Facultad de Ciencias

DESARROLLO DE UNA APLICACIÓN WEB PARA LA ESTIMACIÓN DEL NÚMERO DE PÁRTICULAS DE UNA IMAGEN

(WEB APPLICATION FOR ESTIMATING THE NUMBER OF PARTICLES OF AN IMAGE)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: ANDRÉS BARRADO MARTÍN

Director: MARCOS CRUZ RODRÍGUEZ

Co-Director: STEVEN VAN VAERENBERGH

SEPTIEMBRE - 2020

Índice de contenido

Índice de figuras

Índice de tablas

Índice de ecuaciones

Agradecimientos

En primer lugar, agradecer a mi familia todo el apoyo recibido durante estos años, ya que sin ellos no habría sido posible.

En segundo lugar, a todos mis amigos y compañeros de clase por hacer posible disfrutar durante todos estos años y por la ayuda que me han brindado cuando la he necesitado.

Por último, a todos los profesores que he tenido durante la carrera, en especial a los de la mención software que me han inculcado los conocimientos necesarios para desenvolverme en la vida laboral y a Marcos Cruz y Steven Van Vaerenbergh por darme la oportunidad de desarrollar este proyecto tan interesante.

Resumen

Hoy en día, hay sistemas para el cálculo de partículas de una imagen que funcionan muy bien pero que en ciertos casos pueden dan un elevado porcentaje de error debido a grandes aglomeraciones de partículas con fondos poco homogéneos, solapamiento de partículas, iluminación poco homogénea etc.

De cara a ofrecer una alternativa al cálculo automático de partículas, se realizó un estudio por parte de la universidad de Cantabria llamado el método Countem que para estos casos en los que el cálculo automático da errores ofrece un porcentaje de error menor. Este método se basa en el cálculo de partículas con un conteo manual de forma parcial. Por ello, se realizó una aplicación de escritorio desarrollada en Python, la cual se debe descargar previamente para su uso.

Con el objetivo de facilitar el acceso a los usuarios y en consecuencia aumentar el número de estos, se ha decidido desarrollar una aplicación web para estimar el número de partículas y proporcionar un porcentaje de error sobre este cálculo.

Palabras clave: Ventana de muestreo, Box Side, Quadrat Side

Abstract

Nowadays, there are systems for calculating particles in an image that work very well but in certain cases they can give a high percentage of error due to large agglomerations of particles with not very homogeneous backgrounds, overlapping of particles, inhomogeneous lighting, etc.

In order to offer an alternative to the automatic calculation of particles, a study was carried out by the University of Cantabria called the Countem method that for these cases in which the automatic calculation gives errors offers a lower percentage of error. This method is based on the calculation of particles with a manual partial count. For this reason, a desktop application developed in Python was made, which must be previously downloaded for use.

Seeking to facilitate access to users and consequently increase their number, it has been decided to develop a web application to estimate the number of particles and provide a percentage of error on this calculation.

Keywords: Quadrat, Box Side, Quadrat Side

1. Introducción

El tamaño de una población es el número total de elementos o partículas de características individuales (por ejemplo, organismos) que constituyen la población. Si este último se encuentra en un área abierta, entonces sus elementos pueden, en principio, identificarse y contarse a partir de fotografías aéreas. A menudo, la reacción natural es contar todas las partículas, tarea que suele realizarse a mano. En la práctica, sin embargo, esto puede ser soportable para tamaños de población del orden de 1000 partículas, aunque es tedioso, lento, muy dependiente de la habilidad del operador y difícil de verificar. Desafortunadamente, en este contexto generalmente faltan estrategias de muestreo adecuadas. La densidad de población generalmente se estima con ventanas de muestreo, pero no se dan criterios claros sobre cómo colocar las ventanas de muestreo, cómo corregir los efectos de borde que surgen en el conteo de cuadrantes, etc. La falta de un mecanismo de muestreo bien definido excluye no solo la estimación insesgada del tamaño de una partícula, sino también una predicción fiable del correspondiente error de la varianza.

El análisis automático de imágenes funciona bien, pero en algunos casos pueden dar errores de cálculo. Por ejemplo, la detección automática de rostros humanos en imágenes fijas se ha estudiado detenidamente. Sin embargo, se sabe que los detectores de rostro humano de última generación funcionan mal debido a una serie de artefactos como grandes variaciones de pose e iluminación, variaciones de expresión, desenfoque y baja resolución de imagen.

Aquí proponemos un muestreo sistemático basado en el diseño para estimar el tamaño de la población con errores bajos y predecibles. En términos matemáticos, el problema consiste en estimar el tamaño finito N de una población limitada 'Y' de partículas en un plano de observación. En general, una partícula se define como un subconjunto compacto y conectado separado de otras partículas. En el contexto de la multitud humana, una partícula es la proyección plana de una cabeza humana, o un fragmento claramente distinguible de la misma.

1.1. El método CountEm

Recientemente se propuso un método de estimación del tamaño de la población no sesgado (en lo sucesivo, método CountEm)[2]. La única limitación práctica es el requisito básico de que todas las partículas de la población deben ser identificables sin ambigüedades para el recuento manual en la imagen considerada. Se basa en principios bien conocidos de muestreo geométrico para estereología que se han aplicado previamente a la microscopía cuantitativa. La idea principal es muestrear y contar adecuadamente entre 50 y 200 partículas para estimar poblaciones de cualquier tamaño y distribución espacial. El muestreo sistemático se realiza con una cuadrícula aleatoria uniforme de ventanas de muestreo, ver Figura 1. La regla de la línea prohibida se usa para evitar el sesgo debido a los efectos de borde: una partícula se cuenta solo si toca la ventana de muestreo, pero no golpee la línea prohibida extendida de la ventana de muestreo (ver Figura 2). El estimador del tamaño de la población, es el número total de partículas muestreadas, multiplicado por el período de muestreo.

En la imagen que se muestra a continuación se puede ver una rejilla de ventanas de muestreo que se superpuso uniformemente al azar para estimar el número total de espectadores en la imagen. Aplicando la regla de línea prohibida para eliminar efectos de borde en el conteo manual, solo las cabezas marcadas con puntas de flecha amarillas se

cuentan en la ventana de muestreo, el resto no porque golpean con el borde prohibido extendido (en rojo).

Figura 1- Rejilla con ventanas de muestreo [1]

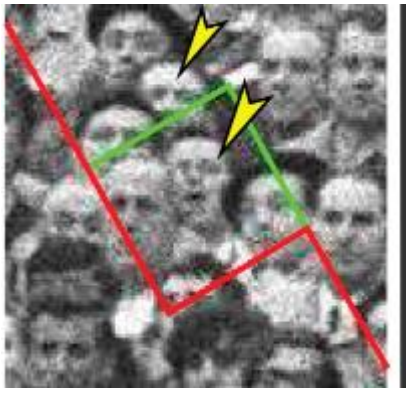

Figura 2 – Ventana de muestreo [1]

1.1.1. Definiciones y notaciones

- N: número de partículas de la población.
- *T (box side)*: separación entre las ventanas de muestreo o quadrats.
    - *Bx*: Anchura de la imagen a procesar
    - *By*: Altura de la imagen a procesar
    - *n0*: Número inicial de quadrats

$$T = \sqrt{\frac{B_x B_y}{n_0}},$$

Ecuación 1

- *t (quadrat side)*: longitud del lado del quadrat.
    - f: fracción de muestreo

$$t = T\sqrt{f}.$$

Ecuación 2

- *Q*: número total de partículas contadas en los quadrats.
- *n*: número de quadrats no vacíos.
- $\widehat{N}$ *(estN)*: nuestro estimador del tamaño de la población.

$$\widehat{N} = \frac{T^2}{t^2} \cdot Q.$$

Ecuación 3

En la imagen siguiente se puede ver que el box side *T*, es el área del cuadro delimitador dividido por el número inicial de quadrats, n0. El tamaño del quadrat, t, es el tamaño del box side multiplicado por la raíz cuadrada de la fracción de muestreo, f.

