tecnologías web estándar tales como HTML5, CSS3 y Javascript. Aunque generalmente no requieren el uso de código nativo, existen como excepción aplicaciones con funciones muy concretas que utilizan plug-ins nativos para acceder a dichas funciones.



Ilustración 2: Logotipos de Apache Cordova, Phonegap y Phonegap Build

Fue creado por la empresa Nitobi Software en un IphoneDevCamp organizado por Apple en 2008 y posteriormente adquirido por Adobe Systems en 2011. Poco después, Adobe donó el código de Phonegap a la Fundación Apache bajo el nombre de Apache Cordova.

Posee una serie de librerías Javascript que permiten el acceso a características de los dispositivos como el acelerómetro, la cámara, el sistema de ficheros o las notificaciones. Además, actualmente está disponible para iOS, Android, Blackberry, Windows Phone, Palm WebOS, Symbian y Bada.

Como añadido a la gran portabilidad, también tiene la ventaja de que las aplicaciones se pueden instalar desde las tiendas específicas de cada dispositivo, ya que se empaquetan utilizando los SDK propios de cada plataforma.

A pesar de todas las cosas buenas que ofrece tiene una serie de desventajas. Para empezar, los entornos de desarrollo son diferentes dependiendo del sistema al que esté orientada la aplicación, Eclipse en Android, Xcode en IOS, Visual Studio en Windows Phone, etc. Además, existe cierta controversia con este tipo de aplicaciones debido a que ofrecen experiencias de usuario menos satisfactorias que las aplicaciones realizadas de forma nativa y generalmente están menos optimizadas.

En este proyecto se han desarrollado la versiones Android e iOS de la aplicación en un PC con Ubuntu y un Mac con OS X respectivamente. Posteriormente se han probado en dispositivos de cada una de las dos plataformas. Se ha elegido Phonegap porque a pesar de sus desventajas con respecto a las aplicaciones nativas, facilita mucho la labor de realizar una aplicación multiplataforma.

2.3.2 HTML5

HTML5^[4] (HyperText Markup Language 5) es la quinta versión del lenguaje de marcado HTML, un estándar del W3C con el que se maquetan las páginas web. Permite especificar una estructura básica que define el contenido de una web tal como texto, enlaces, imágenes, etc. HTML5 añade una serie de nuevos elementos y API's con respecto a sus versiones anteriores a la vez que elimina algunos obsoletos.



Ilustración 3: Logotipos de HTML5, CSS3 y Javascript

HTML fue creado por Tim Berners-Lee en 1991, dos años después de que él mismo crease la World Wide Web. Entre ese año y 1998 continuó su desarrollo, pero en el año 2000 el W3C empezó a recomendar XHTML como alternativa.

En 2008 fue presentado al público HTML5 por parte del WHATWG (Web Hypertext Application Technology Working Group), un grupo creado en respuesta al lento desarrollo del W3C y a su decisión de cerrar el desarrollo de HTML en favor de XHTML. Finalmente, el W3C empezó a recomendar HTML5 en octubre de 2014.

El principal problema de HTML5 es que no es reconocido por las versiones viejas de navegadores debido a sus nuevas etiquetas. A pesar de esto, poco a poco se va estableciendo como la versión de HTML más utilizada en el desarrollo web y se recomienda encarecidamente a los usuarios actualizar sus navegadores a la versión más reciente para poder disfrutar de todo su potencial.

2.3.3 CSS3

CSS3^[4] (Cascading Style Sheets 3) es la tercera versión del lenguaje de hojas de estilo en cascada CSS, un estándar del W3C con el que se define el estilo de un documento HTML. Permite dar formato a los elementos de una web, separando su estructura (HTML) de su presentación (CSS).

La primera versión de CSS fue publicada por el W3C a finales de 1996 y se amplió a CSS2 a mediados de 1998. Más adelante se publicó una revisión de este último conocida como CSS2.1, que corrige algunos errores, eliminaba funcionalidades obsoletas y añadía una nueva especificación. CSS3 empezó a desarrollarse en 1999, pero a diferencia de las versiones anteriores está dividido en varios módulos separados, motivo por el cual los diferentes módulos están en diferentes grados de desarrollo.

A día de hoy todavía presenta varias limitaciones, como la dificultad del alineamiento vertical o la falta de expresiones de cálculo numérico para expresar valores. Sin embargo, aporta 3 principales ventajas: permite tener un control centralizado sobre la presentación de un sitio web, optimiza el ancho de banda al poderse definir un solo selector CSS que abarque varios elementos y mejora la accesibilidad al evitar prácticas poco recomendables como las tablas.

2.3.4 Javascript

Javascript^[4] ^[5] es un lenguaje de programación interpretado que se utiliza principalmente para desarrollar sitios web dinámicos. No requiere compilación ya que las instrucciones se interpretan y ejecutan directamente en el navegador web. Gracias a esto tiene la ventaja de ser un lenguaje independiente de la plataforma y no necesita ningún programa a parte del navegador para su ejecución.

Javascript surgió de la necesidad de crear páginas web que permitiesen interactuar con los usuarios, ya que en principio HTML solo permitía desarrollar páginas estáticas. Fue creado por Netscape en 1990 bajo el nombre de LiveScript y posteriormente renombrado a Javascript en 1995 cuando Netscape y Sun Microsystems (creadores de Java) se unieron para desarrollar el proyecto.

Es un lenguaje usado principalmente, aunque no exclusivamente, de lado del cliente y permite mejoras en la interfaz de usuario y páginas web más dinámicas. A pesar de tener convenciones propias de Java, es un lenguaje totalmente diferente a éste y tiene una sintaxis más similar a C. A día de hoy todos los navegadores modernos permiten interpretar Javascript, lo que unido a su sencillez, rapidez y ligereza lo hace uno de los lenguajes más utilizados en el ámbito web.

2.3.5 Java

Java^[12] es un lenguaje de programación orientado a objetos y de propósito general desarrollado por Sun Microsystems y actualmente propiedad de Oracle. Su principal ventaja es que su código puede ser ejecutado en cualquier dispositivo debido a que se ejecuta sobre una máquina virtual (JVM, Java Virtual Machine) en vez de hacerlo directamente sobre el sistema. Esta ventaja es a la vez uno de sus mayores inconvenientes, ya que ofrece un peor rendimiento que otros lenguajes de programación.



Ilustración 4: Logotipos de Java y Oracle

Su sintaxis está basada en C y C++, pero está diseñado para ser más fácil de usar. Esta facilidad de uso e independencia de la plataforma, junto con otras características, hacen de Java uno de los lenguajes de programación más utilizados del mundo tanto en desarrollo como en enseñanza. Además es el lenguaje utilizado para el desarrollo de aplicaciones Android, por lo que se ha utilizado para la parte nativa Android de la aplicación.

2.3.6 Objective-C

Es un lenguaje de programación orientado a objetos utilizado principalmente en aplicaciones de iOS y OS X. Fue creado originalmente por Brad Cox y StepStone en 1980, adoptado como lenguaje de programación de NEXTSTEP en 1988 y liberado bajo licencia GPL en 1992. La parte nativa iOS del proyecto está realizada en este lenguaje.



Ilustración 5: Logotipo de Objective-C

Se ha elegido por encima de su principal alternativa Swift debido a que esta última todavía es muy reciente y se va a dar soporte a Objective-C durante varios años más. Su sintaxis está basada en C y C++ y mantiene elementos de éstos como las estructuras y punteros, que no están presentes en otros lenguajes como Java. Aunque a primera vista es bastante diferente a otros lenguajes, resulta bastante sencillo e intuitivo.

2.3.7 Clasificadores Haar

Los Clasificadores Haar son un método ampliamente utilizado para el reconocimiento de objetos en imágenes digitales y permiten detectar casi cualquier elemento dentro de ellas. Fueron

desarrollados por Paul Viola y Michael Jones, que en principio los idearon para la detección de rostros, y posteriormente ampliados por Rainer Lienhart y Jochen Maydt.

Viola y Jones demostraron como se podía desarrollar un detector rostros bastante eficiente a partir de los cambios de intensidad entre las diferentes zonas de la imagen. Utilizaron los llamados filtros Haar, que consisten en aplicar filtros de diversos tamaños que recorren la imagen buscando valores por encima de un cierto umbral, los cuales clasifica como rostro.

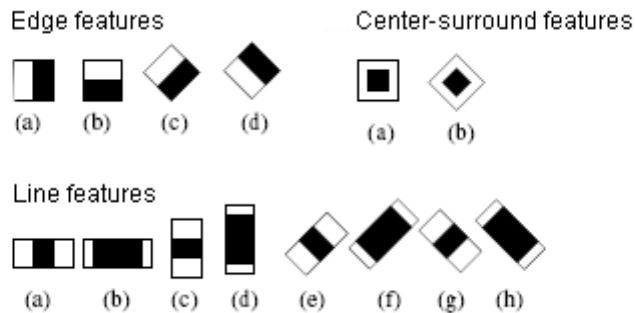


Ilustración 6: Filtros Haar

Los clasificadores obtenidos a partir de un solo filtro (también conocidos como débiles) por si solos no proporcionan muy buenos resultados, pero combinando varios de ellos se puede conseguir un clasificador notablemente más robusto con resultados mucho más fiables. A esta técnica de combinar varios clasificadores simples para formar uno más preciso se le conoce como Boosting y permite llegar a tasas del 99.9% de aciertos. Sin embargo, presenta una tasa de falsos positivos cercana al 50%.

Para solventar este problema, propusieron realizar la detección mediante una cascada o sucesión de clasificadores robustos, conocida como Cascada Haar. De esta forma, para cascadas de 20 etapas se pueden conseguir tasas de acierto cercanas al 98% y tasas de falsos positivos del orden de 0.0001%. Mientras que en las primeras etapas de la cascada se rechazan las zonas con más diferencias con el rostro, es en las últimas etapas cuando se rechazan las zonas más complejas.

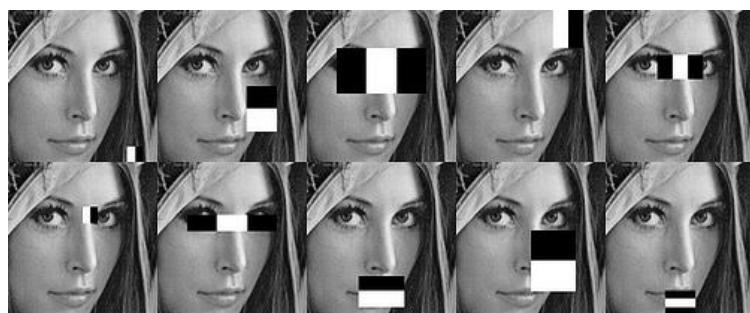


Ilustración 7: Ejemplo de uso de los clasificadores Haar

La técnica del Boosting fue posteriormente mejorada con lo que se conoce como Adaboost (Adaptative Boosting), que se ajusta adaptativamente a los resultados obtenidos en las primeras etapas de la cascada, atribuyendo pesos a los clasificadores débiles en función de su efectividad.

De la misma manera que se puede utilizar para detectar rostros, es posible entrenar Cascadas Haar para reconocer otros objetos dentro de las imágenes. Durante este proyecto se han entrenado dos Cascadas Haar diferentes, una para la detección de la parte frontal del DNI y otra para el reverso. Los pasos a seguir para el entrenamiento de una cascada Haar serán descritos más adelante en el apartado correspondiente.

2.3.8 OpenCV

OpenCV^[6] (Open Source Computer Vision Library) es una librería de código abierto de visión artificial en tiempo real desarrollada por Intel. Contiene más de 2500 algoritmos, entre los que se incluyen: detección y reconocimiento de objetos, seguimiento del movimiento, corrección de ojos rojos o visión robótica. Es ampliamente utilizada por compañías, grupos de investigación y entidades gubernamentales.



Ilustración 8: Logotipo de OpenCV

Tiene interfaces C++, C, Python, Java y MATLAB y soporta Windows, Linux, OS X, Android e IOS. Su licencia BSD permite que sea utilizada libremente para propósitos comerciales y de investigación bajo las condiciones detalladas en ella. Estos son los principales motivos, junto con su facilidad de uso, por los que se ha elegido en lugar de otras librerías.

Entre todos sus algoritmos se encuentran los de aprendizaje y clasificadores, como el Adaboost y las Cascadas Haar, los cuales se utilizan en este proyecto para la detección del DNI dentro de la fotografía. Además, se han utilizado otros algoritmos de esta librería para procesar la imagen con el propósito de mejorar la efectividad de la lectura de caracteres que se realiza posteriormente.

2.3.9 Tesseract OCR

Tesseract OCR^[7] es una librería de código abierto para el reconocimiento óptico de caracteres (OCR). Fue desarrollada entre 1985 y 1995 por Hewlett Packard y posteriormente liberada como código abierto en 2005. Actualmente la desarrolla Google bajo la Licencia Apache 2.0.

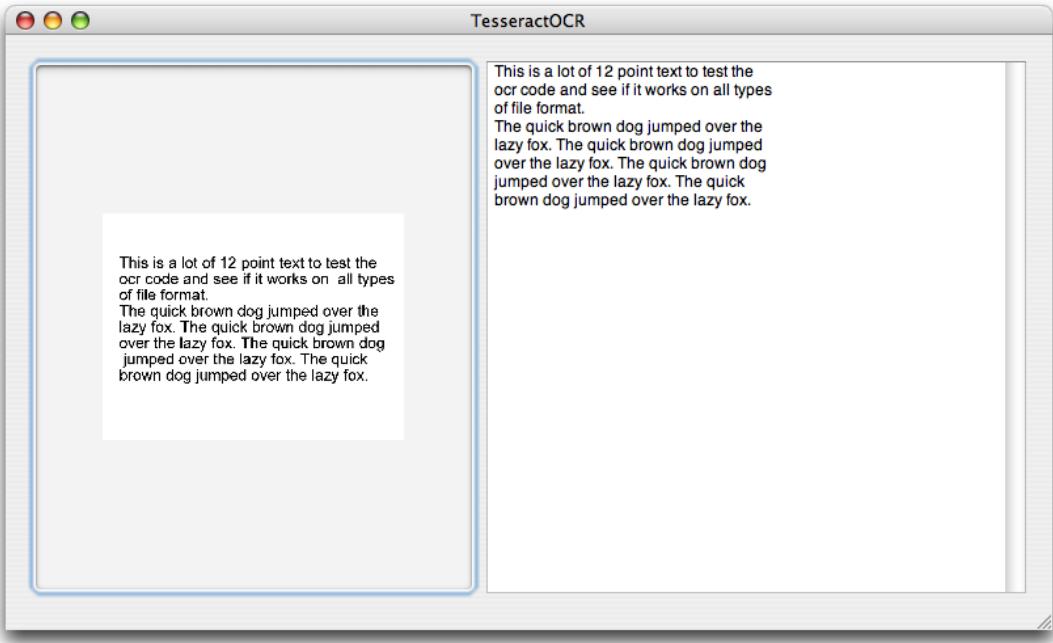


Ilustración 9: Reconocimiento de caracteres con Tesseract OCR

Es uno de los mejores motores OCR en cuanto a precisión y funciona con una amplia lista de idiomas, entre los que se incluyen inglés, español, italiano, francés o alemán, además de poder ser entrenado para reconocer otros idiomas. A pesar de que no tiene versiones oficiales para plataformas móviles existen proyectos de terceros que implementan esta funcionalidad. Para la versión de Android se ha utilizado Tess-two y para la de IOS Tesseract-OCR-iOS, disponibles en los siguientes repositorios GitHub:

Tess-two: <https://github.com/rmtheis/tess-two>

Tesseract-OCR-iOS: <https://github.com/gali8/Tesseract-OCR-iOS>

2.3.10 Android SDK e iOS SDK

Si bien la mayor parte del proyecto está implementado con Phonegap, este no permite la interacción con librerías externas como OpenCV y Tesseract OCR. Para este propósito ha sido necesario el desarrollo de plug-ins que conecten Phonegap con los SDKs nativos de Android^[8] e iOS^[9] y de esta manera se puedan utilizar todas las funciones provistas por estas librerías. Mientras que la parte nativa de Android se ha realizado con Java, la parte de iOS está implementada en Objective-C.

Ambos SDK ofrecen una serie de herramientas y API's que permiten desarrollar de forma nativa para sus respectivas plataformas. Cabe destacar que ya que la aplicación hace uso de la cámara, ha sido necesario probarla en un dispositivo Android y otro iOS, en lugar de hacerlo en los simuladores proporcionados junto con estos SDKs.

