ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADO EN INGENIERÍA DE SOFTWARE

Herramienta web para la extracción y procesamiento de información a partir de ficheros CSV

Parte 1

Web Tool for data extraction and processing from CSV files

Part 1

Realizado por
Ricardo Peralta Vilaseca

Tutorizado por
Eduardo Guzmán de los Riscos

Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2019

Fecha defensa:

Fdo. El/la Secretario/a del Tribunal

Resumen

En este trabajo de fin de grado se presenta una aplicación que será capaz de realizar acciones de lectura, modificación y análisis de datos contenidos en ficheros de formato excel o csv que el usuario quiera tratar.
El usuario podrá acceder al sistema mediante una cuenta personal donde se le permitirá mantener un registro de los ficheros que ha tratado y poder consultarlos en cualquier momento, pudiendo además visualizar la información mediante infogramas.

Palabras clave:

CSV, Excel, archivo, datos, información, guardar, descargar, gráficas, analizar, tratar, usuario, historial.

Abstract

In this end-of-grade project, an application is presented that will be able to perform reading, modification and analysis of data contained in excel or csv files that the user wants to deal with.
The user will be able to access the system through a personal account where they will be able to keep a record of the files they have dealt with and be able to consult them at any time, and can also view the information through infograms.

Keywords:
CSV, Excel, file, data, information, save, download, graphics, analyze, treat, user, history.

Índice

1

Introducción

1.1 Motivación

La necesidad de manejar grandes volúmenes de datos en muchos entornos de trabajo es palpable. Normalmente, la adición o la eliminación de estos datos surgen sobre la marcha, debiendo ser relacionados con datos anteriores. Este proceso de relación de datos se realiza de manera completamente manual, dando lugar a posibles errores y a una gran pérdida de tiempo. De aquí surge la motivación de esta aplicación.
Se busca realizar una aplicación capaz de automatizar todos estos procesos anteriormente citados, aumentando la eficiencia y disminuyendo el número de errores que el usuario pudiera generar de forma rudimentaria.

1.2 Objetivos

El objetivo principal de este proyecto es realizar una aplicación web que permita a cualquier usuario, sin necesidad de tener prácticamente conocimientos informáticos, llevar una gestión automatizada de los archivos de datos necesarios para su trabajo personal, pudiendo realizar con ellos múltiples acciones: mostrarlos por pantalla, combinar varios archivos, filtrar la información dentro de ellos para optimizar el trabajo, mantener un historial de todos los archivos tratados dentro de su cuenta personal y analizar la información contenida mediante varios gráficos a elegir por el usuario.

1.3 Estructura de la memoria

La estructura de la memoria se ha organizado siguiendo la metodología usada para el desarrollo del proyecto, la cual ha tenido las etapas que encontramos en cualquier desarrollo software. Así el lector podrá observar la evolución que ha tenido el proyecto de manera iterativa.

1.3.1 Introducción

En este primer capítulo se exponen los motivos por los que surge la necesidad de desarrollar esta plataforma, los objetivos del proyecto, y un resumen del contenido de esta memoria.

1.3.2 Tecnologías utilizadas

En esta sección se explicará el motivo por el que hemos elegido utilizar esas tecnologías.

El lector podrá observar las tecnologías utilizadas dividas por categorías. Entre las explicadas encontramos frameworks, base de datos, herramientas de desarrollo web y librerías java utilizadas.

1.3.3 Metodología y planificación

En esta sección se trata la gestión del proyecto, desde la metodología elegida para el desarrollo del mismo hasta la planificación que hemos realizado para llevarlo a cabo.

1.3.4 Análisis de especificación.

Esta sección es de gran importancia debido a que en ella desarrollamos la primera fase del proyecto, la captura de requisitos.

Describimos con detalle los requisitos tanto funcionales como no funcionales que obtenemos a partir de reuniones celebradas con el tutor del proyecto.