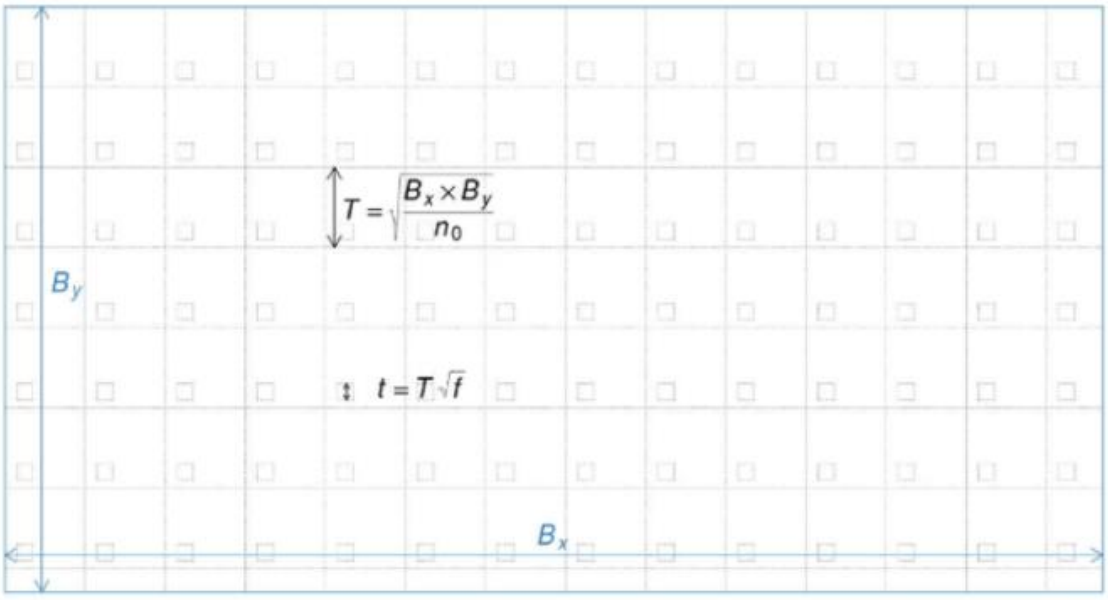

Figura 3 – Box side, quadrat side [2]

1.2. Estimación de la varianza

Para aplicar las siguientes fórmulas hay que pensar que la plantilla con las ventanas de muestreo es una matriz, donde cada ventana de muestreo está situada en una fila – columna y tiene un número de partículas.

1.2.1. Definiciones

- n: Número de columnas
- *Qi*: Número total de partículas capturadas en los quadrats para la columna i.
- *Qoi*: Número total de partículas capturadas en los quadrats impares (filas) para la columna i.
- *Qei*: Número total de partículas capturadas en los quadrats pares (filas) para la columna i.
- t/T $\in (0, 1]$: Fracción de muestreo

$$v_n = \frac{(1-\tau)^2}{3-2\tau} \cdot \sum_{i=1}^{n} (Q_{oi} - Q_{ei})^2.$$

Ecuación 4

$$C_k = \sum_{j=1}^{n-k} Q_j Q_{j+k}, \quad k = 0, 1, 2.$$

Ecuación 5

$$\mathrm{var}_{\mathrm{Cav}}\left(\widehat{N}\right) = \frac{1}{6} \cdot \frac{(1-\tau)^2}{\tau^4(2-\tau)} \cdot [3(C_0 - v_n) - 4C_1 + C_2] + \frac{v_n}{\tau^4},$$

Ecuación 6 – Error de estimacion de la varianza (Varcav(N))

1.2.2. Coeficiente de error de la varianza

El coeficiente de error de la varianza calcula el porcentaje de error de la estimación de la varianza (más fácil de interpretar).

$$Ce = 100 * \sqrt{(varcav/estN)}$$

Ecuación 7

2. Metodología y herramientas

El presente apartado establece la metodología utilizada durante el desarrollo de la aplicación, las fases establecidas y por último las herramientas que se han utilizado para desarrollarlo.

2.1. Metodología

La metodología utilizada durante el desarrollo de este proyecto es iterativa e incremental. Esta metodología consiste en dividir el proyecto en diferentes bloques o iteraciones, donde cada iteración tiene como objetivo incorporar nuevas funcionalidades respecto a iteraciones anteriores, añadiendo nuevos objetivos/requisitos o mejorando los que ya fueron completados.

Para ello, cada requisito se debe completar en una única iteración, en la cual se deben realizar todas las tareas necesarias para completarlo (incluyendo pruebas y documentación).

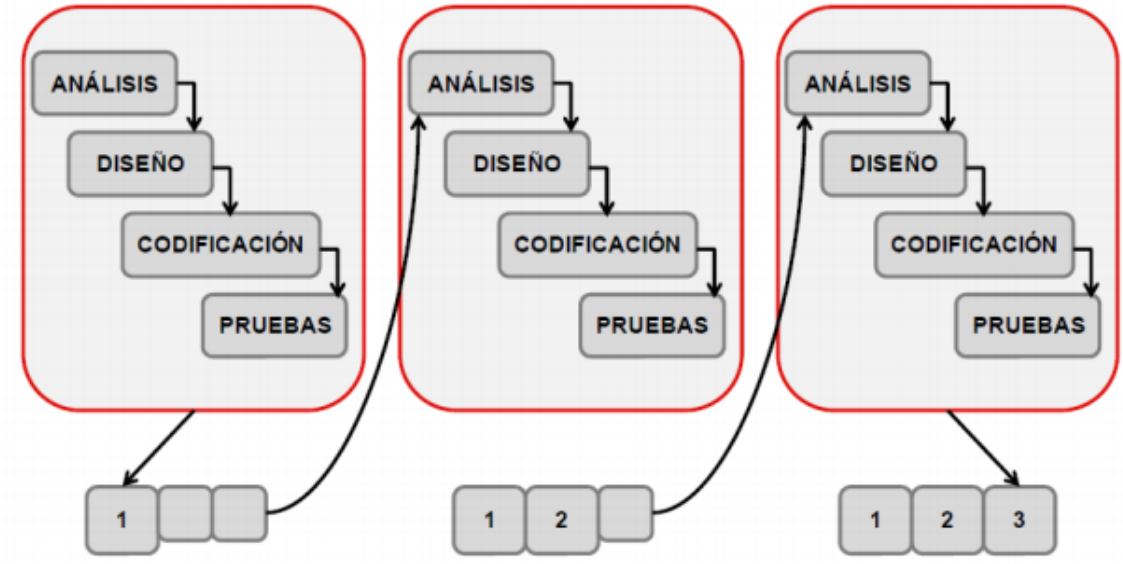

Figura 4 – Metodología iterativa e incremental

A continuación, se muestra el contenido de las iteraciones de las que se ha compuesto el proyecto:

- Iteración 1

    - Diseño y creación de la BBDD
    - Implementación de la capa de persistencia utilizando el framework JPA
    - Implementación de la capa de servicio
    - Definición de los recursos de la capa controlador
    - Implementación de la capa controlador
    - Pruebas unitarias de cada capa
    - Pruebas de integración de las capas

- Iteración 2

    - Diseño de la interfaz del login y la de registrarse
    - Implementación de las interfaces (ver Figura 5 y Figura 6)
    - Interactividad de las interfaces
    - Implementación de las llamadas a la capa controlador para poder loggearse y registrarse en la aplicación
    - Pruebas

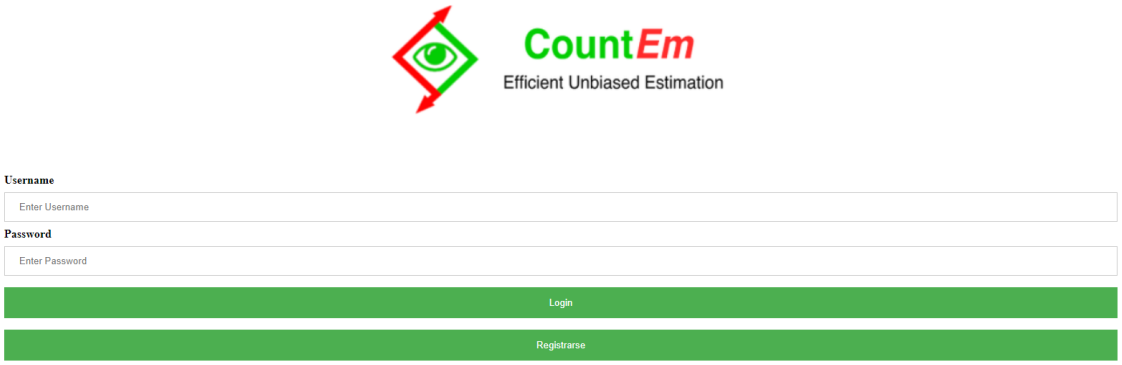

Figura 5 - Interfaz Login

Figura 6 - Interfaz Registrarse

- Iteración 3
    - Diseño de la interfaz 1
    - Implementación interfaz 1 (ver Figura 7)
        - Cargar imagen desde la aplicación
        - Inicialización de los valores por defecto de los campos (Sampling fraction, Initial Number of Quadrats, Rotation angle)
        - Calcular y mostrar el Box Side y Quadrat Side
        - Calcular y mostrar automáticamente el Box Side y Quadrat Side en caso de que el usuario introduzca nuevos valores en los campos Sampling fraction e Initial number of Quadrats
    - Pruebas