2.3.11 Node.js

Node.js^[10] es un entorno de programación de lado del servidor basado en el motor Javascript V8 de Google. Es totalmente asíncrono y utiliza una arquitectura orientada a eventos, lo que hace que sea muy útil en la creación de programas de red altamente escalables y aplicaciones en tiempo real.



Ilustración 10: Logotipo de Node.js

Se ha usado para el desarrollo de la aplicación web que permite acceder a los datos del histórico y del servicio REST que hace de intermediario entre la base de datos y la aplicación móvil. En concreto se ha utilizado el framework Express.js^[11]. Se ha elegido sobre PHP debido a que necesita menos recursos, funciona más rápido y por lo tanto ofrece un mejor rendimiento general. Además, al ser Javascript ha permitido desarrollar la parte del backend en el mismo lenguaje que la aplicación móvil.

2.4 Herramientas

2.4.1 Eclipse IDE

Es un entorno de desarrollo multiplataforma de propósito genérico cuya funcionalidad puede ser ampliamente extendida por medio de plug-ins. A pesar de que no está pensado en concreto para ningún lenguaje de programación, tiene una gran popularidad entre los desarrolladores Java. Este es el motivo por el que se ha elegido para el desarrollo de la parte nativa de Android y se ha aprovechado también para desarrollar las partes Javascript, HTML y CSS de la aplicación.

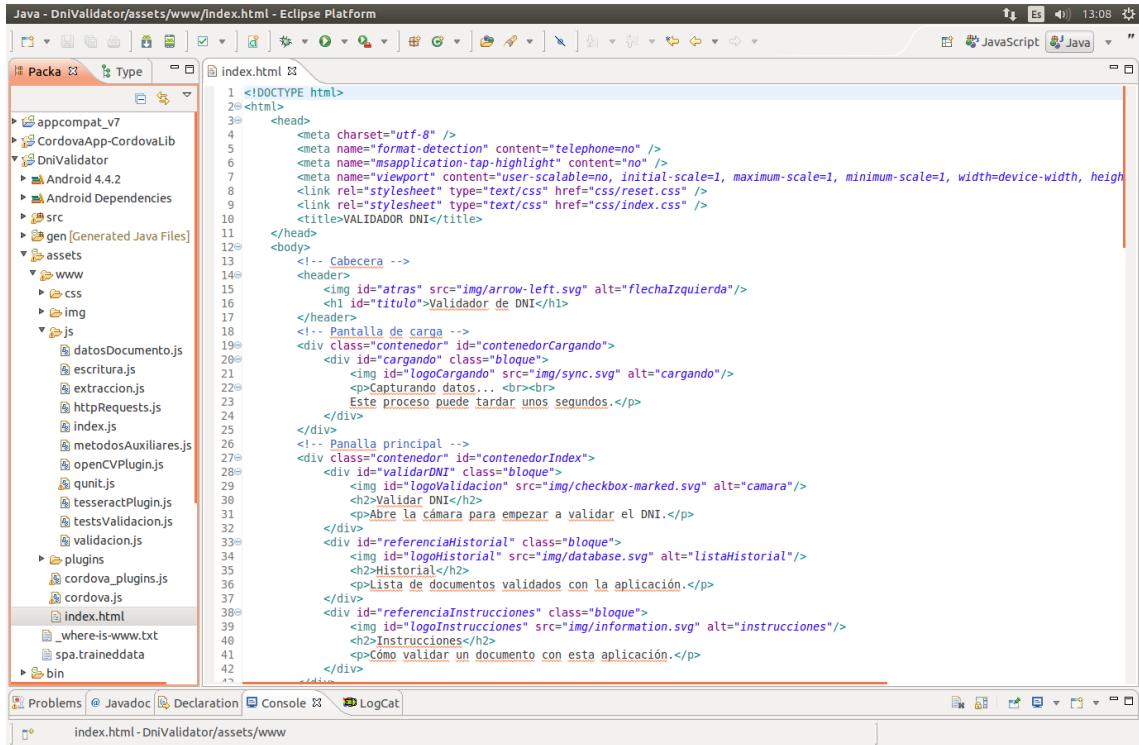


Ilustración 11: Interfaz de Eclipse

Posee diversos editores y vistas que nos permiten adaptar el entorno a nuestro gusto, optimizando el desarrollo. Se basa en proyectos, que son conjuntos de recursos relacionados entre sí como pueden ser código, documentación o ficheros. Además incluye un depurador de código potente y fácil de usar con una vista específica para él.

2.4.2 Xcode

Es un entorno de desarrollo integrado de Apple que permite desarrollar aplicaciones para iOS y OS X. A diferencia de Eclipse que es de propósito más general, está enfocado principalmente a estos dos tipos de aplicaciones, lo cual mejora su eficiencia y facilidad de uso. Su principal desventaja es que solo se puede usar en equipos con OS X y por lo tanto ha sido necesario un Mac para el desarrollo de la parte nativa iOS de la aplicación.

```

68     pluginResult =[CDVPluginResult resultWithStatus:CDVCommandStatus_ERROR messageAsString:@"--(!)Error cargando imagen."];
69 }
70 else{
71     pluginResult =[CDVPluginResult resultWithStatus:CDVCommandStatus_ERROR messageAsString:@"--(!)Error cargando cascada."];
72 }
73 [self.commandDelegate sendPluginResult:pluginResult callbackId:command.callbackId];
74 }
75 /**
76 * Detecta el DNI dentro de la imagen, lo recorta y extrae los campos de interés.
77 * @param Mat imagen - Imagen completa en formato Mat.
78 * @param String tipo - Indica si es la imagen de la parte frontal o del reverso del DNI.
79 */
80 - (void) detectarYCortar:(cv::Mat) imagen deTipo:(NSString*) tipo{
81     std::vector<cv::Rect> dnsDetectados;
82     cascada.detectMultiScale(imagen, dnsDetectados, 1.05, 1, 0|CV_HAAR_SCALE_IMAGE, cv::Size(imagen.cols*0.30, imagen.rows*0.30));
83     cv::Mat imagenCortada = imagen.clone();
84     if(dnsDetectados.size()>0){
85         cv::Rect rectanguloDNI = dnsDetectados[dnsDetectados.size()-1];
86         cv::Mat imagen, rectanguloDNI).copyTo(imagenCortada);
87     }
88     [imagenes removeAllObjects];
89     [self anadirImagen:imagenCortada conLlave:@"0"];
90     cv::Mat imagenPreprocesada = [self preprocesar:imagenCortada deTipo:tipo];
91
92     if([tipo isEqualToString:@"frente"]){
93         //Recorta los campos del frente del DNI.
94     }
95 }
96
97
98
99
100

```

Ilustración 12: Interfaz de Xcode

2.4.3 GThumb

GThumb es un visor, organizador y editor de imágenes libre. Ofrece la funcionalidad necesaria para editar imágenes, convertirlas a diferentes formatos, recortar o redimensionar.

Se ha utilizado para la preparación de las muestras positivas necesarias para el entrenamiento de la Cascada Haar. Estas muestras, como se explica más adelante, deben tener las mismas proporciones y un tamaño similar. Se ha elegido sobre otras alternativas como Adobe Photoshop o Gimp debido a que es mucho más sencillo y ofrece una funcionalidad más que suficiente para el presente proyecto. Además, permite trabajar con cantidades relativamente grandes de imágenes de manera mucho más cómoda.

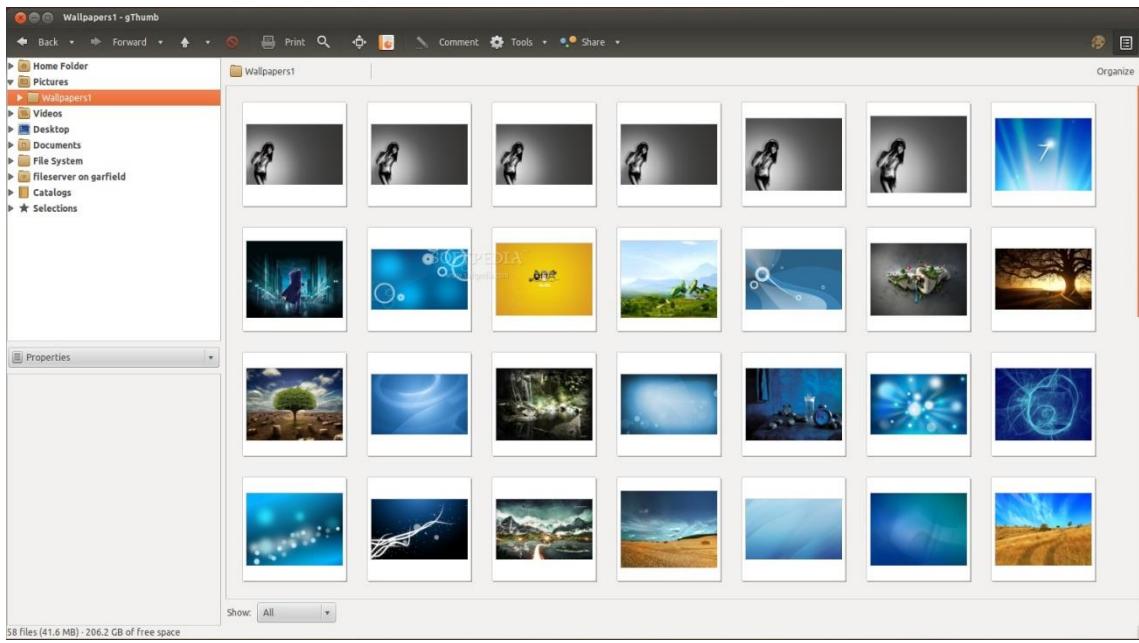


Ilustración 13: Interfaz de GThumb

2.4.4 Sublime Text 2

Sublime Text es un potente editor de texto y código fuente creado originalmente como extensión de Vim. A pesar de ser software propietario se puede obtener una licencia gratuita de uso ilimitado y sin restricciones. Además, se puede ampliar su funcionalidad con infinidad de plug-ins dependiendo del lenguaje o tecnologías para las que se vaya a programar.

La imagen muestra la interfaz de usuario de Sublime Text 2. La barra superior muestra la ruta del archivo actual: '~node/dnIExpress/app.js (www, node) - Sublime Text (UNREGISTERED)'. A la derecha de la barra, se encuentran los iconos para cambiar el tema, el idioma, el tamaño de fuente y la hora ('13:30'). La barra lateral izquierda muestra una estructura jerárquica de carpetas y archivos, dividida en 'GROUP 1' y 'GROUP 2'. Los grupos contienen archivos como app.js, documentos.js, documentos.jade, apiRest.js, cliente.js, etc. La parte central de la pantalla muestra el contenido del archivo app.js, que es un script de Node.js que configura un servidor Express. El código incluye la importación de módulos como 'path', 'body-parser', 'express', 'http', 'socket.io', entre otros, y la configuración del servidor para manejar solicitudes y respuestas. La interfaz tiene un diseño oscuro y moderno.

```

var path = require('path');
var bodyParser = require('body-parser');

var express = require('express');
var http = require('http');
var app = express();
var server = http.Server(app);
var io = require('socket.io')(server);
var expressIo = io;

var routes = require('./routes/index');
var users = require('./routes/users');
var documentos = require('./routes/documentos');
var api = require('./routes/apiRest');

app.set('port', process.env.PORT || 3000);

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(bodyParser.json({limit: '25mb'}));
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);

app.get('/documentosweb', documentos.get_documentos);
app.get('/documentosweb/estadisticas', documentos.get_estadisticas);
app.get('/documentosweb/filtros/validez', documentos.get_filtro);
app.get('/documentosweb/filtros/nif', documentos.get_filtro);
app.get('/documentosweb/filtros/digitos', documentos.get_filtro);
app.get('/documentosweb/filtros/caducidad', documentos.get_filtro);
app.get('/documentosweb/:id', documentos.get_documento);
app.get('/documentosweb/:id', documentos.get_documento);

app.get('/documentos', api.get_documentos);
app.get('/documentos/:id', api.get_documento);
app.get('/documentos/:id/imagenes', api.get_imagenes);
app.post('/documentos', api.post_documento);
app.delete('/documentos/:id', api.delete_documento);

server.listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});

module.exports = app;

```

Ilustración 14: Interfaz de Sublime Text 2

Se ha elegido sobre otras opciones debido a su potencia, sencillez y al hecho de haber trabajado con él en numerosas ocasiones. Se ha usado para desarrollar la capa de acceso a datos en Node.js.

2.4.5 MySQL Workbench

MySQL Workbench es una herramienta de diseño de bases de datos MySQL. Permite su creación, administración y mantenimiento entre otras muchas opciones. Se ha utilizado para el desarrollo de la base de datos sobre la que se guardan los datos capturados de los DNI y se hacen las consultas para mostrar el histórico.

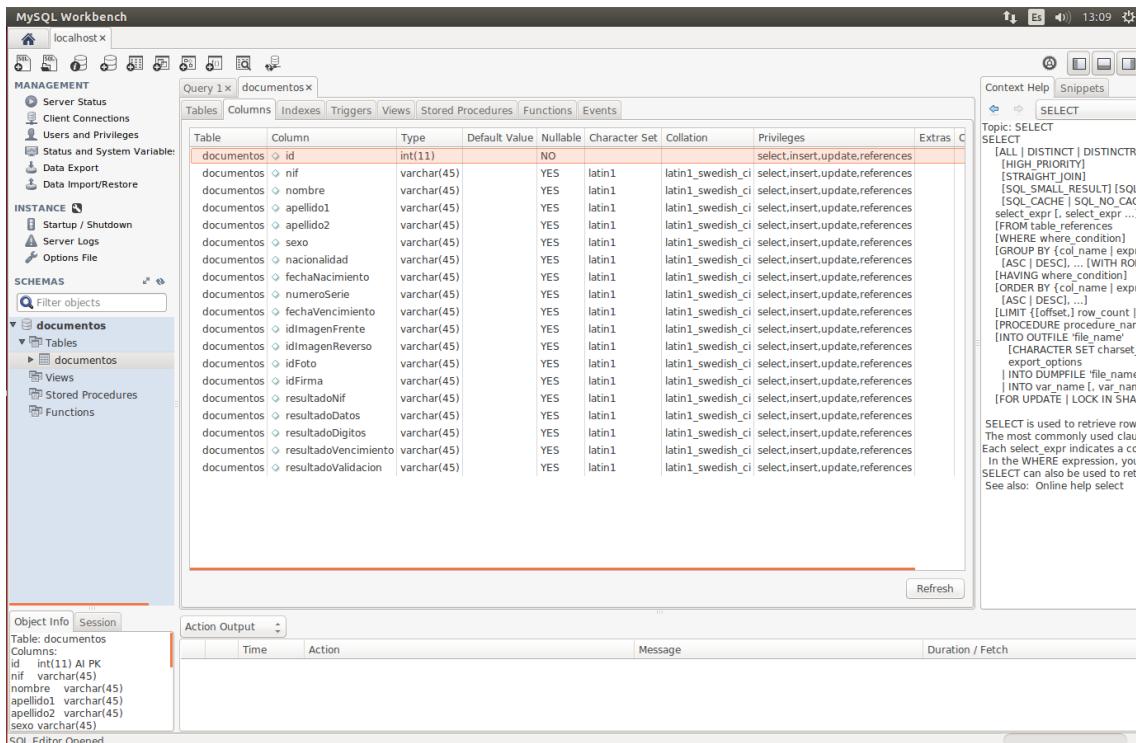


Ilustración 15: Interfaz de MySQLWorkbench

2.4.6 Herramientas utilizadas para las pruebas

Dado que la mayoría de la funcionalidad del proyecto está implementada en Javascript, se ha escogido un framework que permita realizar pruebas unitarias en este lenguaje. Dentro de las opciones que había, se ha optado por la más utilizada, QUnit. Este framework permite realizar pruebas unitarias tanto en Javascript genérico como en JQuery , JQuery UI y JQuery Mobile. Por otro lado, se han utilizado las herramientas de validación del W3C para validar los ficheros HTML y CSS.

Al solo haber realizado pruebas unitarias sobre el código Javascript, no ha sido necesario utilizar ninguna herramienta que permita realizarlas sobre Java u Objective-C.

3. Análisis de requisitos

En este apartado se detallan los requisitos funcionales y no funcionales del sistema, así como algunos diagramas de casos de uso.