1.3.5 Diseño

En esta sección encontramos la especificación de los casos de uso más significativos. Detallaremos los actores, escenarios y condiciones que se deben cumplir. Gracias a esto podremos realizar la definición del sistema y el diagrama de clases, los cuales serán esenciales para llevar a cabo el proyecto.

1.3.6 Implementación y Pruebas

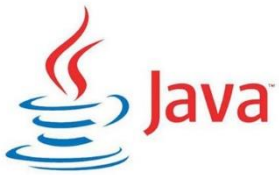
Esta sección describirá la última fase del proceso. Trataremos con detalle la implementación back-end y front-end; se hará un resumen sobre los problemas más importantes que hemos encontrado y cómo los hemos solucionado.

1.3.7 Conclusiones y líneas futuras

En esta última sección de la memoria podremos encontrar un resumen de los conocimientos adquiridos gracias al desarrollo del proyecto, y haremos una comparación entre los objetivos perseguidos al inicio del proyecto y los resultados conseguidos.

2

Tecnologías y herramientas

Plataforma de desarrollo	
	**Java EE** es una plataforma de programación —parte de la Plataforma Java— para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.
SOFTWARE	
	**NetBeans** es un entorno de desarrollo integrado libre, hecho principalmente para el lenguaje de programación Java. Existe además un número importante de módulos para extenderlo

	**Maven** es una herramienta de software para la gestión y construcción de proyectos Java. Tiene un modelo de configuración de construcción más simple, basado en un formato XML. Maven utiliza un **Project Object Model (POM)** para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos.
	**MagicDraw UML** es una herramienta CASE desarrollada por No Magic. Incorporada en el TFG para realizar el modelado del sistema, el diseño y los diagramas de secuencia.
BASE DE DATOS	
	**MongoDB** es un sistema de base de datos NoSQL orientado a documentos de código abierto. MongoDB guarda estructuras de datos BSON (una especificación similar a JSON), permitiendo una flexibilidad y una comodidad a la hora de manejar los datos que no nos aporta una base de datos relacional.
	**Robo 3T** es una interfaz gratuita para la base de datos Mongo, pudiendo realizar múltiples acciones a través de ella, como ver las bases de datos, los objetos que se encuentran en ella y realizar operaciones sobre ellos.

FRAMEWORKS	
	**Spring** es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java. A pesar de que no impone ningún modelo de programación en particular, este framework se ha vuelto popular en la comunidad al ser considerado una alternativa, sustituto, e incluso un complemento al modelo EJB (Enterprise JavaBean).
DESARROLLO WEB	
	**JavaScript** (abreviado comúnmente **JS**) es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se utiliza principalmente en su forma del lado del cliente (*client-side*), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas.
	**JQuery es una librería de JavaScript.** Esta librería de código abierto, simplifica la tarea de programar en JavaScript y **permite agregar interactividad a un sitio web** sin tener conocimientos del lenguaje.

	**D3.js** (o simplemente D3 por las siglas de Data-Driven Documents) es una biblioteca (informática) de JavaScript para producir, a partir de datos, infogramas dinámicos e interactivos en navegadores web.C3js hace mucho más fácil la generación de gráficas basadas en D3js, sin necesidad de escribir código en D3js cuya complejidad es mucho mayor.
	**Bootstrap** es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales.
LIBRERÍAS JAVA	
	**Opencsv** es una librería para Java que integra funciones para el tratado de archivos CSV (separados por coma) de una forma muy sencilla e intuitiva.
	**Apache POI** es una librería para Java que permite la lectura y el tratado de datos que se encuentren archivos con formato XML, XLSX,etc...

3

Metodología y planificación

3.1 Metodología

El desarrollo de este proyecto se ha llevado a cabo mediante una metodología ágil. El desarrollo ágil de software se traduce en un enfoque para la toma de decisiones en los proyectos de software, que se refiere a métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto. Gracias a esta metodología elegida, ha sido posible una muy buena coordinación con mi compañero de proyecto, donde hemos repartido las tareas sin ningún problema en algunos casos, y en otros hemos ejecutado el desarrollo de forma conjunta para facilitar la toma de decisiones y para obtener una mayor eficacia. Esto también nos ha permitido una perfecta coordinación con el tutor del proyecto, el cual nos ha ido proponiendo nuevas funcionalidades a añadir al proyecto conforme este avanzaba de forma progresiva.