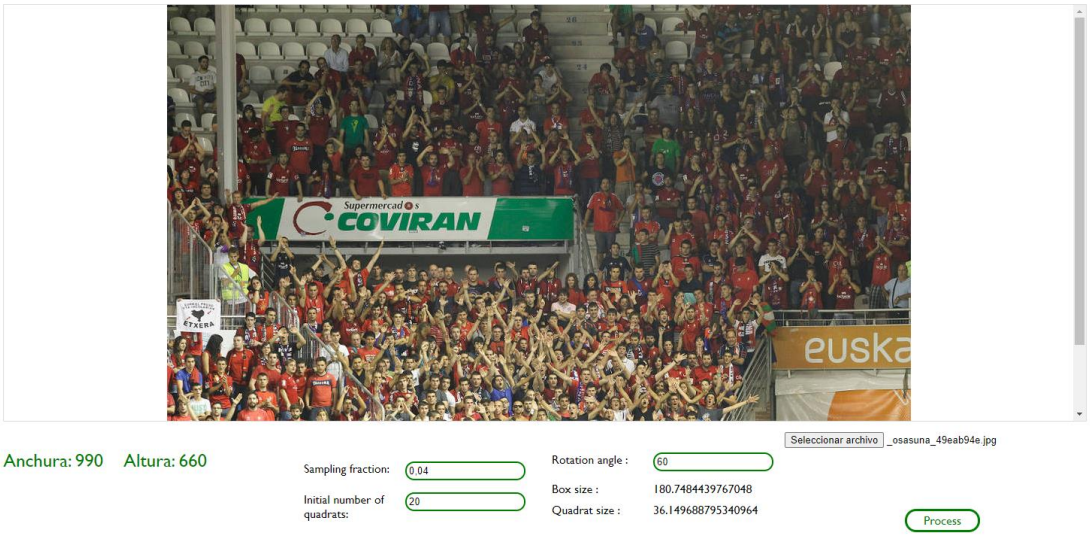

Figura 7 - Interfaz 1

- Iteración 4
    - Diseño de la interfaz 2
    - Implementación botón process
        - Cálculo del número de ventanas de muestreo
        - Quitar elementos htlm de la interfaz 1 y mostrar los de la interfaz 2
    - Implementación interfaz 2 CountEm (ver Figura 8)
        - Botón back para volver a la interfaz 1
        - Pintar ventanas de muestreo
        - Recortar ventanas de muestreo
        - Recorrer diferentes ventanas de muestreo
        - Guardar número de partículas introducidas por el usuario
        - Botón overview para mostrar la imagen con todas las ventanas de muestreo (ver Figura 9)

Figura 8 - Interfaz 2

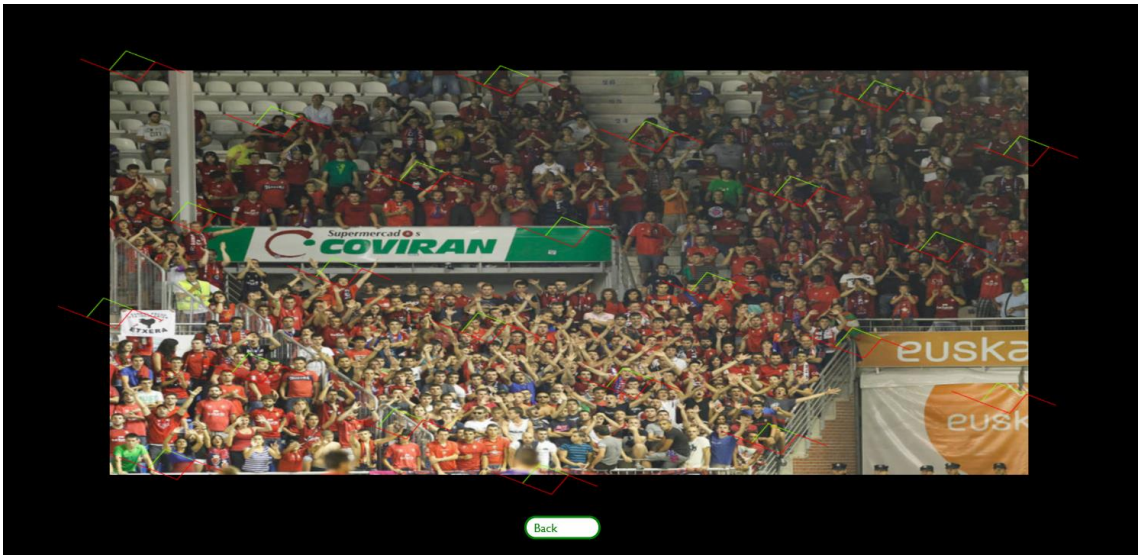

Figura 9 - Overview

- Iteración 5
    - Diseño de la interfaz 3
    - Implementación botón continue
        - Cálculo de los resultados que se van a mostrar en la interfaz 3: Estimated number of visible particles (estN), Number of particles counted in the quadrats, Number of non empty quadrats, Predicted standard error of estN, Predicted coefficient of error of estN (%).
        - Quitar elementos html de la interfaz 2 y mostrar los de la interfaz 3
    - Implementación interfaz 3 (ver Figura 10)
        - Mostrar resultados en la tabla
        - Implementación botón back para volver a la interfaz 2
        - Exportar resultados a formato csv

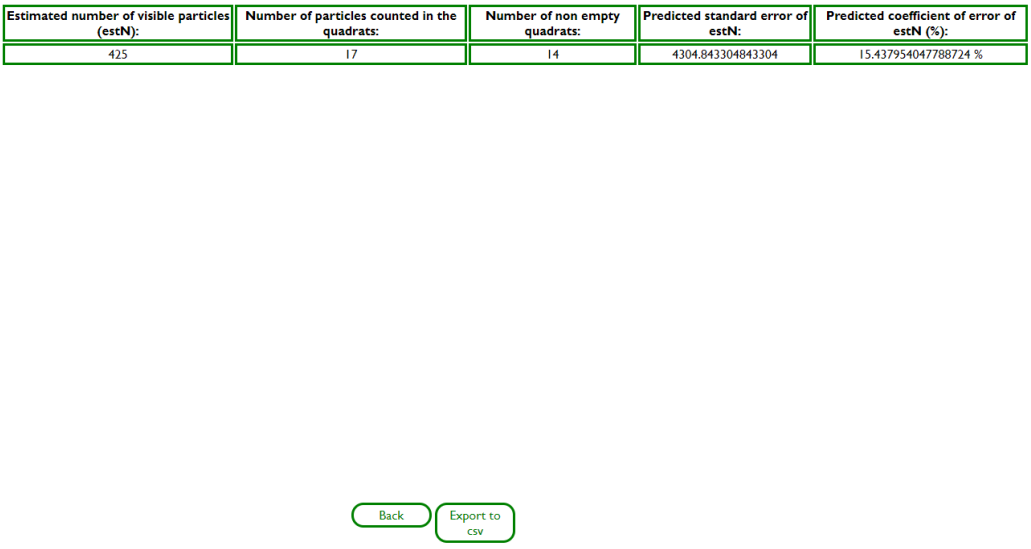

Figura 10 – Interfaz 3 Resultados

2.2. Tecnologías y herramientas

En este apartado se detallan las tecnologías y herramientas utilizadas durante el desarrollo del proyecto.

Tecnologías

- **HTML5**

HTML 5 es la quinta revisión importante del lenguaje básico HTML. HTML5 especifica dos variantes de sintaxis para HTML, una clásica HTML, conocida como HTML5, y una variante XHTML conocida como XHTML 5. Contiene un conjunto más amplio de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance.

- **CSS3**

CSS es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML.

- **JavaScript**

JavaScript es el lenguaje de programación encargado de dotar de mayor interactividad y dinamismo a las páginas web. Cuando JavaScript se ejecuta en el navegador, no necesita de un compilador. El navegador lee directamente el código, sin necesidad de terceros. Por tanto, se le reconoce como uno de los tres lenguajes nativos de la web junto a HTML (contenido y su estructura) y a CSS (diseño del contenido y su estructura).

- **jQuery**

jQuery es una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

- **Ajax**

Ajax es una tecnología asíncrona, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página, aunque existe la posibilidad de configurar las peticiones como síncronas de tal forma que la interactividad de la página se detiene hasta la espera de la respuesta por parte del servidor.