3.1 Requisitos funcionales

Aquí se muestra una tabla con los requisitos funcionales del sistema.

ID	Descripción del requisito
RF01	El usuario podrá realizar una fotografía de la parte frontal del DNI y otra del reverso.
RF02	El sistema deberá reconocer y extraer el DNI de cada una de las dos fotografías.
RF03	El sistema deberá leer y extraer los datos de los campos del DNI.
RF04	El usuario podrá ver las instrucciones de uso de la aplicación.
RF05	El sistema deberá extraer la fotografía y la firma del DNI.
RF06	El usuario podrá ver un histórico con todos los datos de los DNIs capturados por la aplicación.
RF07	El sistema deberá ordenar el histórico por fecha y mostrar el nombre y apellidos del DNI validado.
RF08	El usuario podrá acceder a cada entrada del histórico para ver todos los datos del documento.
RF09	El usuario podrá borrar entradas del histórico.
RF10	El sistema deberá comprobar que la letra del DNI es correcta.
RF11	El sistema deberá comprobar que los campos del DNI del frente y el reverso coinciden.
RF12	El sistema deberá comprobar que los dígitos de control son correctos.
RF13	El sistema deberá comprobar que el DNI no está caducado.
RF14	El usuario podrá descartar cualquiera de las fotografías y volver al paso anterior.
RF16	El sistema deberá proporcionar una interfaz web desde la que visualizar los datos del histórico en tiempo real.

3.2 Diagramas de casos de uso y plantillas

En este apartado se muestran el diagrama de casos de uso y plantillas realizados en el desarrollo del proyecto. Solo se han realizado plantillas de los casos de uso que se no se han considerado triviales.

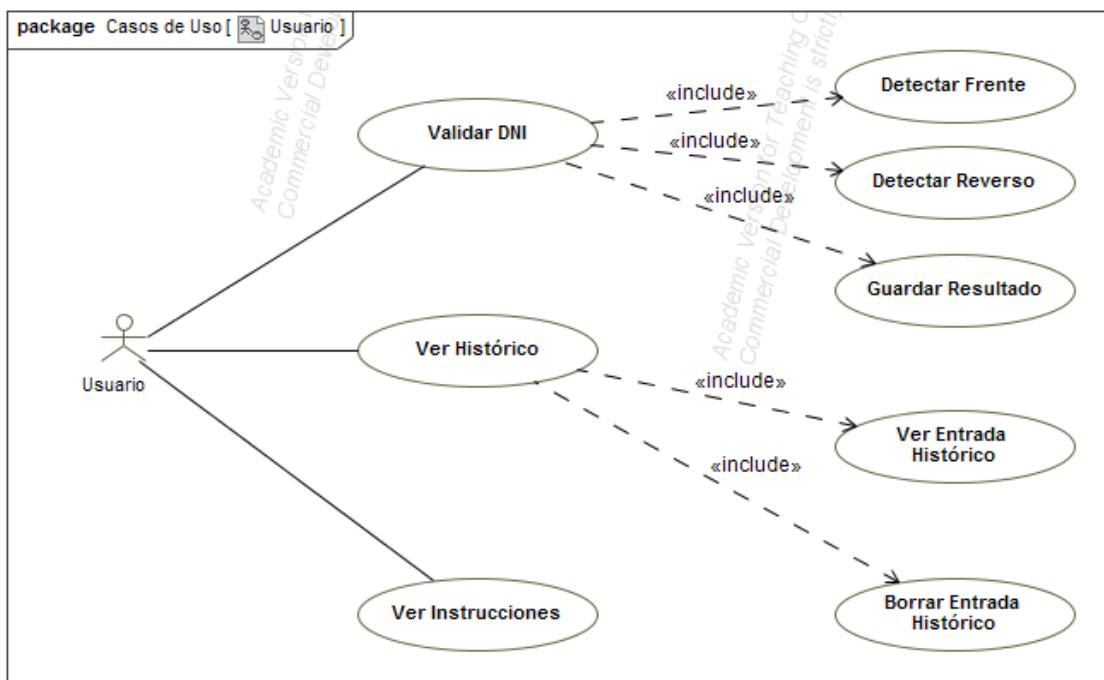


Ilustración 15: Diagrama de casos de uso

Id + Nombre	Ref-01-Detectar Frente
Autor	Marcos López Martínez
Actor principal	Usuario
Descripción	El usuario puede comenzar a validar el DNI sacando una foto de la parte frontal. El sistema se encargará de extraer los datos de esta foto.
Evento de activación	El usuario selecciona "Capturar parte frontal".
Precondición	Ninguna.
Garantías si éxito	El sistema extraerá los datos de la parte frontal del DNI.
Garantías mínimas	Los datos del sistema se mantienen consistentes.
Escenario Principal	<ol style="list-style-type: none"> 1. Abre la cámara del dispositivo. 2. El usuario toma una fotografía de la parte frontal del DNI. 3. El sistema extrae el DNI de la fotografía. 4. El sistema extrae los datos de la parte frontal del DNI. 5. El sistema muestra en pantalla los datos extraídos, así como la foto y la firma del DNI.
Extensiones	<ol style="list-style-type: none"> 1a. El usuario sale de la cámara. <ol style="list-style-type: none"> a.1. El sistema sale de la validación. 5a. El usuario selecciona la opción volver a capturar frente. <ol style="list-style-type: none"> a.1. El sistema vuelve al paso 1. 5b. El usuario selecciona la opción cancelar. <ol style="list-style-type: none"> b.1. El sistema sale de la validación.

Id + Nombre	Ref-02-Detectar Reverso
Autor	Marcos López Martínez
Actor principal	Usuario

Descripción	El usuario puede terminar de validar el DNI sacando una foto del reverso. El sistema se encargará de extraer los datos de esta foto y comprobar que el documento es válido.
Evento de activación	El usuario selecciona “Capturar reverso”.
Precondición	Ya se han extraído los datos de la parte frontal.
Garantías si éxito	El sistema extraerá los datos del reverso del DNI y comprobará su validez.
Garantías mínimas	Los datos del sistema se mantienen consistentes.
Escenario Principal	<ol style="list-style-type: none"> 1. Abre la cámara del dispositivo. 2. El usuario toma una fotografía del reverso del DNI. 3. El sistema extrae el DNI de la fotografía. 4. El sistema extrae los datos del reverso del DNI. 5. El sistema comprueba la validez del DNI a partir de estos datos y los de la parte frontal previamente extraídos. 6. El sistema muestra en pantalla los datos extraídos, así como el resultado de la validación.
Extensiones	<ol style="list-style-type: none"> 1a. El usuario sale de la cámara. <ol style="list-style-type: none"> a.1. El sistema vuelve al último paso del caso de uso “Validar Frente”. 6a. El usuario selecciona la opción volver a capturar reverso. <ol style="list-style-type: none"> a.1. El sistema vuelve al paso 1. 6b. El usuario selecciona la opción cancelar. <ol style="list-style-type: none"> b.1. El sistema sale de la validación.

3.3 Requisitos no funcionales

Aquí se muestra una tabla con los requisitos no funcionales del sistema.

ID	Descripción del requisito	Categoría	Importancia
RNF01	El sistema ha de ser compatible con las principales plataformas móviles: Android e iOS.	Portabilidad	Alta
RNF02	El sistema deberá poseer una interfaz sencilla e intuitiva que facilite el uso de la aplicación.	Accesibilidad	Media
RNF03	Se debe respetar la integridad de los datos capturados por la aplicación.	Seguridad	Alta
RNF04	La interfaz de la aplicación debe adaptarse al tamaño de pantalla del dispositivo.	Accesibilidad, Portabilidad	Media
RNF05	El tiempo de procesado para el reconocimiento del DNI y la lectura de los campos no debe ser superior a 20 segundos para cada lado del DNI.	Rendimiento	Baja
RNF06	El sistema debe ser relativamente fácil de portar a otros sistemas operativos móviles.	Portabilidad, Escalabilidad	Media
RNF07	Los datos de los DNI capturados se deben guardar en una base de datos en un servidor.	Escalabilidad	Alta

4. Diseño

En este apartado se especifican el diseño arquitectónico y detallado de la aplicación.

4.1 Diseño arquitectónico

Se ha optado por un diseño arquitectónico^[13] en 3 capas. Esta arquitectura tiene el objetivo de separar la interfaz de usuario, la lógica de negocio y el acceso a datos en diferentes módulos lógicos, haciéndolas independientes unas de otras. Su principal ventaja es que al estar cada capa abstraída del resto, en caso de ser necesario realizar cambios solo se debe modificar la capa que los requiera. A continuación se describe la funcionalidad de las distintas capas en el sistema.

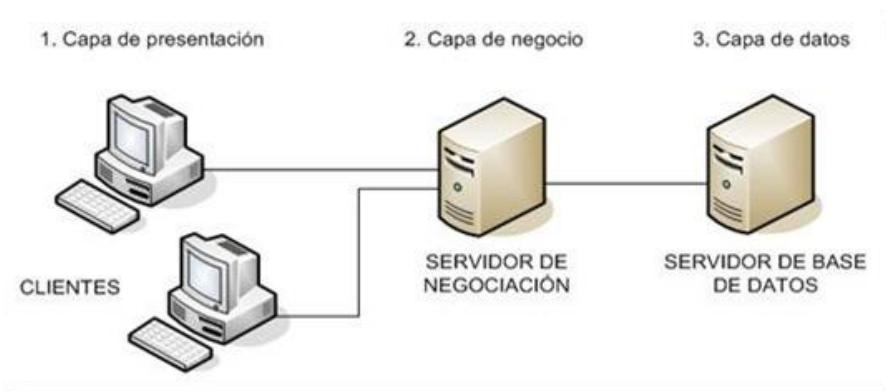


Ilustración 16: Arquitectura en 3 capas

Capa de presentación: es la interfaz gráfica o parte visible para el usuario. Su función principal es mostrar información y obtener datos del usuario. Debe ser simple y fácil de usar. Se compone de los ficheros HTML y CSS que definen la interfaz y del código Javascript encargado de controlar la navegación entre las distintas partes de la aplicación. También se incluye en esta capa la interfaz web externa a la aplicación, teniendo 2 niveles de presentación totalmente separados. Se comunica con la capa de negocio para enviar la solicitudes realizadas por el usuario y mostrar los resultados.

Capa de negocio: es donde reside la lógica de negocio y se implementa la funcionalidad de la aplicación. En esta capa se reciben las peticiones de usuario y se envían las respuestas después de procesarlas. Se compone de todo el código Javascript encargado de realizar la funcionalidad principal de la aplicación y del código nativo Java (Android) y Objective-C (iOS) que realiza el reconocimiento del DNI y los caracteres.

Capa de datos: es donde se almacenan los datos y posteriormente se accede a ellos. Está formada por una base de datos MySQL y una API REST en Node.js que hace de interfaz entre la base de datos y la capa de negocio. Se comunica con la capa de negocio para recibir y almacenar los datos capturados por ésta y enviar los datos requeridos. También forma parte de esta capa la aplicación web en Node.js encargada de extraer los datos.

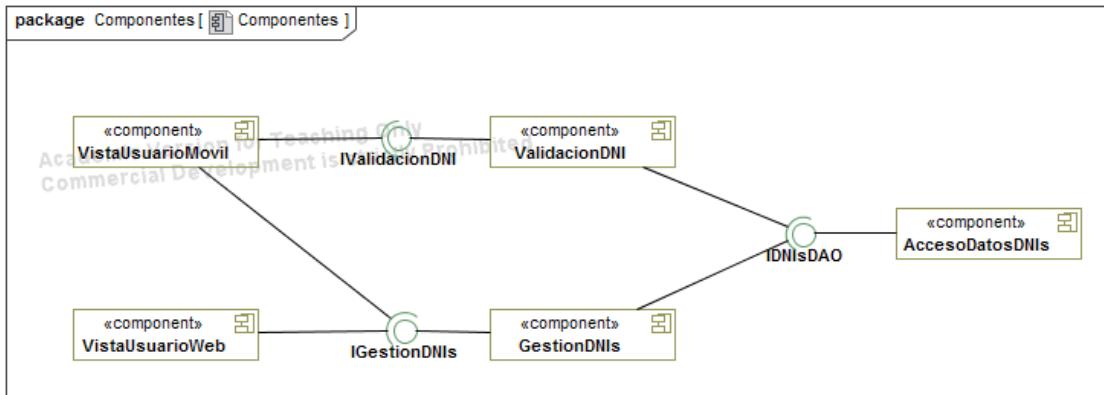


Ilustración 17: Diagrama de componentes del sistema

Estas capas pueden estar implementadas en un mismo dispositivo o dividirse entre varios. A cada dispositivo físico en el que se implementan las capas lógicas se le conoce como nivel. La aplicación móvil está implementada en 2 niveles, las capas de presentación y negocio en el dispositivo móvil y la capa de datos en un servidor. Por otro lado, la aplicación web está implementada en su totalidad en el servidor.

4.2 Diseño detallado

En este apartado se pueden ver varios diagramas^{[14]/[15]} realizados durante el desarrollo del proyecto. En primer lugar está el diagrama de navegación que detalla las diferentes vistas de la capa de presentación de la aplicación y como se puede navegar entre ellas. En segundo lugar se definen los métodos y operaciones de cada una de las interfaces del diagrama de componentes del apartado anterior. Por último, se define la única clase del sistema así como sus atributos.