3.2 Planificación

A continuación, se muestra una tabla donde se representa las horas empleadas a cada fase del desarrollo.

Fase	Ricardo	Javier
Análisis y captura de requisitos	**20**	**20**
Modelado	**3**	**3**
Diseño	**40**	**40**
Implementación back-end		
Conexión a la base de datos	1	4
Identificar formato ficheros	1	3
Convertir excel a csv	10	14
Elegir archivo principal	0	2
Elegir tipado de las cabeceras	8	8
Elegir cabeceras a mostrar	3	3
Relacionar datos entre ficheros	100	100
Mostrar resultado	7	18
Descargar fichero	1	1
Guardar fichero en base de datos	4	4
Mostrar historial de ficheros	4	0
Gestión de usuarios	19	11
Combinar datos	6	0
Buscador en resultado	6	0
Implementación front-end		
CSS y boostrap	8	34
Gráficas de datos	30	6
Realización de la memoria	**25**	**25**

4

Análisis de requisitos y especificación

4.1 Análisis del diseño

Antes de comenzar a describir los requisitos recopilados, los casos de uso y demás especificaciones tomadas para la realización de este proyecto, introduciremos al lector en la situación encontrada antes de comenzar el mismo.
Hoy en día, cada vez más, nos encontramos ante una gran presencia de la informática en muchos ámbitos del mundo laboral. Esta informatización, conlleva que también surja la necesidad de organizar los datos y ficheros a tratar en una empresa mediante la informática en lugar de realizarlo a papel como típicamente encontrábamos.
El problema que surge de esta nueva necesidad, es que cada empresa o ámbito laboral necesita guardar esta información de una forma muy específica, resultando imposible que cada organismo realice su propio software para este tipo de tareas, y por tanto, esto les lleve a utilizar formatos estándar como puedan ser xlsx o CSV para el tratamiento de estos datos.
Este tipo de formatos son muy útiles por la gran flexibilidad que aportan al usuario a la hora de trabajar con ellos, pero justamente por el hecho de aportar tanta flexibilidad, requiere de muchas acciones manuales por parte del usuario para manejar los datos. Y gracias a todo esto, surge la idea de este proyecto. Realizar un software capaz de automatizar todos estos procesos, aportando al usuario diversas utilidades que le ayuden a incrementar la eficiencia en su trabajo.

Estas utilidades, a grosso modo, son las de procesar los datos al gusto del usuario mediante las columnas que se encuentren en él, y pudiendo realizar operaciones entre varios ficheros, como unir datos, unir columnas, trabajar con distintos tipos de datos en función de lo que el usuario introduzca, y mantener un historial de todos los archivos con los que ha trabajado a lo largo del tiempo.
Además, también se ha añadido la funcionalidad que permite analizar los datos y mostrar representaciones gráficas en función de dos columnas, donde necesariamente una es numérica y la otra debe pertenecer a una categoría a elegir por el usuario.
Por tanto, la idea fue agilizar el proceso manual que el usuario debía realizar, además de aportar más calidad al trabajo pudiendo entender los datos introducidos y mostrarlos mediante gráficas de forma que el usuario visualmente pudiera en pocos segundos agrupar toda la información que quisiera tratar.

4.2. Requisitos

4.2.1. Requisitos Funcionales

En esta sección vamos a listar los requisitos funcionales. Estos requisitos definen la funcionalidad del sistema y definen los roles que tendrán los diferentes usuarios del sistema. Para una mejor compresión del lector los requisitos serán listados por categorías, con un identificador, nombre y una breve descripción.

Las categorías elegidas son sistema y aplicación web, en ellas se encuentran divididas la funcionalidad considerando si pertenece más a la parte del sistema o a la parte web.