- **Java**

Java es un lenguaje orientado a objetos, independiente de la plataforma hardware donde se desarrolla, y que utiliza una sintaxis similar a la de C++ pero reducida. Java, como lenguaje de programación, ofrece un código robusto, que ofrece un manejo automático de la memoria, lo que reduce el número de errores.

- **Spring Boot Framework**

Es una de las tecnologías de Spring Framework que permite de forma mas sencilla la configuración inicial y preparación de las aplicaciones.

- **Maven**

Maven es una herramienta de software para la gestión y construcción de proyectos Java. Es similar en funcionalidad a Apache Ant, pero tiene un modelo de configuración de construcción más simple, basado en un formato XML.

- **JPA Framework**

Es un framework del lenguaje de programación Java que maneja datos relacionales en aplicaciones usando la Plataforma Java en sus ediciones Standard (Java SE) y Enterprise (Java EE).

Herramientas

- **XAMPP**

XAMPP es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl.

- **STS**

SpringSource Tool Suite (STS) es un IDE basado en la versión Java EE de Eclipse, pero altamente customizado para trabajar con Spring Framework.

- **Visual Studio Code**

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

- **Postman**

Se trata de una herramienta dirigida a desarrolladores web que permite realizar peticiones HTTP a cualquier API. Postman es muy útil a la hora de programar y hacer pruebas, puesto que nos ofrece la posibilidad de comprobar el correcto funcionamiento de nuestros desarrollos.

- **GitHub**

Sistema de control de versiones donde almacenar proyectos y el cual te brinda herramientas muy útiles para el trabajo en equipo.

- **SonarQube**

Sistema para evaluar código fuente, usando diversas herramientas de análisis estático de código fuente para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

3. Especificación de requisitos

En el siguiente apartado se detalla la especificación de requisitos de la aplicación, tanto funcionales, no funcionales y casos de uso.

3.1. Requisitos funcionales

En la siguiente tabla se recogen los requisitos funcionales identificados que el sistema ha de cumplir.

ID	Descripción
RF-001	El sistema debe permitir al usuario loguearse en la aplicación introduciendo los valores en un formulario.
RF-002	El sistema debe permitir al usuario registrarse en la aplicación introduciendo los valores en un formulario.
RF-003	El sistema debe permitir al usuario cargar una imagen desde su propio equipo.
RF-004	El sistema debe ser capaz de mostrar el número de quadrats dependiendo de los datos introducidos por el usuario.
RF-005	El sistema debe permitir al usuario introducir el número de partículas por quadrat.
RF-006	El sistema debe ser capaz de almacenar el número de partículas por quadrat introducidas por el usuario.
RF-007	El sistema debe ser capaz de mostrar los resultados finales en una tabla.

RF-008	El sistema debe permitir al usuario exportar los resultados a formato csv.

Tabla 1 – Requisitos funcionales

3.2. Requisitos no funcionales

En la siguiente tabla se recogen los requisitos no funcionales identificados que el sistema ha de satisfacer.

ID	Descripción
RNF-001	El sistema deberá ser mantenible y adaptable a nuevos requerimientos o funcionalidades.
RNF-002	Las interfaces de usuario deben ser intuitivas y de fácil aprendizaje para los usuarios.
RNF-003	El sistema deberá poder ser accesible desde navegador Chrome y Mozilla Firefox.

Tabla 2 – Requisitos no funcionales

3.3. Casos de uso

A continuación, se muestran los casos de uso que serán implementados en el sistema.

Id + Nombre	CU – 001 - Login
Autor	Andrés Barrado Martín
Actor Principal	Usuario
Descripción	Tras abrir la página en el navegador el usuario desea loguearse en la aplicación para hacer uso de esta.
Precondición	
Garantías si éxito	El usuario puede hacer uso de la aplicación.
Garantías mínimas	La aplicación se mantiene estable.
Escenario principal	1. El sistema muestra al usuario el formulario de 'Login' (ver Figura 5). 2. El usuario introduce el nombre de usuario y contraseña. 3. El usuario pulsa el botón 'Login' 4. El sistema muestra la interfaz 1 de la aplicación (ver Figura 7).
Extensiones	3a. El usuario o contraseña no son válidos. a.1 El sistema notifica al usuario que el usuario y contraseña no son válidos. 3b. Error interno de la aplicación. b.1 El sistema notifica al usuario que lo intente más tarde.
Comentarios	

Tabla 3 – Caso de uso: Login

Id + Nombre	CU – 002 – Registrarse
Autor	Andrés Barrado Martín
Actor Principal	Usuario
Descripción	El usuario solicita registrarse en la aplicación para hacer uso de esta.
Precondición	
Garantías si éxito	El sistema muestra al usuario la página de login una vez registrado en la aplicación
Garantías mínimas	La aplicación se mantiene estable.
Escenario principal	1. El sistema muestra al usuario el formulario de 'Login' (ver Figura 5). 2. El usuario pulsa el botón 'Registrarse' 3. El sistema muestra al usuario el formulario de 'Registrarse' (ver Figura 6). 4. El usuario rellena el formulario 5. El usuario pulsa el botón 'Aceptar' 6. El sistema valida los datos 7. El sistema almacena los datos en la base de datos 8. El sistema regresa a la página de login
Extensiones	6a. El campo 'Username' va vacío a.1 El sistema notifica al usuario que este campo no puede ir vacío a.2 Se vuelve al paso 4 6b. El nombre de usuario ya existe b.1 El sistema notifica al usuario que el nombre de usuario introducido ya existe b.2 Se vuelve al paso 4 6c. El campo 'Nombre Completo' va vacío c.1 El sistema notifica al usuario que este campo no puede ir vacío c.2 Se vuelve al paso 4 6d. El campo 'Apellidos' va vacío d.1 El sistema notifica al usuario que este campo no puede ir vacío d.2 Se vuelve al paso 4 6e. El campo 'Email' va vacío e.1 El sistema notifica al usuario que este campo no puede ir vacío e.2 Se vuelve al paso 4 6f. El campo 'Email' no tiene el formato correcto e.1 El sistema notifica al usuario que este campo no tiene el formato correcto e.2 Se vuelve al paso 4 6g. El campo 'Password' va vacío e.1 El sistema notifica al usuario que este campo no puede ir vacío e.2 Se vuelve al paso 4 6h. Los valores del campo 'Password' y 'Repeat Password' no son iguales e.1 El sistema notifica al usuario que las contraseñas no coinciden. e.2 Se vuelve al paso 4

	7a. Error interno de la aplicación a.1 El sistema notifica al usuario que hubo un error con su petición de registrarse en la aplicación, que lo intente más tarde. a.2 Se vuelve al paso 4.
Comentarios	

Tabla 4. Caso de uso: Registrarse

Id + Nombre	CU – 003 – Exportar resultados
Autor	Andrés Barrado Martín
Actor Principal	Usuario
Descripción	El usuario exporta los resultados obtenidos de la aplicación
Precondición	El usuario ha tenido que loguearse en la aplicación.
Garantías si éxito	El sistema dejará en la carpeta Descargas del equipo del usuario los resultados obtenidos en formato csv.
Garantías mínimas	La aplicación se mantiene estable.
Escenario principal	1. El sistema muestra al usuario la interfaz 1 2. El usuario pulsa el botón 'Seleccionar archivo' 3. Se abre el explorador de windows 4. El usuario selecciona la imagen que quiere subir 5. El sistema carga la imagen y rellena los campos por defecto 6. El usuario rellena los campos del formulario en caso de que quiera cambiar alguno 7. El usuario pulsa el botón 'Process' 8. El sistema abre la interfaz 2 y muestra la imagen procesada 9. El usuario introduce el número de partículas por quadrat 10. El usuario pulsa el botón 'Continue' 11. El sistema muestra la interfaz 3 con la tabla resultados 12. El usuario pulsa el botón 'Export to csv' 13. El sistema guarda los resultados en la carpeta Descargas del usuario
Extensiones	5a. La imagen cargada no tiene el formato correcto a.1 El sistema notifica al usuario que la imagen cargada no cumple los requisitos a.2 Se vuelve al paso 4 7a. El usuario no ha cargado ninguna imagen a.1 El sistema notifica al usuario que debe cargar una imagen a.2 Se vuelve al paso 4
Comentarios	

Tabla 5. Caso de uso: Exportar resultados

4. Diseño e implementación

Una vez establecidos los requisitos funcionales y no funcionales que ha de cumplir el sistema, se procede a exponer el diseño e implementación de cada una de las capas que componen el sistema. Cabe destacar que la aplicación está dividida en Front-end donde estará la capa de presentación y el Back-end donde estarán alojadas el resto de las capas.