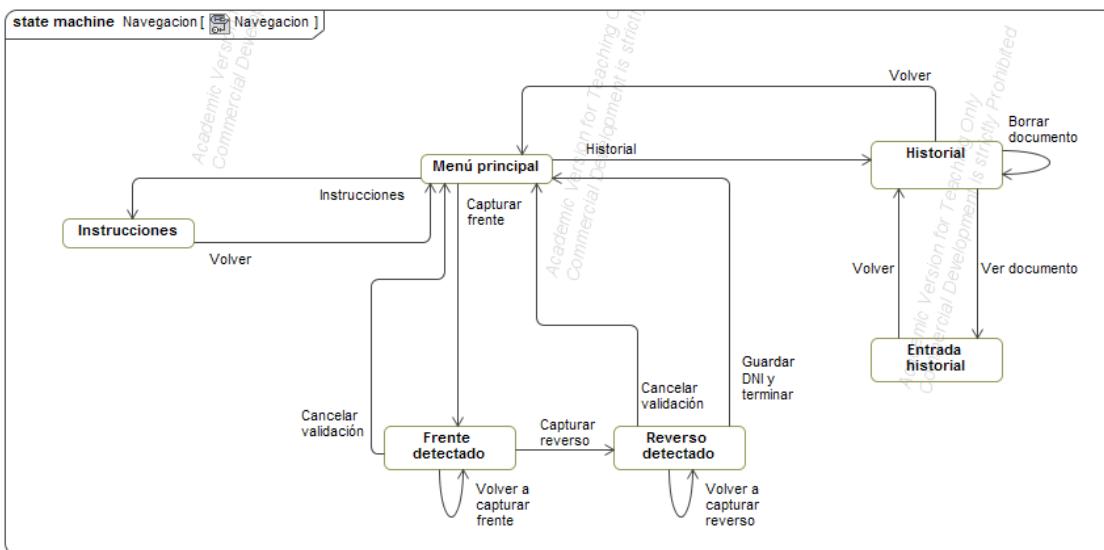


Ilustración 18: Diagrama de navegación

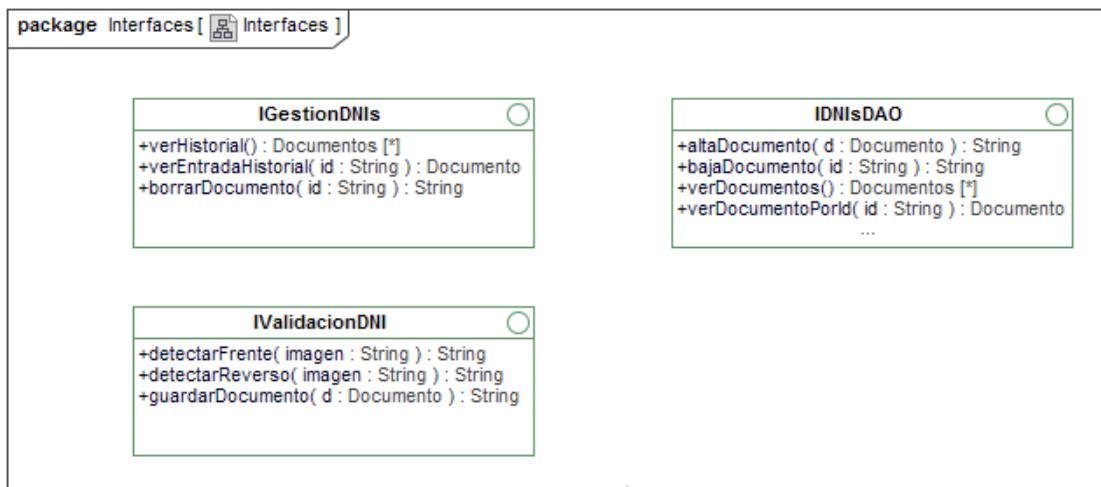


Ilustración 19: Definición de interfaces

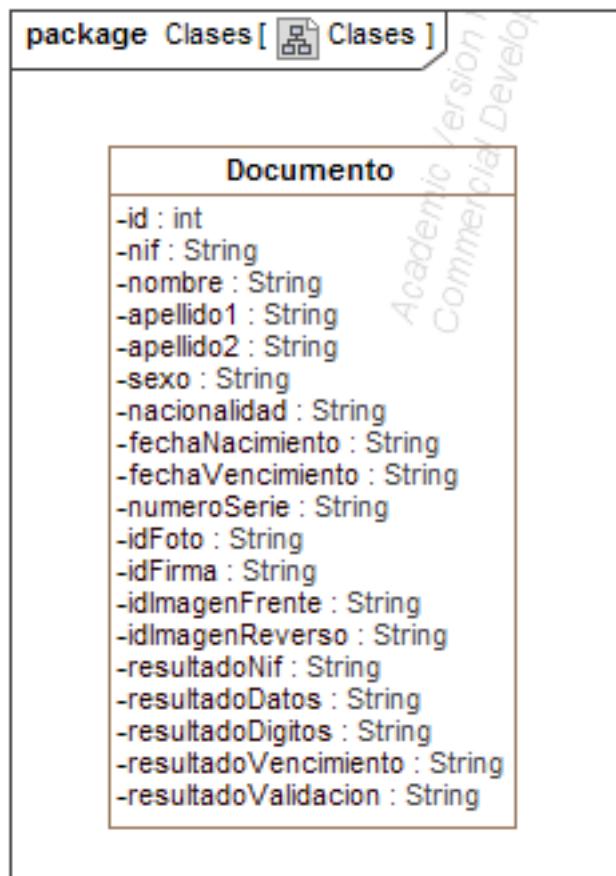


Ilustración 20: Clase Documento

5. Implementación

5.1 Entrenamiento de la Cascada Haar

En este apartado se detalla el proceso seguido para el entrenamiento de la Cascada Haar^[16] utilizando los datos de la versión final de la cascada que detecta la parte frontal del DNI.

Antes de nada es necesario tener instalado OpenCV en el sistema en el que se pretenda entrenar la cascada. Después se preparan las muestras positivas, que son imágenes con el objeto a detectar y las negativas, que son imágenes sin el objeto.

Los clasificadores realmente buenos requieren del orden de entre 1000 y 10000 muestras positivas y el mismo número de negativas. Sin embargo, la recopilación y preparación de una cantidad de muestras tan elevada supondría un trabajo desproporcionado, por lo que para este proyecto se han utilizado 435 muestras positivas para cada una de las dos cascadas (870 en total) y 3120 negativas (las mismas para ambas cascadas). A pesar de ser un número menor que el recomendado para cascadas de máxima precisión, el resultado final ha sido bastante satisfactorio.

5.1.1 Preparación de las muestras negativas

En nuestro caso, las muestras negativas deben ser imágenes que no contengan el DNI y que preferiblemente sean lo más parecidas a los fondos sobre los que se van a tomar las fotos reales.

Primero, se han recolectado por internet un total de 100 imágenes de suelos, mesas y demás fondos que probablemente estén presentes en las imágenes reales. Después, se han descargado 3020 de las muestras negativas disponibles en el siguiente repositorio subversion:

<http://tutorial-haartraining.goolecode.com/svn/trunk/data/negatives>

Una vez se tienen todas las imágenes, se introducen en una carpeta que hemos llamado `neg` y se utiliza el comando `find` para guardar sus directorios relativos en un archivo, `negativos.txt`:

```
find neg -iname "*.jpg" > negativos.txt
```



Ilustración 21: Una de las imágenes utilizadas para las muestras negativas

5.1.2 Preparación de las muestras positivas

Las muestras positivas deben mostrar imágenes que contengan el DNI, en concreto para esta cascada necesitamos fotos de la parte frontal. Para conseguirlas se han sacado un total de 435 fotos de 29 DNI (15 fotos de cada uno). Después, se han tratado una a una con GThumb para recortar solo el DNI (en las muestras positivas no interesa el fondo) y redimensionarlas para que sus proporciones sean iguales.



Ilustración 22: Una de las imágenes utilizadas para las muestras positivas

Si solamente utilizásemos estas 435 muestras para el entrenamiento de la cascada los resultados serían bastante menos precisos de lo deseado. Por suerte OpenCV cuenta con una utilidad llamada `opencv_createsamples` que genera varias muestras positivas a partir de una sola aplicando transformaciones y distorsiones. En este caso se han creado 20 muestras a partir de cada una de las imágenes iniciales, consiguiendo un total de 8700. Este es el comando completo utilizado para la primera imagen (se ha desarrollado un script que aplica el mismo comando sobre las 435 muestras):

```
opencv_createsamples -num 20 -bg negativos.txt -maxxangle 0 -  
maxyangle 0 -maxzangle 0.2 bgcolor 255 -w 40 -h 25 -vec pos1.vec  
-img pos/dnif0001.jpg
```

-num: número de muestras que se desean crear a partir de la imagen.

-bg: archivo donde se listan las muestras negativas.

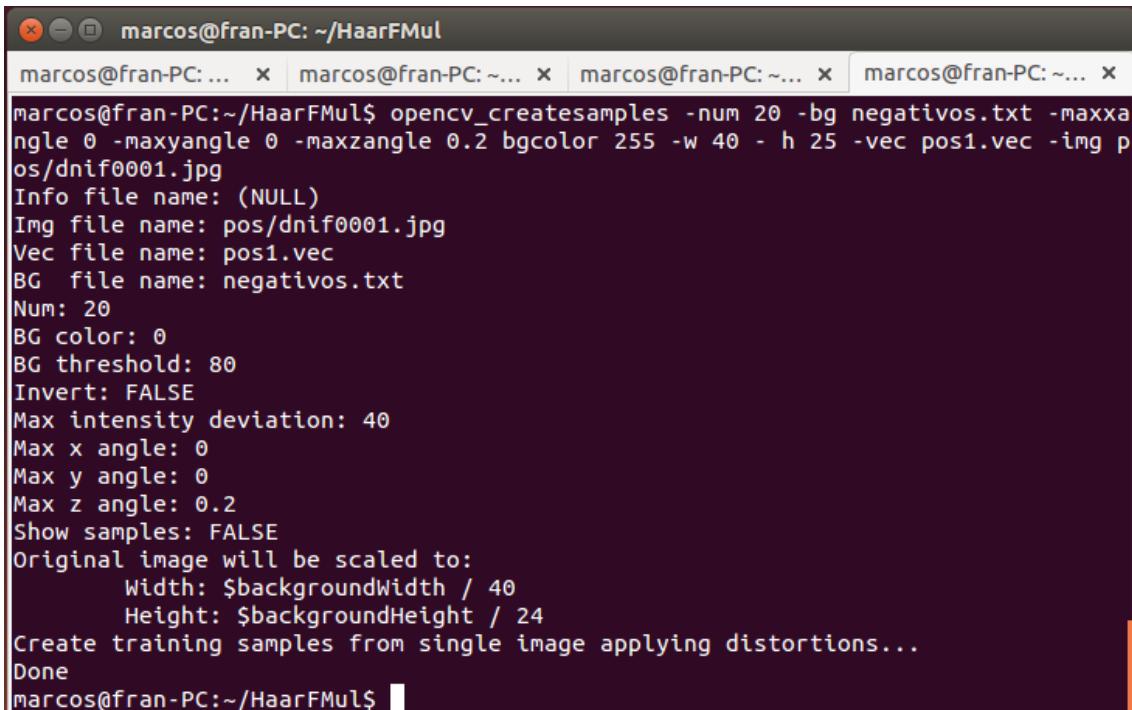
-maxxangle, -maxyangle, -maxzangle: ángulos máximos que se puede rotar la imagen en cada uno de los 3 ejes.

-bgcolor: color de fondo.

-w y -h: anchura y altura de las muestras resultantes en píxeles. Deben guardar la misma proporción que la imagen.

-vec: nombre del archivo de muestras resultante.

-img: nombre de la imagen a partir de la cual se desean extraer las muestras.

A screenshot of a terminal window titled "marcos@fran-PC: ~/HaarFMul". The window contains a command-line session where the user runs the "opencv_createsamples" command with various parameters. The command specifies 20 samples, uses a background file named "negativos.txt", and includes rotation angles of 0 for X, Y, and Z axes. It also sets a background color of 255, a width of 40 pixels, and a height of 25 pixels. The resulting file is named "pos1.vec". The user also specifies an input image "pos/dnif0001.jpg". The output shows the configuration details and the creation of training samples from the image.

```
marcos@fran-PC: ... x marcos@fran-PC: ~... x marcos@fran-PC: ~... x marcos@fran-PC: ~... x
marcos@fran-PC:~/HaarFMul$ opencv_createsamples -num 20 -bg negativos.txt -maxxangle 0 -maxyangle 0 -maxzangle 0.2 bgcolor 255 -w 40 - h 25 -vec pos1.vec -img pos/dnif0001.jpg
Info file name: (NULL)
Img file name: pos/dnif0001.jpg
Vec file name: pos1.vec
BG file name: negativos.txt
Num: 20
BG color: 0
BG threshold: 80
Invert: FALSE
Max intensity deviation: 40
Max x angle: 0
Max y angle: 0
Max z angle: 0.2
Show samples: FALSE
Original image will be scaled to:
    Width: $backgroundWidth / 40
    Height: $backgroundHeight / 24
Create training samples from single image applying distortions...
Done
marcos@fran-PC:~/HaarFMul$
```

Ilustración 23: Ejemplo de uso de la utilidad createsamples

Una vez se tienen los 435 archivos .vec resultantes se ha utilizado el programa `mergevec.cpp` creado por Naotoshi Seo para unirlos en un solo archivo. Primero se guardan sus directorios relativos en un solo archivo `positivos.txt` y después se usa `opencv_mergevec` para unirlos:

```
find -name "*.vec" > positivos.txt
./opencv_mergevec positivos.txt positivos.vec
```

Con esto ya tendremos el archivo `positivos.vec` necesario para entrenar la cascada.

5.1.3 Entrenamiento de la cascada

OpenCV nos ofrece dos utilidades diferentes a la hora de entrenar una cascada Haar: `opencv_haartraining` y `opencv_traincascade`. En este proyecto se ha utilizado la segunda ya que es más reciente y permite que el proceso de entrenamiento sea multi-hilo, reduciendo así su duración. Este es el comando exacto utilizado para entrenar la cascada:

```
opencv_traincascade -data dniHaarCascade -vec positivos.vec -bg negativos.txt -numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 5475 -numNeg 3120 -w 40 -h 25 -mode ALL
```

- data**: carpeta en la que se guardará el resultado.
- vec**: archivo que contiene las muestras positivas.
- bg**: archivo donde se listan las muestras negativas.
- numStages**: número de etapas de la cascada.
- minHitRate**: número mínimo de aciertos para cada etapa.
- maxFalseAlarmRate**: número máximo de falsos positivos para cada etapa.
- numPos**: número de muestras positivas.
- numNeg**: número de muestras negativas.
- w y h**: anchura y altura de las muestras positivas en píxeles.
- mode**: tipos de filtros Haar utilizados en el entrenamiento.

Una vez hecho esto la cascada comienza su entrenamiento y entre 48 y 96 horas después genera un archivo `.xml` que podrá ser utilizado por la librería OpenCV de la aplicación.

```

marcos@fran-PC: ~/HaarFMul
marcos@fran-PC: ... x marcos@fran-PC: ~... x marcos@fran-PC: ~... x marcos@fran-PC: ~... x
marcos@fran-PC:~/HaarFMul$ opencv_traincascade -data dniHaarCascade -vec positivos.vec -bg negativos.txt -numStages 20 -minHitRate 0.999 -maxFalseAlarmRate 0.5 -numPos 5475 -numNeg 3120 -w 40 -h 25 -mode ALL
Training parameters are loaded from the parameter file in data folder!
Please empty the data folder if you want to use your own set of parameters.
PARAMETERS:
cascadeDirName: dniHaarCascade
vecFileName: positivos.vec
bgFileName: negativos.txt
numPos: 5475
numNeg: 3120
numStages: 20
precalcValBufSize[Mb] : 256
precalcIdxBufSize[Mb] : 256
stageType: BOOST
featureType: HAAR
sampleWidth: 40
sampleHeight: 25
boostType: GAB
minHitRate: 0.999
maxFalseAlarmRate: 0.5
weightTrimRate: 0.95
maxDepth: 1
maxWeakCount: 100
mode: ALL

===== TRAINING 0-stage =====
<BEGIN
POS count : consumed 5475 : 5475
NEG count : acceptanceRatio 3120 : 1

```

Ilustración 24: Ejemplo de uso de la utilidad traincascade

Como se puede observar los números de muestras positivas utilizadas en el entrenamiento no coinciden con las reales. Mientras que tenemos 8700 muestras positivas, en el comando se indican 5475 muestras. Esto se debe a que la cascada va descartando algunas muestras positivas durante el entrenamiento. Si se le indicase que usara todas las disponibles, en cuanto descartase solo una no podría llegar al número indicado de muestras en la próxima iteración. Las muestras que se deben indicar se calculan mediante al siguiente fórmula:

$$\text{numPositivos} = (x + (\text{numEtapas} - 1) * (1 - \text{minNumAciertos}) * x) + \text{numNegativos}$$

$$\begin{aligned}
8700 &= (x + (20-1) * (1-0.999) * x) + 3120 \\
8700 &= (x + 19 * 0.001 * x) + 3120 \\
8700 &= x + 0.019x + 3120 \\
5580 &= 1.019x \\
x &= 5580/1.019 = 5475
\end{aligned}$$

Cabe mencionar, que aunque solo se ha descrito el proceso de entrenamiento de una cascada , para llegar a esta versión final se han entrenado un total de 14 cascadas diferentes (9 para la parte frontal y 5 para el reverso). Este alto número se debe a que se han ido probando entrenamientos con diferentes parámetros y muestras hasta encontrar las más precisas.

Por otro lado, como se ha mencionado en este apartado, para el entrenamiento de una cascada se necesitan entre 48 y 96 horas de media y al ser un proceso que consume una gran cantidad de recursos, impide el uso del equipo. Esto ha ralentizado el desarrollo de la aplicación en las partes que hacen uso de las cascadas.