4.1. Diseño Arquitectónico

En este apartado se presenta el diseño arquitectónico del sistema, para facilitar la compresión de la aplicación y así favorecer la mantenibilidad.

El sistema se ha diseñado utilizando una arquitectura en N capas (ver figura 11). En este caso, el sistema dispondrá de 4 capas, siendo estas las capas de:

- Presentación: Capa que presenta las distintas interfaces con las que el usuario interactúa con la aplicación.
- Controlador: Capa encargada de exponer los diferentes recursos que conforman el microservicio, como el modo de poder acceder a ellos definiendo sus identificadores.
- Servicio: Capa intermedia entre la capa controlador y la capa de persistencia, que implementa la lógica de negocio.
- Persistencia: Es la capa encargada de obtener y transferir los datos. La recuperación y persistencia de estos datos se realizará mediante la utilización del framework JPA.

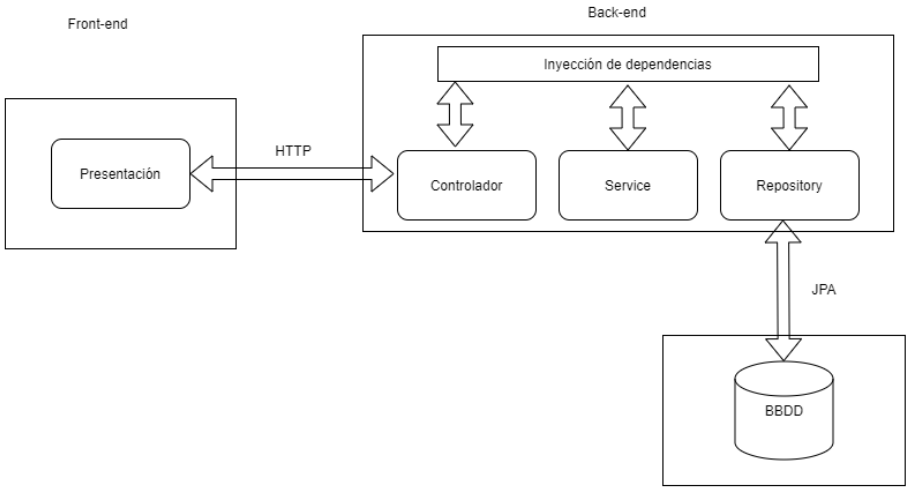

Figura 11 - Diagrama de arquitectura

Componentes del sistema

Como se muestra en la Figura 12 la parte del back de la aplicación solo consta de un componente por capa.

Figura 12 – Diagrama de componentes

Diseño de la capa controlador

En la tabla que viene a continuación se muestra la definición de los recursos, en este caso del recurso Usuario.

Recurso	URI	Método	Parámetro
Usuario	/user/authorization/{user}/{password}	GET	user: String password: String
Usuario	/user/signIn	POST	User : JSON Object

Tabla 6 – Definición recursos

Método	Código	Comentario
GET	200 OK	Usuario autorizado / no autorizado, depende del body de la respuesta
	500 INTERNAL SERVER ERROR	Error del sistema
POST	200 OK	Usuario ya registrado
	201 CREATED	Usuario registrado correctamente
	500 INTERNAL SERVER ERROR	Error del sistema

Tabla 7 – Definición respuesta recursos

4.2. Implementación

Tanto para la capa controlador, servicio y acceso a datos que irán en el mismo servidor se ha hecho el uso de Spring Boot Framework [8].

Spring define un conjunto de anotaciones [3] que garantizan cada uno de los componentes asociándoles una responsabilidad concreta.

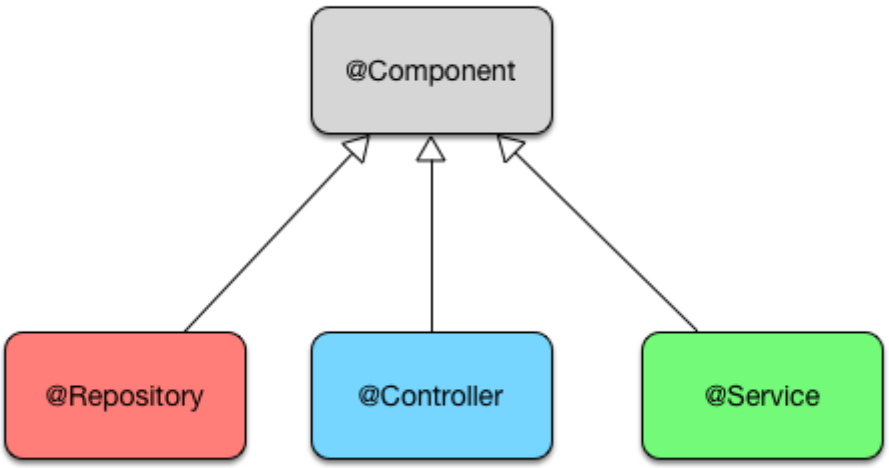

Figura 13 – Spring Stereotypes

@Component: Es el estereotipo general y permite anotar un bean para que Spring lo considere uno de sus objetos.

@Repository: Es el estereotipo que se encarga de dar de alta un bean para que implemente el patrón repositorio que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. Es la anotación utilizada en los componentes de la capa de persistencia.

@Service: Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Es la anotación utilizada en los componentes de la capa de servicio.

@Controller: El último de los estereotipos que es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y el aplicativo. En este caso se ha hecho uso de la anotación @RestController que hereda de la anotación @Controller.

Para la inyección de dependencias de los diferentes componentes del sistema se ha hecho uso de la anotación @Autowired.

```
@RestController
public class UserController {

    @Autowired
    private UserService userService;
```

Figura 14 – Inyección del componente UserService

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;
```

Figura 15 - Inyección del componente UserRepository

4.2.1. Implementación de la capa de persistencia

Para este proyecto se ha hecho uso de MySQL como sistema de gestión de bases de datos. Para utilizarlo en este proyecto hemos usado la herramienta XAMPP [12] para poder crear de forma local una base de datos y la tabla usuarios donde se recogerán y almacenarán sus datos.

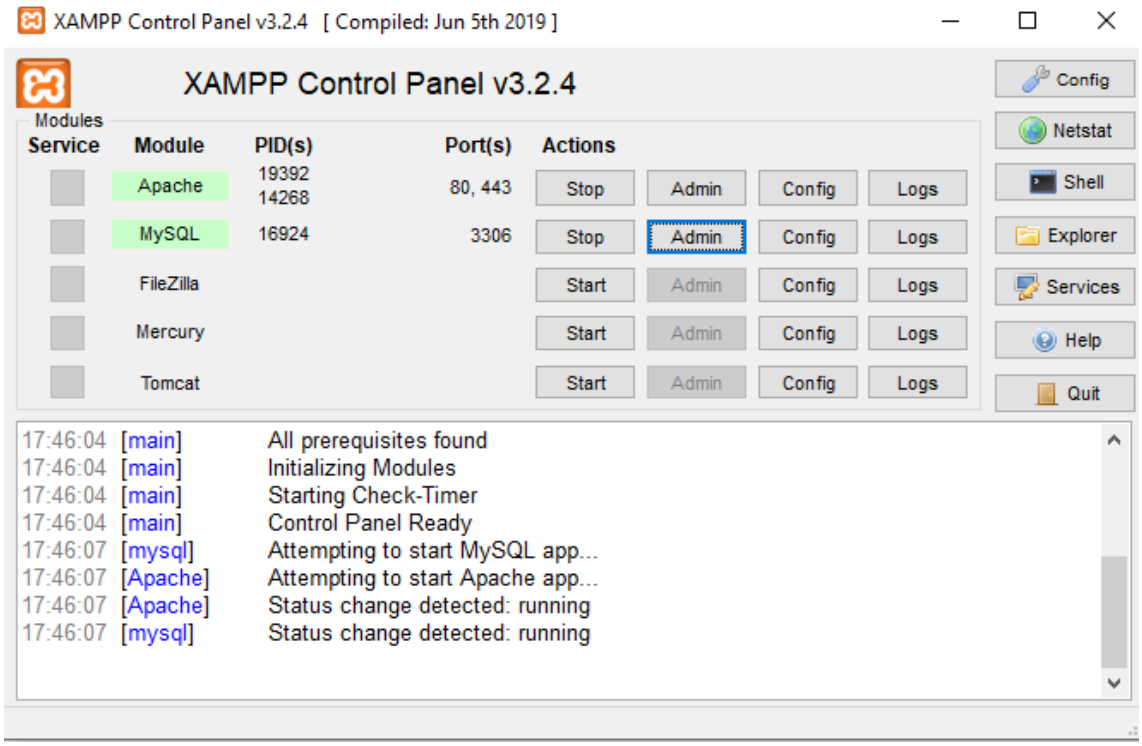

Figura 16 – XAMPP Control Panel

Para la persistencia de los datos con MySQL se ha utilizado el framework JPA [7]. En las figuras que se muestran a continuación viene la configuración y dependencias necesarias para la conexión y persistencia a la base de datos.