5.2 Implementación de la capa de presentación

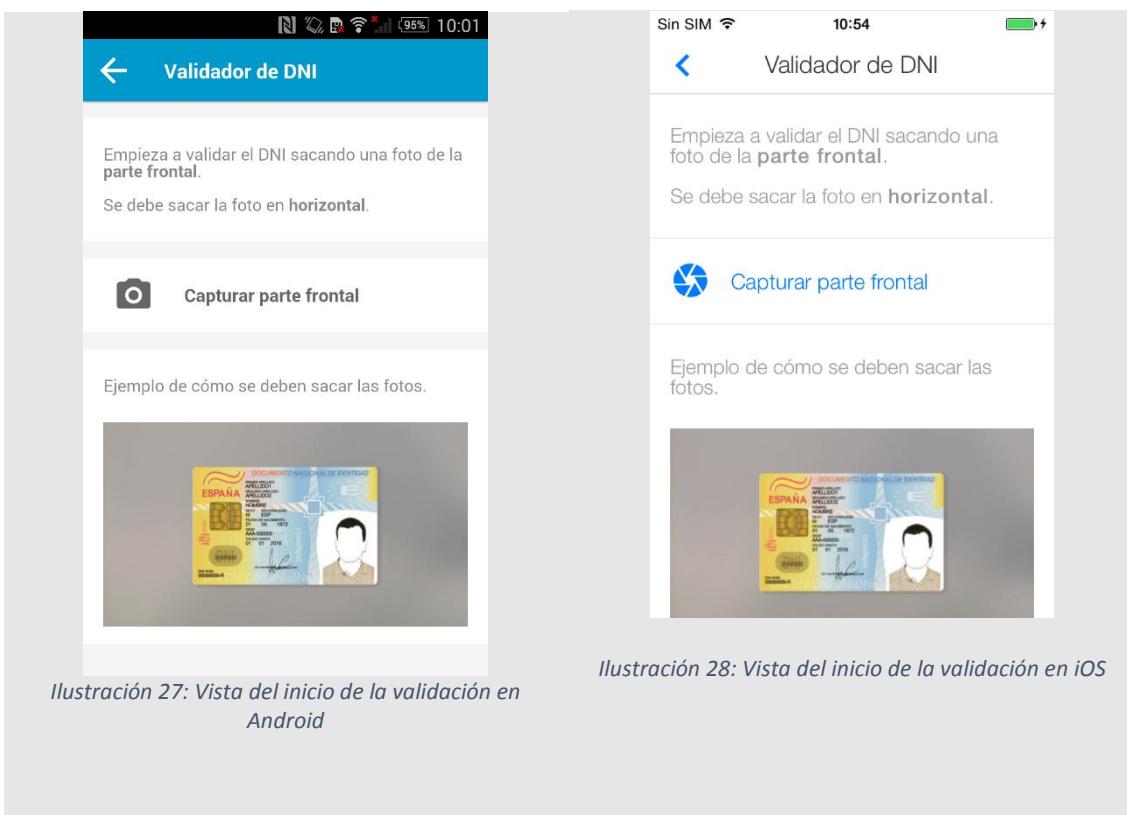
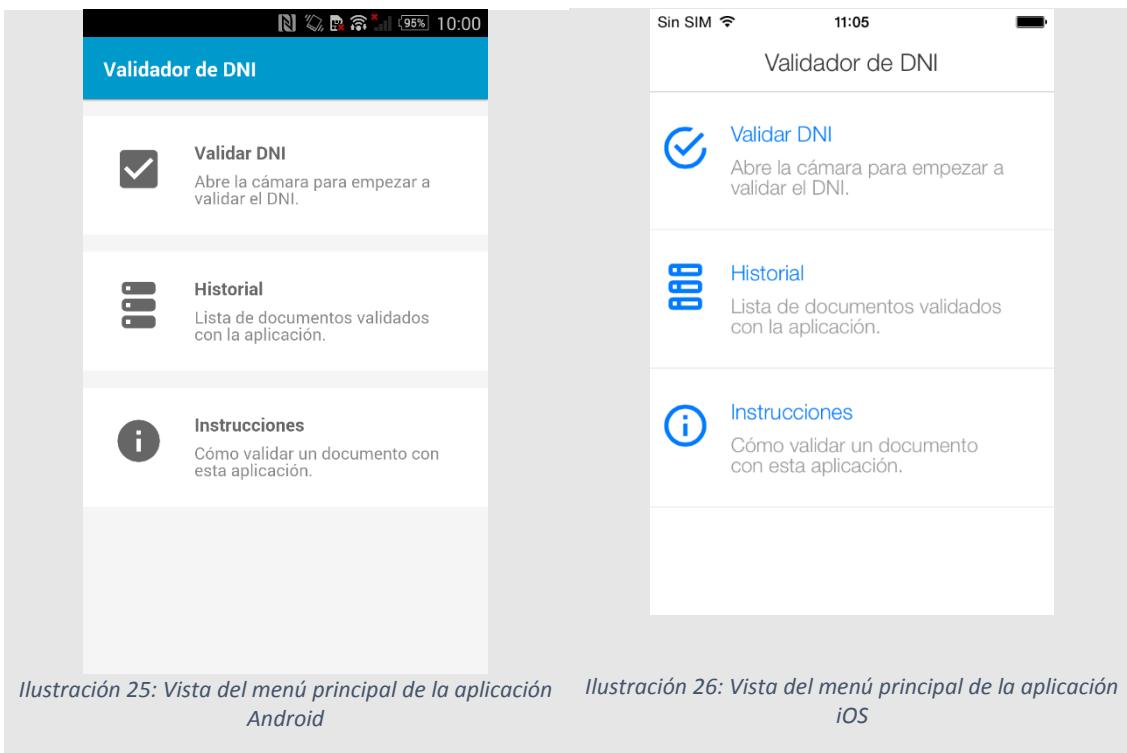
La implementación de la capa de presentación consta de 2 partes totalmente independientes. Por un lado tenemos la interfaz de la aplicación móvil y por otro la de la aplicación web.

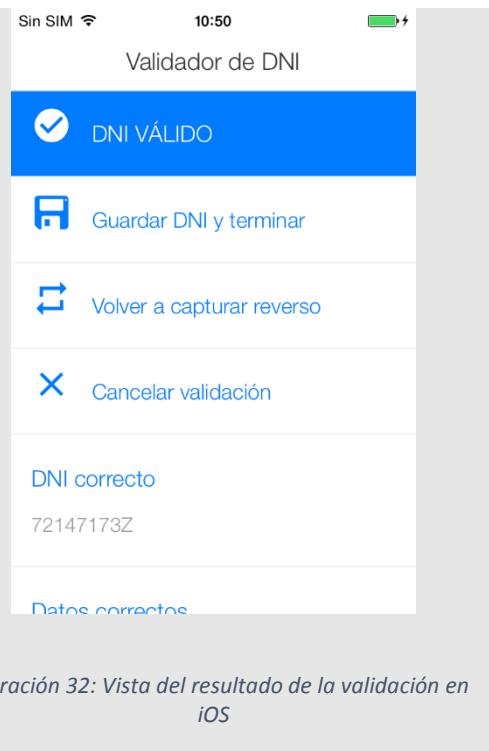
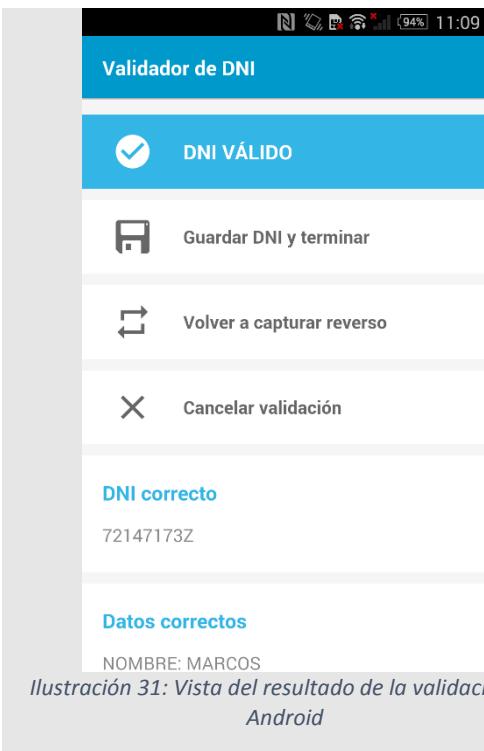
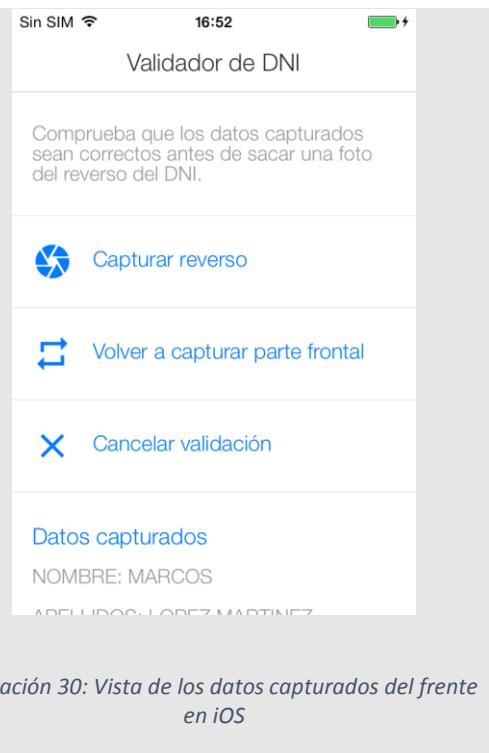
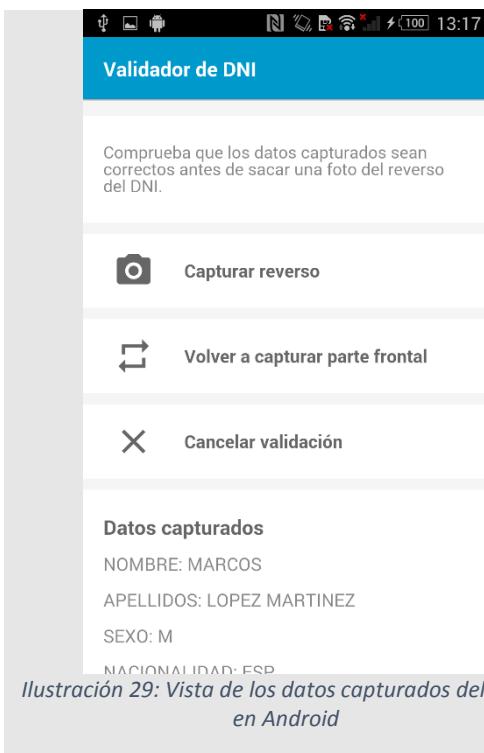
5.2.1 Interfaz móvil

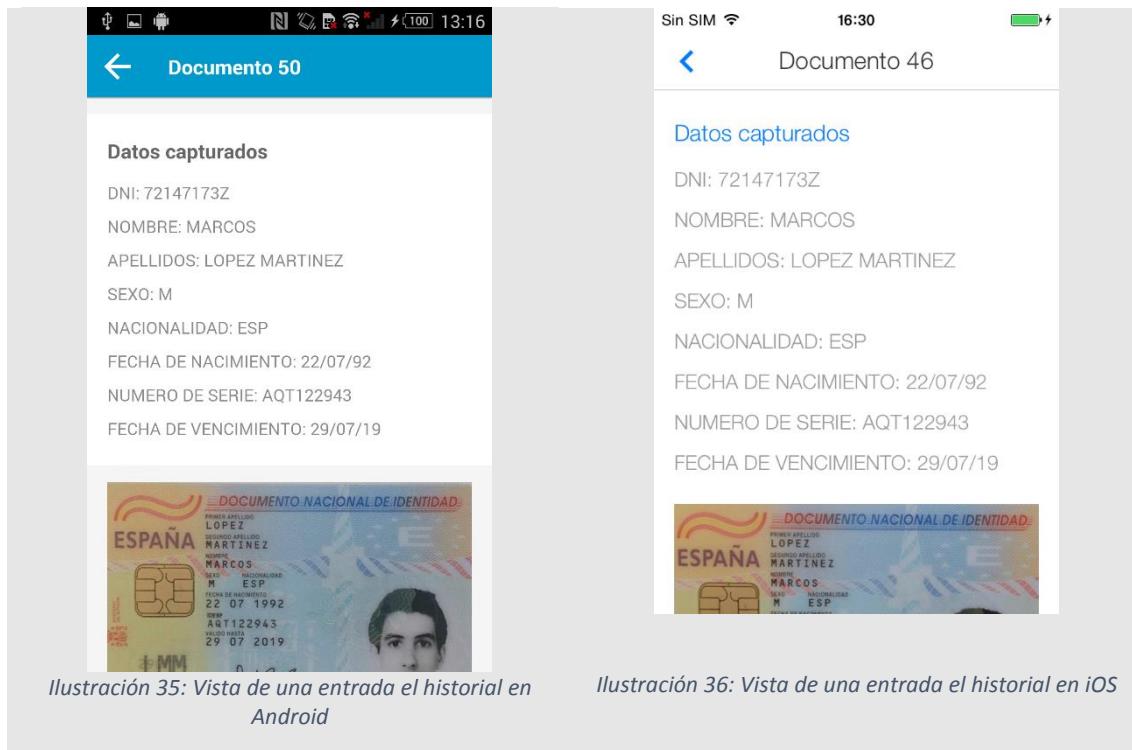
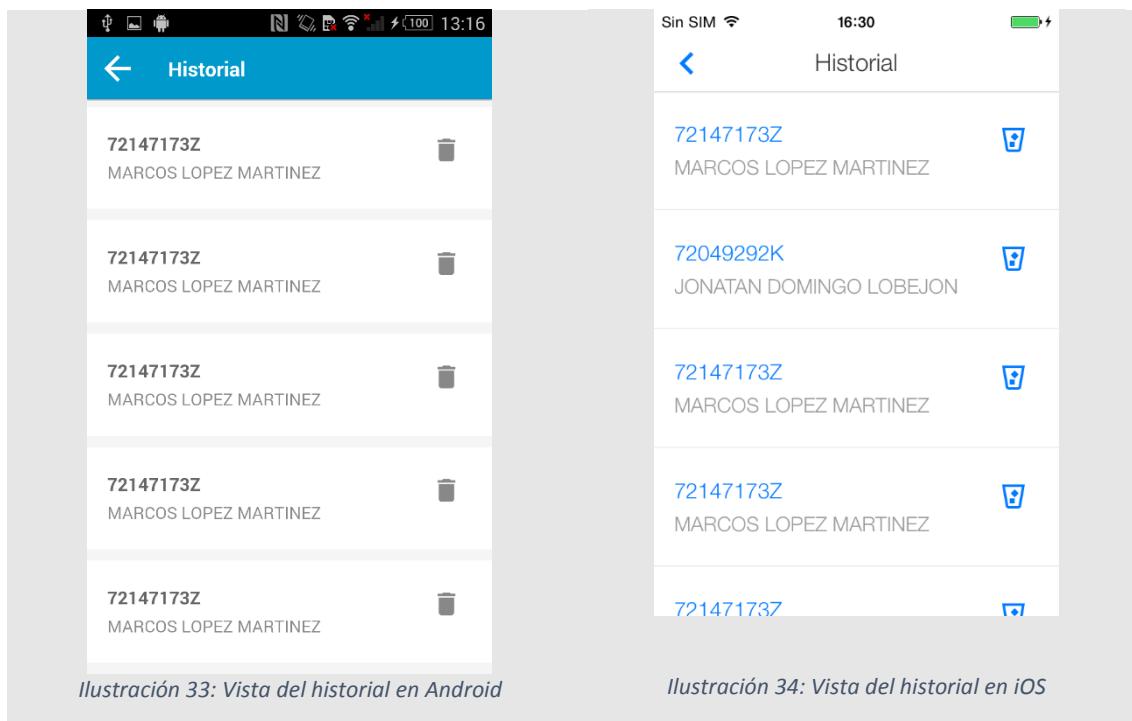
La interfaz de la aplicación móvil está implementada en un solo fichero HTML. Este fichero está dividido en varios *divs*, correspondiendo cada uno a una vista de la aplicación. El código Javascript se encarga de ir mostrando y ocultando los *divs* para crear el efecto de estar navegando entre diferentes pantallas. Al estar todas las vistas en un mismo fichero, se cargan por completo al inicio de la aplicación y da como resultado una navegación prácticamente instantánea entre ellas.

El estilo de la interfaz se define en 2 ficheros diferentes, un fichero reset.css que elimina los estilos añadidos por el navegador (al final una aplicación en Phonegap es como un navegador web) y uno style.css donde realmente se definen los estilos del HTML.

A continuación se muestran las vistas de la aplicación en Android e iOS:







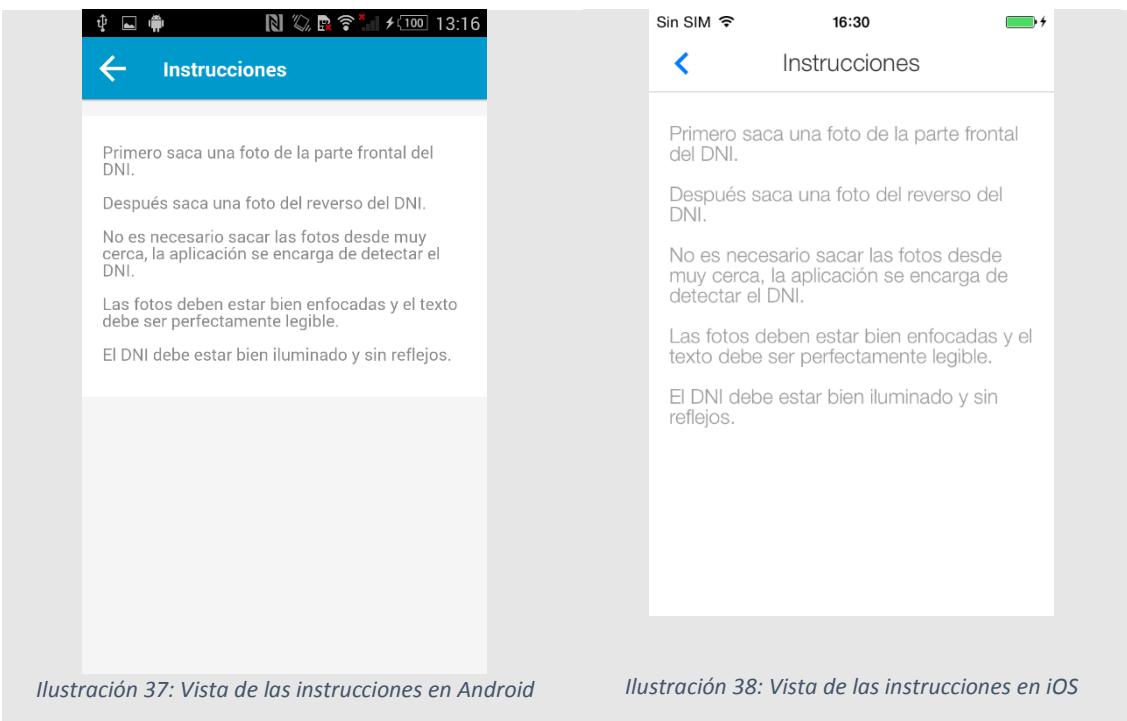


Ilustración 37: Vista de las instrucciones en Android

Ilustración 38: Vista de las instrucciones en iOS

5.2.2 Interfaz web

La interfaz web está implementada en varios ficheros HTML en Jade. Jade es el lenguaje de plantilla usado por el framework Express.js que permite escribir código más simple que el HTML y automáticamente generar HTML a partir de este. Además da la opción de incluir bucles, variables y condicionales entre otras opciones.

El estilo de la interfaz se ha definido utilizando el framework de desarrollo web Bootstrap, que permite crear páginas o aplicaciones web adaptativas y atractivas visualmente simplemente añadiendo clases a los elementos del HTML.

La aplicación web consta de las siguientes vistas o pantallas:

The screenshot shows a web browser window with the URL 192.168.50.213:3000/documentosweb/. The title bar says "VALIDADOR DE DNI". The page contains four search results for the DNI 72147173Z, each showing the name MARCOS LOPEZ MARTINEZ and a green "Ver" button.

Ilustración 39: Vista principal de la aplicación web

The screenshot shows a web browser window with the URL 192.168.50.213:3000/documentosweb/50. The title bar says "VALIDADOR DE DNI". The page displays a message "Documento válido." and three sections: "Datos", "Foto y firma", and "Documento completo".

- Datos:**
 - Dni: 72147173Z
 - Nombre: MARCOS
 - Primer Apellido: LOPEZ
 - Segundo Apellido: MARTINEZ
 - Sexo: M
 - Nacionalidad: ESP
 - Fecha de nacimiento: 22/07/92
 - Número de serie: AQT122943
 - Fecha de vencimiento: 29/07/19
- Foto y firma:** Displays a portrait photo and a handwritten signature.
- Documento completo:** Displays a scanned image of a Spanish National ID Card (DNI) showing all fields including photo, name, date of birth, and signature.

Ilustración 40: Vista de una entrada de la lista

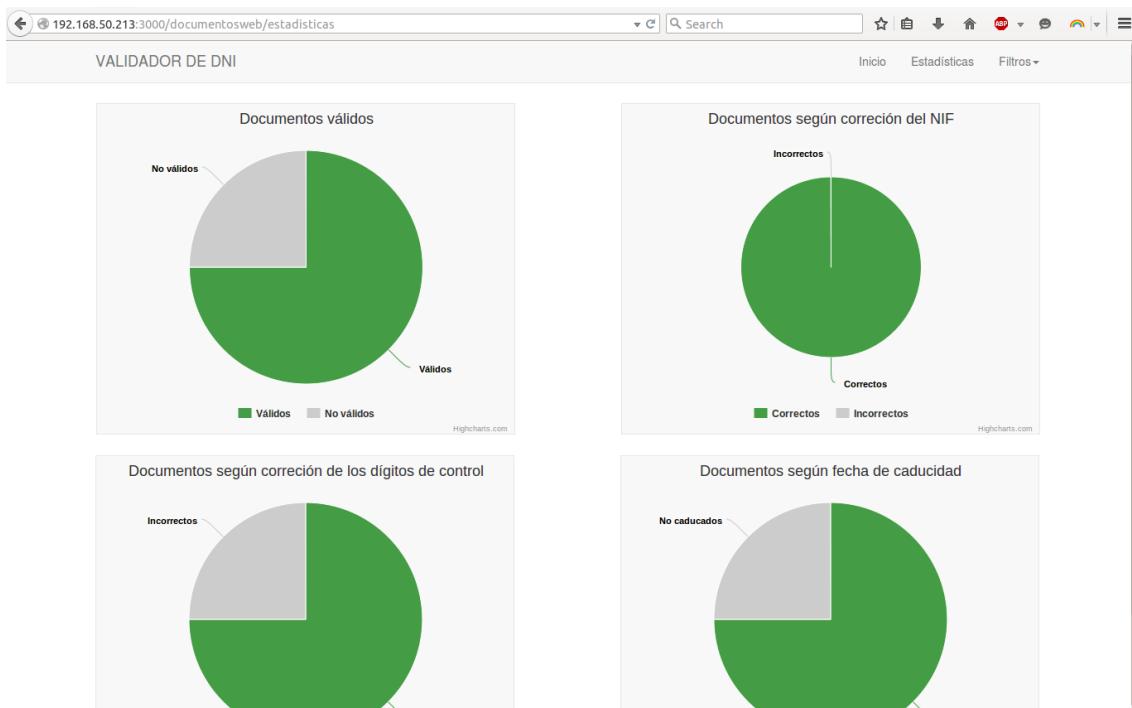


Ilustración 41: Vista de los gráficos de estadísticas

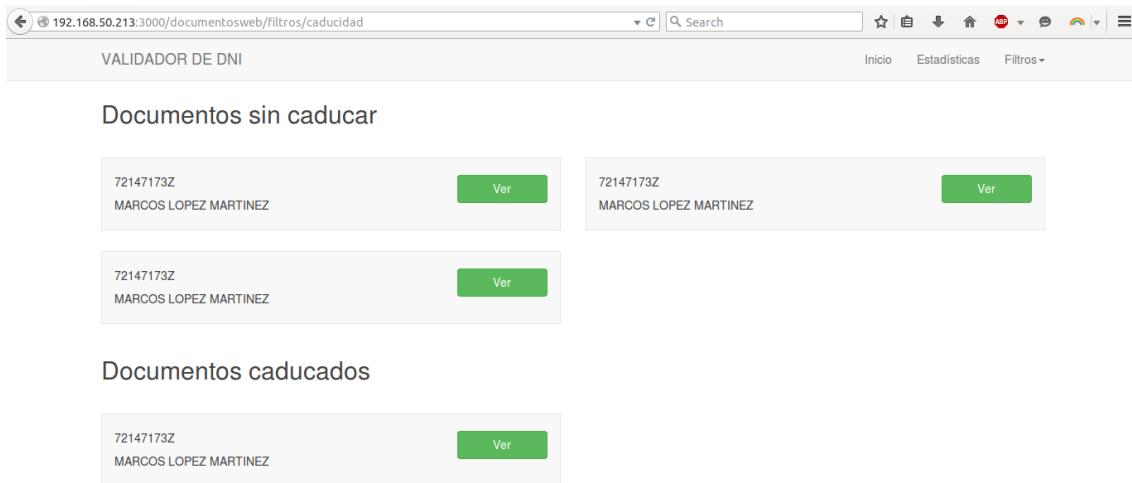


Ilustración 42: Vista la lista de elementos filtrados

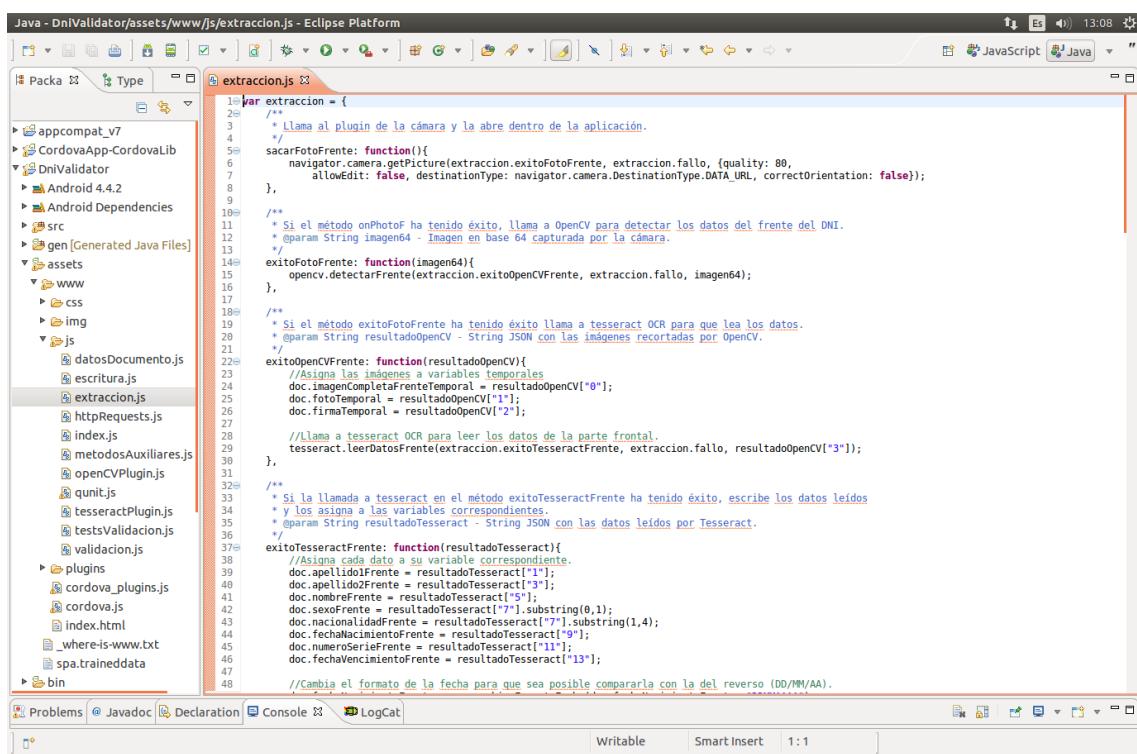
Se utiliza la librería Socket.io de node.js para mostrar los cambios en la base de datos en tiempo real, de tal manera que si se añade o borra un documento a la base de datos desde la aplicación móvil, los cambios se muestren al instante en pantalla sin necesidad de recargar la página.

5.3 Implementación de la capa de negocio

La implementación de la capa de negocio se ha realizado en dos partes diferenciadas: la parte Javascript, donde se controla el flujo principal de la aplicación y se realiza la validación del documento y la parte nativa, donde se extrae el DNI de la imagen y se leen los datos de los diferentes campos.

5.3.1 Parte Javascript

Es la parte que se encarga de controlar el flujo principal de la aplicación y de posteriormente validar los campos extraídos del DNI en la parte nativa. Por un lado, se encarga de realizar las llamadas a los plug-ins de los que hace uso la aplicación (los dos realizados desde cero, el de la cámara y el de las notificaciones) y de procesar los datos devueltos por estos.



```
Java - DniValidator/assets/www/js/extraccion.js - Eclipse Platform
[...] Packa Type [...]
appcompat_v7
CordovaApp-CordovaLib
DniValidator
Android 4.4.2
Android Dependencies
src
gen [Generated Java Files]
assets
www
css
img
js
datosDocumento.js
escritura.js
extraccion.js
httpRequests.js
index.js
metodosAuxiliares.js
openCVPlugin.js
qunit.js
tesseractPlugin.js
testsValidacion.js
validacion.js
plugins
cordova_plugins.js
cordova.js
index.html
where-is-www.txt
spa.traineddata
bin
extraccion.js
1 var extraccion = {
2     /**
3      * Llama al plugin de la cámara y la abre dentro de la aplicación.
4      */
5     sacarFotoFrente: function(){
6         navigator.camera.getPicture(extraccion.exitoFotoFrente, extraccion.fallo, {quality: 80,
7             allowEdit: false, destinationType: navigator.camera.DestinationType.DATA_URL, correctOrientation: false});
8     },
9
10    /**
11     * Si el método onPhotoF ha tenido éxito, llama a OpenCV para detectar los datos del frente del DNI.
12     * @param String imagen64 - Imagen en base 64 capturada por la cámara.
13     */
14    exitoFotoFrente: function(imagen64){
15        opencv.detectarFrente(extraccion.exitoOpenCVFrente, extraccion.fallo, imagen64);
16    },
17
18    /**
19     * Si el método onPhotoF ha tenido éxito llama a tesseract OCR para que lea los datos.
20     * @param String resultadoOpenCV - String JSON con las imágenes recortadas por OpenCV.
21     */
22    exitoOpenCVFrente: function(resultadoOpenCV){
23        //Asigna las imágenes a variables temporales
24        doc.imagenCompletaFrenteTemporal = resultadoOpenCV["0"];
25        doc.fotoTemporal = resultadoOpenCV["1"];
26        doc.firmaTemporal = resultadoOpenCV["2"];
27
28        //Llama a tesseract OCR para leer los datos de la parte frontal.
29        tesseract.leerDatosFrente(extraccion.exitoTesseractFrente, extraccion.fallo, resultadoOpenCV["3"]);
30    },
31
32    /**
33     * Si la llamada a tesseract en el método exitoTesseractFrente ha tenido éxito, escribe los datos leídos
34     * y los asigna a las variables correspondientes.
35     * @param String resultadoTesseract - String JSON con los datos leídos por Tesseract.
36     */
37    exitoTesseractFrente: function(resultadoTesseract){
38        //Asigna cada dato a su variable correspondiente.
39        doc.apellido1Frente = resultadoTesseract["1"];
40        doc.apellido2Frente = resultadoTesseract["3"];
41        doc.nombreFrente = resultadoTesseract["1"],substring(0,1);
42        doc.seorFrente = resultadoTesseract["1"],substring(1,4);
43        doc.nacionalidadFrente = resultadoTesseract["7"],substring(0,1);
44        doc.fechaNacimientoFrente = resultadoTesseract["9"];
45        doc.numeroSerieFrente = resultadoTesseract["11"];
46        doc.fechaVencimientoFrente = resultadoTesseract["13"];
47
48        //Cambia el formato de la fecha para que sea posible compararla con la del reverso (DD/MM/AA).
```

Ilustración 43: Parte del código de extracción del DNI y reconocimiento de los caracteres

Por otro lado, se encarga de comprobar que todos los campos del documento sean correctos y de esta manera validar el DNI. Primero, comprueba que la letra del DNI coincide con el resultado de calcular el módulo 23 del número (cada uno de los 24 resultados posibles tiene asignada una letra). Despues, calcula los cuatro números de control y compara el resultado con los que se encuentran en el reverso del documento (los cálculos para validar estos números son algo más complicados). Por último, comprueba que todos los datos de la parte frontal y el reverso coincidan y que el DNI no esté caducado.

```

92 /**
93 * Comprueba que los dígitos de control sean correctos.
94 */
95 comprouebaDigitosControl: function(){
96     var fechaNacimiento = aux.cambiarFormatoFecha(doc.fechaNacimientoReverso, "DD/MM/AA");
97     var fechaVencimiento = aux.cambiarFormatoFecha(doc.fechaVencimientoReverso, "DD/MM/AA");
98
99     //Llamo a las funciones correspondientes para calcular cada dígito.
100    var resultadoDigito1 = validacion.calculaDigitoControl(doc.numeroSerieReverso, 0);
101    var resultadoDigito2 = validacion.calculaDigitoControl(fechaNacimiento, 1);
102    var resultadoDigito3 = validacion.calculaDigitoControl(fechaVencimiento, 2);
103    var resultadoDigito4 = validacion.calculaDigito4(doc.numeroSerieReverso, resultadoDigito1, doc.numeroONI,
104                                                    fechaNacimiento, resultadoDigito2, fechaVencimiento, resultadoDigito3);
105
106    //Comprueba que todos los dígitos tengan el valor calculado.
107    if(resultadoDigito1==doc.digitoControl1&&
108       resultadoDigito2==doc.digitoControl2&&
109       resultadoDigito3==doc.digitoControl3&&
110       resultadoDigito4==doc.digitoControl4){
111        doc.resultadoDigitos = "true";
112    } else{
113        doc.resultadoDigitos = "false";
114    }
115
116 /**
117 * Calcula el valor de los 3 primeros dígitos.
118 * @param String campo - Campo a partir del que calcular el dígito.
119 * @param int numeroDigito - Índice si es el primero, el segundo o el tercer dígito.
120 * @return String resultado - Valor el dígito calculado.
121 */
122 calculaDigitoControl: function(campo, numeroDigito){
123     //Asigna variables auxiliares.
124     var resultado = 0;
125     var charTemporal = "0";
126     var j = 0;
127
128     //Calcula el valor de cada dígito.
129     for(i=0; i<campo.length; i++){
130         charTemporal = campo.substring(i, i+1);
131         if(numeroDigito==0){
132             if(i==0||i==1||i==2)
133                 charTemporal = validacion.calculaCaracterNumeroSerie(charTemporal);
134         }
135     }
136
137     i=13;
138
139 }

```

Ilustración 44: Parte del código de validación

5.3.2 Parte nativa

Es la parte de la aplicación encargada de obtener el DNI de la fotografía tomada con la cámara y a partir de este realizar la lectura de los campos presentes en el frente y el reverso del documento. Esta parte se ha realizado por separado para la versión de Android (en Java) y para la versión de iOS (en Objective-C). Sin embargo, ambas realizan exactamente la misma función, con pequeños cambios debidos a las diferencias entre las plataformas y los lenguajes utilizados.