```
3 spring.datasource.url=jdbc:mysql://localhost:3306/countem
4 spring.datasource.username=root
5 spring.datasource.password=
```

Figura 17 – Variables por defecto en el fichero.properties para la conexión con la BBDD.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

Figura 18 – Dependencias dentro del fichero pom.xml

Una de las grandes ventajas de JPA es que nos permite manipular la base de datos a través de objetos, para que JPA sea capaz de identificar estos objetos hacemos uso de la anotación @Entity, en la figura que se muestra a continuación se puede ver a modo de ejemplo:

```
@Entity
@Table(name = "user")
public class User {


    @Id
    private String user;
    private String password;
    private String nombre;
    private String apellidos;
    private String email;
```

Figura 19 – Anotaciones para la persistencia de la clase Usuario

4.2.2. Implementación de la capa de servicio

Como se explicó anteriormente esta es la capa que se encarga de gestionar las operaciones de negocio como la lectura y registro de usuarios, llamando a las operaciones del componente que se encuentra en la capa de persistencia. Esta capa solo consta del componente UserService el cual lleva la anotación @Service y depende del componente UserRepository el cual lleva la anotación @Autowired para la inyección de dependencias (Figura 15).

4.2.3. Implementación de la capa controlador

Este componente se encarga de exponer los diferentes recursos los cuales van a ser llamados por la capa de presentación.

Anotaciones utilizadas para implementación de los recursos:

- @GetMapping - Anotación utilizada para recibir peticiones HTTP GET.
- @PostMapping - Anotación utilizada para recibir peticiones HTTP POST.
- @RequestParam - Anotación que obtiene parámetros en la URI.
- @RequestBody - Anotación que mapea el cuerpo de la petición HTTP.

```
@GetMapping(value= "/authorization" )
public ResponseEntity<Message> authorization(@RequestParam String user,@RequestParam String password) {
```

Figura 20 – Implementación método authorization

```
@PostMapping(value= "/signIn" ,consumes="application/json")
public ResponseEntity<Message> signIn( @RequestBody User user){
```

Figura 21 – Implementación método signIn

4.2.4. Implementación capa de presentación

La capa de presentación consta de las diferentes interfaces con las que el usuario interactua con la aplicación. Las interfaces de usuario están basadas en HTML [11] y el estilo que usan es CSS [10], toda la funcionalidad de las interfaces está implementada en JavaScript y jQuery [5].

Para que una página HTML haga uso de los estilos y las diferentes funcionalidades hay que importarlas, los ficheros css van importados dentro de la etiqueta <head></head> y los ficheros js van importados dentro de la etiqueta <body></body>.

```
<head>
    <title>CountEm</title>
    <link rel="stylesheet" href="css/login.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
</head>
```

Figura 22 – Importación ficheros .css

```
<script type="text/javascript" src= "js/login.js"></script>

</body>
```

Figura 23 – Importación ficheros .js

Para implementar los botones se ha hecho uso de la función propia de jQuery click(), está función a parte de ejecutarse para botones también sirve para cualquier elemento del DOM.

```
$('#button-back').click(function(){
    document.getElementById('miniaturas').innerHTML='';
    $("#p1").css({'display':'block'});
    $("#p2").css({'display':'none'});
    $("#miniaturas").css({'display':'none'});
    $("#process").css({'display':'block'});
});
```

Figura 24 – Implementación botón 'Back'

Para detectar los cambios introducidos por los usuarios y recalcular todos los valores necesarios para que la página funcione se ha hecho uso del método addEventListener(), este método nos sirve para registrar un evento a un objecto específico.

A modo de ejemplo, la página muestra ciertos valores por defecto cuando se introduce una imagen, estos valores pueden ser cambiados (rotation angle, number of quadrats …) pero la página debe calcular nuevos campos (box side, quadrat side …) a partir de estos valores, para ello hemos usado el método anterior con el evento input, que cada vez que un usuario introduce un valor dentro de dicho campo se ejecuta el evento.

```
document.getElementById('nQuadrats').addEventListener('input',function(e) {
    var bS=Math.sqrt(($("#image").width()*$("#image").height())/this.value);
    $("#boxSize").val(bS);
    var qS=boxSize*Math.sqrt(($("#samplingFraction").val()));
    $("#quadratSize").val(qS);


});
```

Figura 25 – Event Listener number of quadrats

La gran parte implementada de la página hace uso de procesamiento de imágenes, para ello hemos hecho uso del elemento canvas [9] (<canvas></canvas>), hay ciertos navegadores que no soportan este elemento. En nuestro caso se ha hecho uso para copiar, pegar y recortar imágenes, y también para dibujar líneas cambiando el color y la anchura.

```
document.getElementById('overview2').innerHTML='';
var newListItem= "<canvas id='canvas-overview' width='"+widthC+"px' height='"+heightC+"px'> </canvas>";
$('#overview2').append(newListItem);
var canvas3=document.getElementById("canvas-overview");
var ctx3=canvas3.getContext("2d");
var img3 = canvas2;
ctx3.drawImage(img3,0,0,canvas2.width,canvas2.height,0,0,canvas3.width,canvas3.height);
```

Figura 26 – Implementación imagen overview

Para la comunicación entre el Front y el Back hemos hecho uso de Ajax con jquery, el método utilizado es $.ajax() [4]. Este método cuenta con dos sintaxis posibles:

En la primera sintaxis, se especifica la url a la que enviar la petición Ajax, y luego se pasa el objeto configurable, y en la segunda, se pasa directamente el objeto, que también contendrá la dirección url de la petición. En nuestro caso hemos hecho uso de la segunda sintaxis.

El objeto configurable contendrá uno o varios de los parámetros siguientes:

- type: tipo de petición, GET o POST.
- url: dirección a la que se envía la petición.
- data: datos a enviar al servidor.
- dataType: tipos de datos que esperas recibir del servidor
- done: función que se ejecuta cuando se obtiene una respuesta con éxito.
- error: función que se ejecuta si la petición no tiene éxito

En la imagen siguiente se muestra la implementación de la llamada Ajax para registrar un usuario. Para todas las llamadas se utiliza la opción async: false para que se espere la respuesta por parte del servidor.

```
var data = {
  "user" : $("#user2").val(),
  "password" : $("#password2").val(),
  "nombre":$("#nombre").val(),
  "apellidos":$("#apellidos").val(),
  "email":$("#email").val()
}
$.ajax({
  url: baseUri+'/signIn',
  type: 'POST',
  data: JSON.stringify(data),
  contentType: "application/json",
  async: false,

}).done(function(response,textStatus,xhr){
  console.log(xhr.status);
  console.log(response);
  if(xhr.status == 201){
    $('#response2').html(response.message);
    $("#user2").val("");
    $("#nombre").val("");
    $("#apellidos").val("");
    $("#email").val("");
    $("#password2").val("");
    $("#password3").val("");

    $("#container2").css({'display':'none'});
    $("#container1").css({'display':'block'});
  } else if (xhr.status == 200){
    $('#response2').html(response.message);
  }

}).error(function(response,textStatus,xhr){
  console.log(xhr.status);
  $('#response2').html("Error - try it later");
});
```

Figura 27 – Implementación ajax signIn

Para guardar los datos introducidos por el usuario se ha utilizado la propiedad localStorage [13], la cual te permite almacenar datos en las diferentes sesiones de navegación.

5. Pruebas

En este apartado se detallan las pruebas que se han llevado a cabo durante el desarrollo del proyecto.

5.1. Pruebas Unitarias

Las pruebas unitarias consisten en probar los diferentes módulos por separado y que su funcionamiento sea el correcto. Para ello se ha hecho uso de la librería JUnit y librerías propias de Spring.

5.1.1. Capa Persistencia

Casos de prueba

Caso: Alta Usuario

a) Alta válida (usuario almacenado en BBDD)

Caso	UserName	Password	Nombre	Apellidos	Email	Resultado
Alta Usuario	test	test	testName	testApe	test@gmail.com	User u

Tabla 8 – Casos de prueba capa persistencia

Implementación

```
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace=Replace.NONE)
public class UserRepositoryTest {
    @Autowired
    TestEntityManager entityManager;

    @Autowired
    private UserRepository userRepository;

    @Test
    public void saveUser() {
        User user = new User("test","test","testName","testApe","test@gmail.com");
        user = entityManager.persistAndFlush(user);
        assertThat(userRepository.findById(user.getUser()).get()).isEqualTo(user);
    }

}
```

Figura 28 – Implementación pruebas unitarias capa persistencia

5.1.2 Capa Servicio

Casos de prueba

Caso: Alta Usuario

a) Alta válida (nuevo Usuario)

b) Alta no válida (nuevo Usuario ya existente)

Caso	UserName	Password	Nombre	Apellidos	Email	Resultado

Alta Usuario	test	test	testName	testApe	test@gmail.com	true
Alta Usuario	admin	admin	admin	admin	admin@gmail.com	false

Tabla 9 – Casos de prueba capa servicio

Implementación

```
@RunWith(MockitoJUnitRunner.class)
public class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserService userService;

    @Before
    public void init() {
        when(userRepository.findById("test")).thenReturn(Optional.empty());
        User user = new User("admin","admin","admin","admin","admin@gmail.com");
        when(userRepository.findById("admin")).thenReturn(Optional.of(user));
    }
    @Test
    public void saveUser() {
        User user = new User("test","test","testName","testApellidos","test@gmail.com");

        boolean created = userService.save(user);

        assertTrue(created);
    }


    @Test
    public void notSaveUser() {
        User user = new User("admin","admin","admin","admin","admin@gmail.com");


        boolean created = userService.save(user);

        assertTrue(!created);
    }
}
```

Figura 29 – Implementación pruebas unitarias capa servicio

5.1.2. Capa Controlador

Casos de prueba

Caso: Login

a) Usuario autorizado
b) Usuario no autorizado

Caso: Registrarse

a) Usuario registrado con éxito
b) Usuario ya existente

Caso	UserName	Password	Resultado
Login	admin	admin	Authorized
Login	test	test	Unathorized

Tabla 10 – Casos de prueba Login capa controlador

Caso	UserName	Password	Nombre	Apellidos	Email	Resultado
Registrarse	test	test	testName	testApe	test@gmail.com	Usuario creado con éxito
Registrarse	admin	admin	admin	admin	admin@gmail.com	Usuario ya existe

Tabla 11 – Casos de prueba Registrarse capa controlador

Implementación

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMockMvc
public class UserControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    ObjectMapper mapper = new ObjectMapper();
    // Test que verifica si el usuario tiene acceso
    @Before
    public void init() {
        when(userService.authorized("admin", "admin")).thenReturn(true);
        when(userService.authorized("test", "test")).thenReturn(false);
        User user = new User("admin","admin","admin","admin","admin@gmail.com");
        when(userService.save(user)).thenReturn(false);
        User user2 = new User("test","test","testName","testApellidos","test@gmail.com");
        when(userService.save(user2)).thenReturn(true);

    }

    @Test
    public void authorized() throws Exception {

        ResultActions resultActions = mockMvc
                .perform(get("/authorization").param("user", "admin").param("password", "admin"))
                .andExpect(status().isOk());

        MvcResult result = resultActions.andReturn();
        String content = result.getResponse().getContentAsString();

        Gson g = new Gson();
        Message m = g.fromJson(content, Message.class);
        assertTrue(m.getMessage().equals("authorized"));
    }
```

Figura 30 – Implementación pruebas unitarias capa controlador (1)

```java
// Test que verifica si el usuario no tiene acceso
@Test
public void unauthorized() throws Exception {

    ResultActions resultActions = mockMvc
            .perform(get("/authorization").param("user", "test").param("password", "test"))
            .andExpect(status().isOk());

    MvcResult result = resultActions.andReturn();
    String content = result.getResponse().getContentAsString();

    Gson g = new Gson();
    Message m = g.fromJson(content, Message.class);
    assertTrue(m.getMessage().equals("unauthorized"));

}

// Test que verifica usuario registrado con exito
@Test
public void signIn() throws Exception {

    User user = new User("test","test","testName","testApellidos","test@gmail.com");

    ResultActions resultActions = mockMvc
            .perform(post("/signIn")
            .contentType(MediaType.APPLICATION_JSON)
            .content(mapper.writeValueAsString(user)))
            .andExpect(status().isCreated());

    MvcResult result = resultActions.andReturn();
    String content = result.getResponse().getContentAsString();

    Gson g = new Gson();
    Message m = g.fromJson(content, Message.class);
    assertTrue(m.getMessage().equals("Usuario registrado con exito"));

}
```

Figura 31 – Implementación pruebas unitarias capa controlador (2)

```java
@Test
public void notSignIn() throws Exception {

    User user = new User("admin","admin","admin","admin","admin@gmail.com");

    ResultActions resultActions = mockMvc
            .perform(post("/signIn")
            .contentType(MediaType.APPLICATION_JSON)
            .content(mapper.writeValueAsString(user)))
            .andExpect(status().isOk());

    MvcResult result = resultActions.andReturn();
    String content = result.getResponse().getContentAsString();

    Gson g = new Gson();
    Message m = g.fromJson(content, Message.class);
    assertTrue(m.getMessage().equals("Este usuario ya existe"));


}
```

Figura 32 – Implementación pruebas unitarias capa controlador (3)

5.2. Pruebas de integración

Las pruebas de integración consisten en probar el funcionamiento correcto de los diferentes módulos conjuntamente.

Lo que se ha querido verificar con estas pruebas es que la comunicación entre el Front y el Back funciona correctamente y para ello se ha hecho uso de la herramienta Selenium.
A continuación, se detallan las pruebas realizadas:

- Login: Se envía un user password incorrecto y se verifica que se muestra el mensaje correspondiente en la pantalla.
- Registrarse: Se envía un usuario ya existente y se verifica que se muestra el mensaje correspondiente en la pantalla.

Se han implementado estas pruebas de esta manera para no persistir nada en BBDD.

Implementación

```
@RunWith(SpringRunner.class)
public class CapaPresentacionTest {

    @Test
    public void notLogin() throws InterruptedException {
        WebDriver driver = null;
        try {
            System.setProperty("webdriver.chrome.driver","D:\\Descargas\\chromedriver\\chromedriver.exe");
            driver = new ChromeDriver();
            driver.get("http://localhost/TFGFinal/");
            Thread.sleep(5000);  // Let the user actually see something!
            WebElement userName = driver.findElement(By.id("user"));
            userName.sendKeys("test");
            Thread.sleep(2000);
            WebElement password = driver.findElement(By.id("password"));
            password.sendKeys("test");
            Thread.sleep(2000);
            WebElement btnLogin = driver.findElement(By.id("login"));
            btnLogin.click();
            Thread.sleep(2000);

            WebElement response = driver.findElement(By.id("response"));
            String re = response.getText();
            assertTrue(re.equals("Usuario - password no valido"));


        }catch(Exception ex) {
            ex.printStackTrace();
        }finally {
            driver.quit();
        }


    }
```

Figura 33 – Implementación pruebas de integración (1)

```java
@Test
public void notSignIn() throws InterruptedException {
    WebDriver driver = null;
    try {
        System.setProperty("webdriver.chrome.driver","D:\\Descargas\\chromedriver\\chromedriver.exe");
        driver = new ChromeDriver();
        driver.get("http://localhost/TFGFinal/");
        Thread.sleep(5000);  // Let the user actually see something!

        WebElement btnLogin = driver.findElement(By.id("registrarse"));
        btnLogin.click();
        Thread.sleep(2000);

        WebElement userName = driver.findElement(By.id("user2"));
        userName.sendKeys("admin");
        Thread.sleep(2000);

        WebElement nombre = driver.findElement(By.id("nombre"));
        nombre.sendKeys("admin");
        Thread.sleep(2000);

        WebElement apellidos = driver.findElement(By.id("apellidos"));
        apellidos.sendKeys("admin admin");
        Thread.sleep(2000);

        WebElement email = driver.findElement(By.id("email"));
        email.sendKeys("admin@gmail.com");
        Thread.sleep(2000);

        WebElement pass = driver.findElement(By.id("password2"));
        pass.sendKeys("pass");
        Thread.sleep(2000);

        WebElement pass2 = driver.findElement(By.id("password3"));
        pass2.sendKeys("pass");
        Thread.sleep(2000);

        WebElement btnAceptar = driver.findElement(By.id("aceptar"));
        btnAceptar.click();
        Thread.sleep(2000);

        WebElement response = driver.findElement(By.id("response2"));
        String re = response.getText();
        assertTrue(re.equals("Este usuario ya existe"));


    }catch(Exception ex) {
        ex.printStackTrace();
    }finally {
        driver.quit();
    }
```

Figura 34 – Implementación pruebas de integración (2)

5.3. Pruebas de aceptación

Las pruebas de aceptación miden el nivel de satisfacción del cliente con el sistema, ajustándose a lo requerido y establecido.