En primer lugar, está la clase OpenCVPlugin, que se encarga del reconocimiento del DNI y del procesado de la imagen. Para ello utiliza las funciones proporcionadas por la librería OpenCV. Primero carga el archivo de cascada correspondiente a la parte del DNI que se quiere reconocer, para después utilizarlo para extraer el documento de la imagen y guardarlo en una variable separada. Más tarde recorta las zonas de interés y realiza un procesado de ellas que tiene como objetivo mejorar la precisión del OCR. Por último retorna las zonas de interés ya procesadas.

```

138     /**
139      * Método que detecta el DNI dentro de la imagen y lo recorta.
140      * Despues corta los campos y los retorna junto con la imagen cortada.
141      * @param String tipo - Indica si es la Imagen de la parte frontal o del reverso del DNI.
142      * @param Mat imagen - Imagen completa capturada por la cámara.
143      */
144     private void detectarYCortar(String tipo, Mat imagen){
145         //Crea una matriz de rectángulos
146         MatOfRect dnisDetectados = new MatOfRect();
147         //Detecta los rectángulos de los DNI en la imagen y los pasa a la matriz.
148         if(tipo.equals("frente")){
149             cascada.detectMultiScale(imagen, dnisDetectados, 1.05, 2, 0, new Size(imagen.width()*0.35,imagen.height()*0.35), new Size());
150         } else if (tipo.equals("reverso")){
151             cascada.detectMultiScale(imagen, dnisDetectados, 1.10, 6, 0, new Size(imagen.width()*0.35,imagen.height()*0.35), new Size());
152         }
153         Mat imagenCortada = Imagen;
154         if(dnisDetectados.toArray().length>0){ //Si se ha detectado al menos 1 DNI.
155             rectanguloDNI = dnisDetectados.toArray()[dnisDetectados.toArray().length-1];
156             //Relacion de altura entre la imagen completa y el DNI detectado.
157             int porcentaje = rectanguloDNI.height*100/imagen.height();
158             //Realiza correcciones en la detección dependiendo de la distancia a la que se ha tomado la foto.
159             if(tipo.equals("frente")){
160                 if(porcentaje<=35&porcentaje>=45){
161                     recortarRectangulo(170, 50, -100, -140);
162                 } else if(porcentaje>=45&porcentaje<=60){
163                     recortarRectangulo(30, 0, -50, 0);
164                 } else if(porcentaje>=60&porcentaje<=70){
165                     recortarRectangulo(-50, -40, 100, 80);
166                 } else if(porcentaje>=70&porcentaje<=100{
167                     recortarRectangulo(-100, -70, 270, 170);
168                 }
169             } else if(tipo.equals("reverso")){
170                 if(porcentaje>=50&porcentaje<=70){
171                     recortarRectangulo(-70, -90, 140, 120);
172                 } else if(porcentaje>=70&porcentaje<=100{
173                     recortarRectangulo(-80, -100, 350, 200);
174                 }
175             }
176         }
177     }
178 }
```

Ilustración 45: Parte del código de la clase OpenCVPlugin en Java

```

78     /**
79      * Detecta el DNI dentro de la imagen, lo recorta y extrae los campos de interés.
80      * @param Mat imagen - Imagen completa en formato Mat.
81      * @param String tipo - Indica si es la imagen de la parte frontal o del reverso del DNI.
82      */
83     - (void) detectarYCortar:(cv::Mat) imagen deTipo:(NSString*) tipo{
84
85         std::vector<cv::Rect> dnisDetectados;
86         cascada.detectMultiScale(imagen, dnisDetectados, 1.05, 1, 0|CV_HAAR_SCALE_IMAGE, cv::Size(imagen.cols*0.30, imagen.rows*0.30));
87
88         cv::Mat imagenCortada = imagen.clone();
89
90         if(dnisDetectados.size()>0){
91             cv::Rect rectanguloDNI = dnisDetectados[dnisDetectados.size()-1];
92             cv::Mat(imagen, rectanguloDNI).copyTo(imagenCortada);
93         }
94
95         [imagenes removeAllObjects];
96         [self anadirImagen:imagenCortada conLlave:@“0”];
97         cv::Mat imagenPreprocesada = [self preprocesar:imagenCortada deTipo:tipo];
98
99         if([tipo isEqualToString:@“frente”]){ //Recorta los campos del frente del DNI.
100
101             //Recorta la foto.
102             [self recortarCampo:imagenCortada
103                 puntoIni:*new cv::Point(imagenCortada.cols*0.65, imagenCortada.rows*0.37)
104                 puntoFin:*new cv::Point(imagenCortada.cols, imagenCortada.rows) conLlave:@“1”];
105
106             //Recorta la firma.
107             [self recortarCampo:imagenPreprocesada
108                 puntoIni:*new cv::Point(imagenPreprocesada.cols*0.24, imagenPreprocesada.rows*0.74)
109                 puntoFin:*new cv::Point(imagenPreprocesada.cols*0.67, imagenPreprocesada.rows) conLlave:@“2”];
110
111             //Recorta los campos del frente.
112             [self recortarCampos:imagenPreprocesada
113                 puntoIni:*new cv::Point(imagenPreprocesada.cols*0.24, imagenPreprocesada.rows*0.74)
114                 puntoFin:*new cv::Point(imagenPreprocesada.cols*0.67, imagenPreprocesada.rows) conLlave:@“3”];
115         }
116     }
117 }
```

Ilustración 46: Parte del código de la clase OpenCVPlugin en Objective-C

En segundo lugar tenemos la clase TesseractPlugin, que realiza el reconocimiento de caracteres utilizando la librería Tesseract OCR. Su método principal va leyendo los diferentes campos del DNI uno a uno y guardándolos en un array, para después retornarlo. Para aumentar al máximo la precisión se aprovecha el hecho de que se vaya leyendo línea a línea (un campo por línea) para delimitar la lista de caracteres posibles para cada campo. De esta forma en los campos

como el nombre o los apellidos solo lee letras, mientras que en los de fechas solo números. Para el reconocimiento se utiliza la versión oficial del idioma español.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer (left):** Shows the project structure with files like `OpenCVPlugin.java`, `TesseractPlugin.java`, `com.netkia.android`, and `OpenCVPlugin.java`.
- Code Editor (right):** Displays the `TesseractPlugin.java` file containing Java code for interacting with the Tesseract library.
- Bottom Bar:** Includes tabs for `Problems`, `Javadoc`, `Declaration`, `Console`, and `LogCat`.

```
Java - DniValidator/src/com/netkia/android/TesseractPlugin.java - Eclipse Platform

101 //Asigna la variable del Path de Tesseract.
102 String TESSBASE_PATH = Environment.getExternalStorageDirectory() + "/tesseract";
103
104 //Inicializa tess-two.
105 TessBaseAPI baseApi = new TessBaseAPI();
106 baseApi.init(TESSBASE_PATH, "spa");
107 baseApi.setImage(imagen);
108
109 //Define variables auxiliares.
110 JSONObject camposTextoJson = new JSONObject();
111 String whiteList = "";
112 String blackList = "";
113 Pix pixaTemporal = baseApi.getTextlines();
114 Pix pixTemporal = new Pix(0);
115 String resultado = "";
116
117 //Comprueba que se han detectado el numero de lineas correcto y si no devuelve un mensaje de error.
118 if(tipo.equals("fronte")&pixaTemporal.size()==14){
119     callbackContext.error("No se ha podido extraer datos de la parte frontal del DNI debido " +
120                           "a la calidad de la imagen. Por favor, inténtalo de nuevo.");
121 }
122 else if(tipo.equals("reverso")&pixaTemporal.size()==3){
123     callbackContext.error("No se han podido extraer datos del reverso del DNI debido " +
124                           "a la calidad de la imagen. Por favor, inténtalo de nuevo.");
125 }
126
127 //Añade cada linea de texto detectada al ArrayList a devolver.
128 for(int i=0; i<pixaTemporal.size(); i++){
129     whiteList = getWhiteList(tipo,i);
130     blackList = getBlackList(tipo,i);
131     baseApi.setVariable(TessBaseAPI.VAR_CHAR_WHITELIST, whiteList);
132     baseApi.setVariable(TessBaseAPI.VAR_CHAR_BLACKLIST, blackList);
133     pixTemporal = pixaTemporal.getPix(i);
134     baseApi.setImage(pixTemporal);
135     resultado = baseApi.getUTF8Text();
136     resultado = resultado.replaceAll(blackList, "");
137
138     try {
139         camposTextoJson.put(Integer.toString(i), resultado);
140     } catch (JSONException e) {
141         e.printStackTrace();
142     }
143 }
144
145 baseApi.end();


```

Ilustración 47: Parte del código de la clase *TesseractPlugin* en Java

Ilustración 48: Parte del código de la clase `TesseractPlugin` en Objective-C

5.4 Implementación de la capa de acceso a datos

La implementación de la capa de datos se compone de 3 partes: una base de datos MySQL, un servicio REST y la parte de datos de la aplicación web.

5.4.1 Base de datos MySQL

Los datos de los DNI capturados por la aplicación se guardan en una base de datos MySQL para su posterior recuperación siempre que se requiera. La base de datos tiene solo una tabla documentos, que a su vez tiene campos para todos los datos del DNI y para las 4 imágenes.

Mientras que los datos se guardan directamente, guardar las 4 imágenes en base 64 sería un gran problema debido a la inmensa longitud de estos String. Para evitar este problema se genera un id automático para cada imagen y se guarda solo el id en la base de datos. A su vez se convierte la imagen a PNG y se guarda en disco con el nombre de su id. Cuando se requiere su recuperación simplemente se carga la imagen en disco que tenga el nombre del campo de la imagen.

5.4.2 API REST

Tanto la API REST como la aplicación web están implementadas en el mismo proyecto Node.js desarrollado con ayuda del framework Express.js. El servicio REST hace de intermediario entre la aplicación móvil y la base de datos MySQL. Contiene funciones para añadir un nuevo documento (POST), ver la lista de documentos (GET), ver todos los datos de un documento (GET) y borrar un documento (DELETE).

Al ser un servicio REST, cada una de estas funciones está disponible enviando el mensaje HTTP indicado a la dirección correspondiente. El servicio se encargará después de realizar las transacciones necesarias en la base de datos y devolver el resultado en formato JSON. Este JSON luego será procesado por la aplicación para mostrar el resultado en pantalla.

The screenshot shows a Sublime Text window with the following details:

- File Path:** ~/node/dniExpress/routes/apiRest.js
- File Type:** www, node - Sublime Text (UNREGISTERED)
- Code Editor:** The main pane displays the `apiRest.js` file content.
- File List:** The sidebar shows the project structure with files like app.js, documentos.js, cliente.js, routes/apiRest.js, and views/documents.jade.
- Code Content:**

```

1 //Incluye los modulos necesarios
2 var mysql = require('mysql');
3 var fs = require('fs');
4 var express = require('express');
5 var connectionpool = mysql.createPool({
6   host : 'localhost',
7   user : 'root',
8   password : 'netkia',
9   database : 'documentos',
10  port: 3306
11 });
12
13 /**
14  * Devuelve la lista de documentos.
15  * @param peticion - Petición HTTP.
16  * @param respuesta - Respuesta HTTP.
17 */
18 exports.get_documentos = function(peticion, respuesta){
19
20   //Define la variable en la que se guarda la respuesta en formato JSON.
21   var respuestaJson = [];
22
23   //Extrae los datos con una consulta y los guarda en la variable lista.
24   connectionpool.getConnection(function(err, connection){
25     if(err){
26       connection.query('SELECT id, nif, nombre, apellido1, apellido2 FROM documentos',
27         function(error, resultado){
28           connection.release();
29           if(error){
30             respuesta.setHeader('Content-Type', 'application/json');
31
32             if(resultado.length > 0){
33               for(var i in resultado){
34                 respuestaJson.push({
35                   result: "ok",
36                   id: resultado[i].id.toString(),
37                   nif: resultado[i].nif,
38                   nombre: resultado[i].nombre,
39                   apellido1: resultado[i].apellido1,
40                   apellido2: resultado[i].apellido2
41                 });
42             }
43             respuesta.status(200).send(respuestaJson);
44           } else{
45             respuestaJson = {result: "No hay documentos"};
46             respuesta.status(204).send(respuestaJson);
47           }
48         }
49       }
50     }
51   });
52 }

```
- Status Bar:** Shows the current tab, file size, and line number (Line 337, Column 18).

Ilustración 49: Parte del código de la API REST

5.4.3 Aplicación web

Al tener simplemente la funcionalidad de mostrar la información guardada en la base de datos, la aplicación web simplemente tiene capas de presentación y datos. La capa de datos engloba la mayoría de la aplicación web y se encarga de definir las URIs desde las que se podrá ver cada vista y las funciones que realizarán las transacciones con la base de datos.

The screenshot shows a Sublime Text window with multiple tabs open. The left sidebar displays the project structure:

```

~/node/dniExpress/routes/documentos.js (www, node) - Sublime Text (UNREGISTERED)
GROUP 1
  app.js
  documentos.js
  documentos.jade
  apiRest.js
  cliente.js
GROUP 2
  FOLDERS
  www
    node
      chatExpress
      chatPrueba
      dniExpress
        bin
        node_modules
      public
        bootstrap
        images
          javascripts
            cliente.js
            graficas.js
            jquery.js
          stylesheets
            style.css
      routes
        apiRest.js
        documentos.js
        index.js
        users.js
      views
        documentos.jade
        documentos.jade
        error.jade
        estadisticas.jade
        filtro.jade
        index.jade
        layout.jade
        README.md
        app.js
        npm-debug.log
        package.json
      dnINode
      estatica

```

The main editor area shows the code for `documentos.js`:

```

1 //Incluye los modulos necesarios
2 var mysql = require('mysql');
3 var fs = require('fs');
4 //Crea la pool de conexiones a la base de datos.
5 var connectionpool = mysql.createPool({
6   host : 'localhost',
7   user : 'root',
8   password : 'netkia',
9   database : 'documentos',
10  port: 3306
11 });
12
13 /**
14  * Devuelve la lista de documentos.
15  * @param peticion - Petición HTTP.
16  * @param respuesta - Respuesta HTTP.
17  */
18 exports.get_documentos = function(peticion, respuesta){
19   //Define la variable en la que se guarda la lista de documentos.
20   var lista = [];
21
22   //Conecta con la base de datos y realiza el select.
23   connectionpool.getConnection(function(err, connection){
24     if(err){
25       connection.query('SELECT id, nif, nombre, apellido1, apellido2 FROM documentos',
26         function(error, resultado){
27           connection.release();
28           if(error){
29             if(resultado.length > 0){
30               for(var i in resultado){
31                 lista.push({
32                   id: resultado[i].id,
33                   nif: resultado[i].nif,
34                   nombre: resultado[i].nombre,
35                   apellido1: resultado[i].apellido1,
36                   apellido2: resultado[i].apellido2
37                 });
38               }
39             respuesta.render('documentos', {
40               lista: lista
41             });
42           } else{
43             respuesta.render('error');
44           }
45         }
46       );
47     }
48   });

```

Ilustración 50: Parte del código de la aplicación web

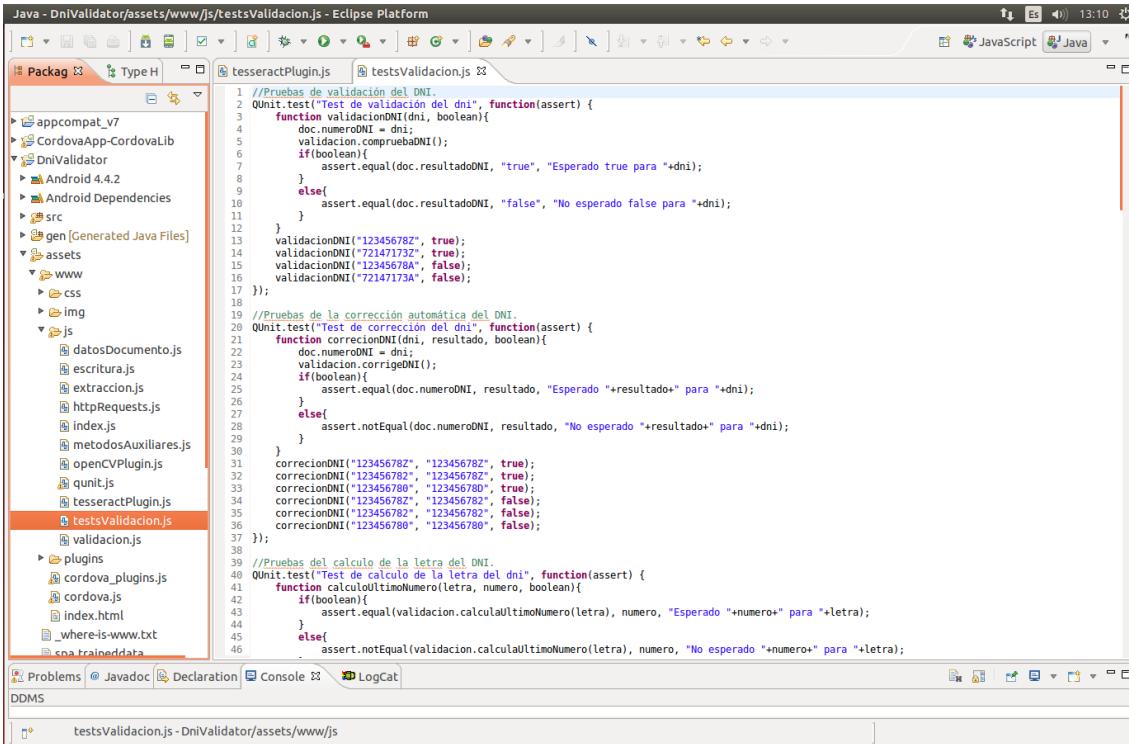
6. Pruebas

En el desarrollo de cualquier sistema software es necesario realizar una serie de pruebas para verificar y validar que todo funciona como se espera y no existen errores. En este proyecto se han realizado primero pruebas unitarias, seguidas de pruebas de integración, sistema y aceptación. En este apartado se detallan cada una de estas pruebas, además de las herramientas utilizadas para realizarlas.

6.1 Pruebas unitarias

Las pruebas unitarias permiten comprobar el correcto funcionamiento de los métodos y funcionalidades del software por separado. Verifican que todas las funciones probadas producen las salidas esperadas para cada entrada introducida.

Solo se han realizado pruebas unitarias de manera formal en las funciones encargadas de la validación y comprobación del DNI. Esto es debido a que se ha considerado que el resto de funciones, como por ejemplo el proceso de detección del DNI, se realizan de forma muy secuencial y es menos probable que produzcan errores. Para estas últimas, simplemente se han ido mostrando los valores de algunas variables por consola.



The screenshot shows the Eclipse Platform interface with the Java-DniValidator project open. The left sidebar displays the project structure, including packages like appcompat_v7, CordovaApp-CordovaLib, DniValidator, and various JavaScript files under js. The main editor window contains a file named testsValidacion.js, which contains QUnit test cases for validation functions. The code includes assertions for validating DNI numbers and checking for automatic correction. The bottom of the screen shows the Eclipse interface with tabs for Problems, Javadoc, Declaration, Console, and LogCat.

```
//Pruebas de validación del DNI.
QUnit.test("Test de validación del dni", function(assert) {
    function validacionONI(dni, boolean){
        doc.numeroONI = dni;
        validacion.compruebaONI();
        if(boolean){
            assert.equal(doc.resultadoONI, "true", "Esperado true para "+dni);
        } else{
            assert.equal(doc.resultadoONI, "false", "No esperado false para "+dni);
        }
    }
    validacionONI("12345678Z", true);
    validacionONI("72147173Z", true);
    validacionONI("12345678A", false);
    validacionONI("72147173A", false);
});

//Pruebas de la corrección automática del DNI.
QUnit.test("Test de corrección del dni", function(assert) {
    function correccionONI(dni, resultado, boolean){
        doc.numeroONI = dni;
        validacion.corrigeONI();
        if(boolean){
            assert.equal(doc.numeroONI, resultado, "Esperado "+resultado+" para "+dni);
        } else{
            assert.notEqual(doc.numeroONI, resultado, "No esperado "+resultado+" para "+dni);
        }
    }
    correccionONI("12345678Z", "12345678Z", true);
    correccionONI("12345678Z", "12345678", true);
    correccionONI("12345678Z", "12345678A", false);
    correccionONI("12345678Z", "123456782", false);
    correccionONI("12345678Z", "123456782", false);
    correccionONI("12345678B", "12345678B", false);
});

//Pruebas del cálculo de la letra del DNI.
QUnit.test("Test de cálculo de la letra del dni", function(assert) {
    function calculaUltimoNúmero(letra, numero, boolean){
        if(boolean){
            assert.equal(validacion.calculaUltimoNúmero(letra), numero, "Esperado "+numero+" para "+letra);
        } else{
            assert.notEqual(validacion.calculaUltimoNúmero(letra), numero, "No esperado "+numero+" para "+letra);
        }
    }
});
```

Ilustración 51: Parte del código de las pruebas unitarias

En la imagen anterior se puede ver parte del código encargado de realizar las pruebas unitarias con ayuda de QUnit. Como se puede comprobar, para cada función probada se introducen varios valores como argumentos y se verifica que retornan las salidas adecuadas. Para las funciones

que devuelven boolean se han utilizado los métodos de QUnit *assert.ok()* y *assert.notOk()* y para el resto se han usado *assert.equal()* y *assert.notEqual()*.

Para la selección de los valores de entrada se han utilizado particiones de equivalencia. Para cada prueba unitaria se han determinado varias clases válidas y varias no válidas y se han escogido valores tratando de cubrir los máximos casos posibles.

6.2 Pruebas de integración

Las pruebas de integración se realizan después de verificar con las pruebas unitarias el correcto funcionamiento de las partes del sistema por separado. Tienen como fin comprobar que estas partes funcionan también cuando son probadas en conjunto.

En cada iteración realizada en el desarrollo del proyecto se han ido probando los distintos componentes implementados en dicha iteración para verificar su funcionamiento en grupo. En la última iteración se ha probado finalmente la aplicación al completo. Al igual que algunas de las pruebas unitarias, se han realizado principalmente mostrando los valores de determinadas variables por consola.

6.3 Pruebas de sistema

Una vez realizadas todas las pruebas anteriores, es necesario comprobar el funcionamiento de del sistema desde un punto de vista no funcional. Como los errores en la funcionalidad suelen salir a la luz en las pruebas unitarias y de integración, las pruebas de sistema sirven principalmente para comprobar la satisfacción de los requisitos no funcionales.

6.3.1 Pruebas de rendimiento

Como se explica en el apartado de implementación de la capa de presentación, toda ella está contenida en un solo fichero HTML. Gracias a esto la navegación por los menús de la aplicación es prácticamente instantánea en todos los terminales en los que se ha probado: un Sony Xperia Z3, un Samsung Galaxy S3 Mini y un iPhone 4.

Los dos únicos procesos que no son instantáneos son las transacciones de la base de datos y el reconocimiento del DNI y los caracteres. El primero no supone ningún problema ya que no bloquea el uso de la aplicación y aunque lo hiciese, no suele superar los 5 segundos. El segundo si es bloqueante, pero se ha cumplido a la perfección el requisito de rendimiento que establece que no se debe tardar más de 20 segundos. Se ha verificado realizando el reconocimiento varias veces y anotando el tiempo empleado en hacerlo. Como resultado se han obtenido 10 segundos de media.

6.3.2 Pruebas de seguridad

Al estar todos los datos de los DNI capturados por la aplicación guardados en una base de datos en un servidor, se puede garantizar mucho más fácilmente su integridad. El hecho de no guardar los datos directamente en el dispositivo y ofrecerlos a través de una API REST, permite impedir que se realicen transacciones no deseadas sobre la base de datos.

6.5 Pruebas de aceptación

Las pruebas de aceptación tienen como fin comprobar la satisfacción de todos los requisitos y objetivos planteados por el cliente.

Estas pruebas se han realizado por cada iteración o nuevo módulo del proyecto desarrollado. En cada una se ha dado a probar la aplicación a varias personas, algunas con conocimientos de desarrollo software y otras sin ellos. A partir de estas pruebas se han ido recopilando las opiniones de todas estas personas y realizando los cambios necesarios en la aplicación. Por ejemplo, en un principio no estaba pensado desarrollar una aplicación web, pero a partir de los resultados de estas pruebas finalmente se decidió hacerlo para aportar una forma de acceder a los datos desde un equipo tradicional.

7. Conclusiones y trabajos futuros

7.1 Conclusiones

Una vez se da por finalizado el proyecto es momento de analizar si se han alcanzado todos los objetivos planteados en un principio y resumir el trabajo realizado. El objetivo principal de este proyecto era el de desarrollar una aplicación que permitiese la validación y captura de datos del DNI electrónico español desde un Smartphone. Se puede considerar que este objetivo ha sido totalmente cumplido ya que la aplicación realiza esta tarea de manera satisfactoria.

La aplicación está disponible tanto en sistemas Android como iOS y para cada uno de ellos dispone de una interfaz diferenciada y adaptada a las líneas de diseño establecidas por cada plataforma. Esta interfaz permite acceder al contenido y utilidad de la aplicación de una forma sencilla e intuitiva. Además, los datos capturados por la aplicación se almacenan en una base de datos en un servidor y se puede acceder a ellos tanto desde la aplicación móvil como desde una aplicación web.

Con respecto a la precisión de la validación, si bien no siempre se capturan a la perfección los datos del DNI, se han corregido todos los errores encontrados que pudiesen bloquear la aplicación o dar por válido un documento que no lo fuese. En el apartado de mejoras en la precisión y rendimiento de la aplicación es probablemente donde más horas se han dedicado, ya que es un aspecto crítico en el funcionamiento del sistema.

Para el desarrollo del sistema se han seguido las actitudes propias de un ingeniero del software, utilizando una metodología de desarrollo y documentando todo el proceso en esta memoria. Además se ha utilizado un arquitectura en 3 capas que ha permitido hacer los módulos de la aplicación independientes unos de otros.

Por último, cabe destacar que ha sido una experiencia muy interesante y útil tanto desde el punto de vista laboral como personal. Gracias al desarrollo de este proyecto he podido aprender una gran cantidad de cosas nuevas además de poner en práctica las ya aprendidas a lo largo de los años de carrera. La realización de un proyecto desde cero y el hecho de desarrollar este proyecto en una empresa me han reportado una gran satisfacción personal y han aumentado en gran medida mi interés por el mundo del desarrollo software.

7.2 Trabajos futuros

7.2.1 Mejora del reconocimiento de caracteres

Una de las cosas que podrían mejorarse del proyecto sería la precisión del reconocimiento de caracteres. Si bien funciona bastante satisfactoriamente, lo hace bajo una serie de condiciones.

El principal motivo por el que disminuye la precisión del OCR es el tamaño de los caracteres presentes en el DNI y el poco contraste de algunos de ellos con el fondo. De esta manera, para el correcto funcionamiento del OCR es necesario que el dispositivo posea una cámara bastante

potente, que la imagen esté totalmente enfocada y sin brillos y que haya buena iluminación, de forma que el texto sea perfectamente legible.

La versión Android de la aplicación se ha desarrollado en un Sony Xperia Z3 con resultados muy satisfactorios debido a la calidad de la cámara de este terminal. Sin embargo, probando esta misma versión en un Samsung Galaxy S3 Mini, los resultados son algo peores. La versión iOS no ha podido ser probada correctamente, ya que solo se disponía de un iPhone 4, el cual posee una cámara bastante menos potente que la del Z3 y por lo tanto el OCR era muy poco preciso.

Se deja como trabajo futuro porque de momento no es posible hacer mucho más para mejorarlo. Con el tiempo, tanto los sistemas de reconocimiento de caracteres como las cámaras de los dispositivos mejorarán, permitiendo extraer el texto del DNI de manera más precisa.

7.2.2 Implementación de la versión de Windows Phone

Aunque Windows Phone es una plataforma que está ganando cada vez más usuarios y por lo tanto podría ser interesante implementar esta versión de la aplicación, finalmente no ha sido posible debido a diversos impedimentos.

En primer lugar, la librería OpenCV no dispone de versión oficial para Windows Phone y aunque existen wrapper para el framework .NET (en el que se incluye Windows Phone) como EmguCV, son de pago y obligarían a cambiar parte de la funcionalidad de la aplicación.

En segundo lugar, tampoco existe ninguna versión de Tesseract OCR para Windows Phone, ni oficial ni de terceros. A pesar de que Microsoft posee una librería OCR gratuita, es bastante menos potente que Tesseract y ofrece resultados notablemente peores. Además no se disponía de ningún dispositivo con Windows Phone sobre el que probar la aplicación.

Al igual que en el apartado anterior, de momento no se puede realizar esta mejora, por lo que habrá que esperar a que los recursos necesarios para implementarla estén disponibles.

7.2.3 Mejora de la interfaz gráfica

Aunque se considera que se ha llegado a desarrollar una interfaz gráfica bastante sencilla e intuitiva a la vez de eficiente, en la parte estética siempre es posible realizar mejoras. Sin embargo, este tema queda fuera de las competencias de un ingeniero informático y más dentro de las de un diseñador gráfico.

8. Bibliografía

- [1] Sommerville. 2012. *Ingeniería del Software*. 9a Edición, Addison-Wesley. 2012.
- [2] Phonegap Beginner's Guide. Andrew, Lunny. Editor: Packt Publishing (2011).
- [3] Apache Cordova Documentation: <http://cordova.apache.org/docs/en/edge/index.html>
- [4] W3Schools: <http://www.w3schools.com>
- [5] Douglas, Crockford. *Javascript: The Good Parts*. O'Reilly, 1st Edition. 2008.
- [6] OpenCV Documentation: <http://docs.opencv.org>
- [7] Tesseract OCR: <https://code.google.com/p/tesseract-ocr>
- [8] Android Developers: <http://developer.android.com/index.html>
- [9] iOS Dev Center: <https://developer.apple.com/devcenter/ios/index.action>
- [10] Node.js Documentation: <https://nodejs.org/documentation>
- [11] Express.js API: <http://expressjs.com/4x/api.html>
- [12] Java API: <http://docs.oracle.com/javase/7/docs/api>
- [13] Richard N. Taylor, Nenad Medvidovic y Eric M. Dashofy, "Software Architecture: Foundations, Theory, and Practice". 2009.
- [14] J.Arlow e I. Neustadt. *UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design*. Addison-Wesley. 2005.
- [15] D. Pilone. *UML 2.0 in a Nutshell*. O'Reilly. 2005.
- [16] OpenCV haartraining (Rapid Object Detection With A Cascade of Boosted Classifiers Based on Haar-like Features) : <http://note.sonots.com/SciSoftware/haartraining.html>