Como se explicó anteriormente, en cada iteración se hace una entrega parcial del producto al cliente el cual comprueba que el sistema se ajusta a lo requerido.

6. Análisis de calidad

Para analizar el código implementado se ha hecho uso de la herramienta SonarQube [11] para corregir errores y mejorar el sistema. La ejecución del análisis nos deja una serie de métricas que nos indican el estado en el que se encuentra el proyecto. En la figura 36, se muestran los resultados obtenidos en el primer análisis.

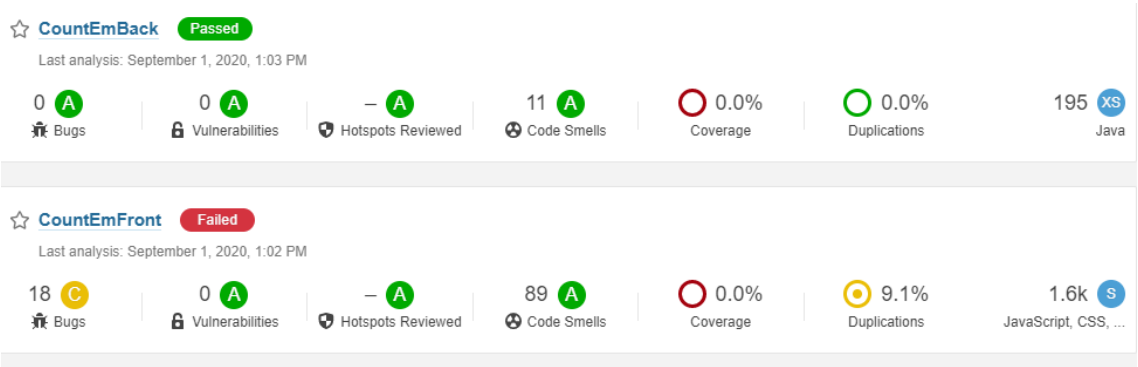

Figura 35 – Primer análisis SonarQube

Como se puede observar la parte del Back pasa perfectamente las métricas de calidad, pero la parte del Front no, por lo que se procede a solucionar todos los Bugs encontrados.

Tras realizar un segundo análisis aplicando las correcciones propuestas por Sonar se puede observar en la Figura 37 que se han conseguido eliminar todos los Bugs.

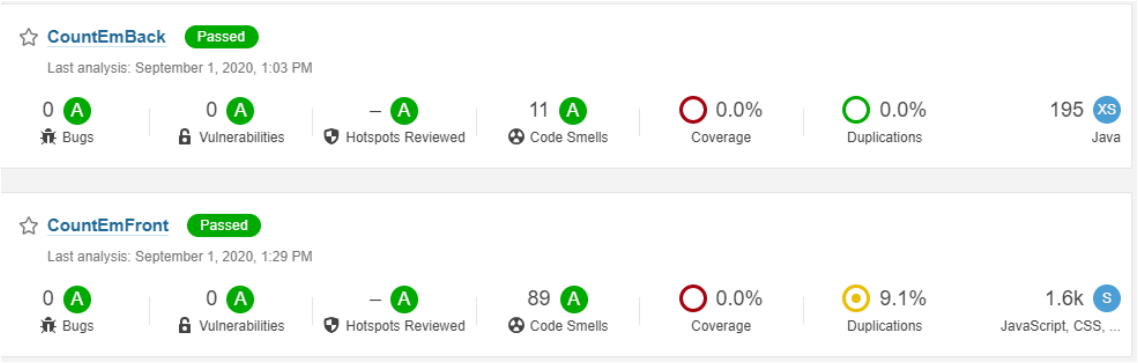

Figura 36 – Segundo análisis SonarQube

En el Front, como se puede observar en la imagen siguiente hay una deuda técnica de 3 días por lo que se procede a reducir el número de 'Code Smells' y así reducirla.

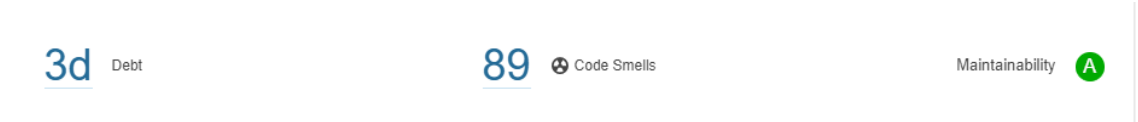

Figura 37 – Code Smells y deuda técnica

En la imagen siguiente se muestra que se ha conseguido reducir la deuda técnica en un día, no se han procedido a quitar el resto de 'Code Smells' ya que trata de variables definidas en los bucles 'var i=0' repetidamente y las detecta como 'Code Smells'.

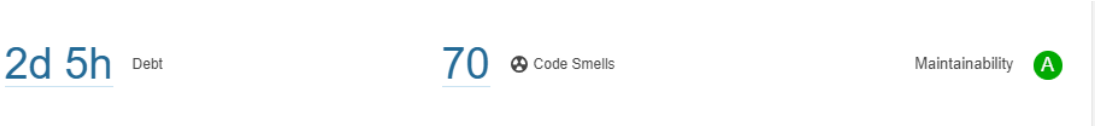

Figura 38 – Reducción Code Smells y deuda técnica

7. Conclusiones y trabajos futuros

En este apartado se recogen las conclusiones obtenidas durante el desarrollo del proyecto, así como las futuras funcionalidades que se lo podrían añadir al mismo.

7.1. Conclusiones

El presente proyecto tiene como finalidad la elaboración de una aplicación web para la estimación de partículas de una imagen con un porcentaje de error pequeño.

El proyecto ha cumplido con los requisitos especificados, teniendo al final una versión funcional en un entorno local.

En el ámbito personal, este proyecto me ha hecho aprender nuevos lenguajes, como pueden ser HTML, CSS, JavaScript y jQuery los cuales son muy demandados en las empresas y en un futuro me pueden venir muy bien. También me ha servido para tener una idea base de métodos de estimación los cuales me han parecido muy interesantes.

7.2. Trabajos futuros

Una vez cumplidos los requisitos establecidos, el sistema desarrollado puede ser extendido con nuevas funcionalidades. A continuación, se detallan brevemente estas:

1. Imagen de carga del procesamiento de la imagen.
2. Aumentar las diferentes ventanas de muestreo
3. Adaptación de la base de datos y guardar los conteos realizados por los usuarios.
4. Mostrar un histórico de los resultados obtenidos por el usuario y poder descargarlos.
5. Pequeñas mejoras visuales de las interfaces.
6. Guardar sesión Login
7. Aplicar seguridad back de la aplicación

Bibliografía

[1] Cruz, M., & Gómez, D. (s.f.). *Efficient and Unbiased Estimation of Population Size*. Obtenido de https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0141868#pone-0141868-t001

[2] Cruz, M., & González-Villa, J. (s.f.). *Simplified procedure for efficient and unbiased population size estimation*. Obtenido de https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0206091

[3] Álvarez, C. (s.f.). *Spring Stereotypes y anotaciones*. Obtenido de https://www.arquitecturajava.com/spring-stereotypes/

[4] jQuery. (s.f.). *jQuery Ajax*. Obtenido de https://api.jquery.com/jquery.ajax/

[5] jQuery. (s.f.). *jQuery API*. Obtenido de https://api.jquery.com/

[6] sonarQube. (s.f.). *sonarQube*. Obtenido de https://www.sonarqube.org/

[7] Spring. (s.f.). *Accessing Data with JPA*. Obtenido de https://spring.io/guides/gs/accessing-data-jpa/

[8] Spring. (s.f.). *Spring Boot*. Obtenido de https://spring.io/projects/spring-boot

[9] w3.unpo<code>todo. (s.f.). *Canvas - la chuleta*. Obtenido de http://w3.unpocodetodo.info/canvas/chuleta.php

[10] w3schools. (s.f.). *CSS*. Obtenido de https://www.w3schools.com/css/

[11] w3schools. (s.f.). *HTML*. Obtenido de https://www.w3schools.com/html/

[12] WIKIPEDIA La enciclopedia libre. (s.f.). *XAMPP*. Obtenido de https://es.wikipedia.org/wiki/XAMPP

[13] MDN. (s.f.). *LocalStorage*. Obtenido de https://developer.mozilla.org/es/docs/Web/API/Storage/LocalStorage