Categoría	Nombre	Descripción
Sistema	Conectar a Base de Datos	La aplicación debe estar conectada con una base de datos
	Convertir archivos xlsx a csv	El usuario debe poder subir archivos de tipo xlsx (Microsoft Excel) o CSV de manera indiferente
	Elegir archivo principal	Posibilidad de elegir un archivo principal para realizar las operaciones
	Elegir el tipado	El usuario podrá seleccionar el tipo de datos de cada cabecera

	Relacionar datos	Posibilidad de elegir qué cabeceras de qué archivos se van a combinar y unir los datos por dichas cabeceras
	Unir datos iguales	Posibilidad de unir datos iguales de una misma columna
	Cálculos al unir	Si los datos a unir son de tipo numérico, dar posibilidad de unir mediante suma o promedio

Categoría	Nombre	Descripción
Aplicación Web	Realizar Login	Los usuarios deberán identificarse en la aplicación
	Realizar Logout	Los usuarios podrán cerrar sesión de la aplicación
	Subir Archivos	Opción de subir uno o más archivos para procesar la información
	Ver resultado	Los usuarios podrán ver el resultado de sus operaciones en la web sin necesidad de descargar o guardar el archivo
	Buscador en resultado	El usuario podrá buscar una fila concreta del archivo poniendo el dato en concreto o parte de él
	Guardar Archivo Resultante en Base de Datos	Posibilidad de guardar el archivo resultante en la base de datos para poder ser consultado a posteriori
	Descargar Archivo Resultante	Posibilidad de descargar el archivo resultante en formato .csv
	Ver historial	Los usuarios podrán ver un historial de sus acciones en el sitio web
	Mostrar gráficas	Los usuarios podrán ver diferentes tipos de gráficas sobre los datos de los archivos

4.2.2. Requisitos No Funcionales

A continuación pasamos a listar los requisitos no funcionales, estos no describen funciones, sino características de funcionamiento. Seguiremos un modelo de listado similar al apartado anterior.

Categoría	Descripción
Seguridad	Autenticación basada en la encriptación y desencriptación de contraseñas.
Seguridad	No se permitirá el acceso a usuarios ajenos al sistema.
Seguridad	Solo se permitirá el acceso a los datos pertenecientes al usuario y no a los de ningún otro.
Accesibilidad	El sistema podrá utilizarse mediante las versiones más actuales de cualquier navegador (Google Chrome, Firefox, Safari, Internet Explorer...)
Interfaz	El usuario podrá interactuar con las gráficas que se muestren con los datos
Usabilidad	La navegación por la aplicación debe ser fácil e intuitiva
Usabilidad	La interfaz de la aplicación dará continuamente información al usuario, de que acción está llevando a cabo
Simplicidad	El usuario realizará su acción en pocos casos
Hardware	El sistema se debe de ejecutar en cualquier máquina con Java 8 y una base de datos MongoDB

4.3 Actores

En este apartado describiremos qué actores o roles consideramos en nuestro sistema.

- Usuario Normal: Este actor representa cualquier persona que acceda al sistema.
- Administrador: Este usuario tendrá las mismas funcionalidades que tiene un usuario normal, a excepción de poder añadir nuevos usuarios, que serán creados sin contraseña para el futuro registro en el sistema de los mismos.

5

Diseño

5.1 Descripción de los casos de uso

Título	Elegir archivo principal
Descripción	El usuario tendrá la posibilidad de elegir cuál será el archivo principal entre todos los subidos a la plataforma.
Pre-condición	Haber subido algún fichero al sistema
Post-condición	El fichero seleccionado será el principal a tratar durante el proceso de análisis
Prioridad	Media
Escenario principal	
1. El usuario introduce uno o varios ficheros al sistema 2. El sistema muestra todos los ficheros por pantalla para que el usuario elija el principal 3. El usuario elige el fichero principal 4. El sistema prosigue con el análisis del fichero	
Escenario alternativo	
2.b El usuario solo ha introducido un fichero y el análisis prosigue.	

Título	Elegir el tipado
Descripción	El usuario será capaz de elegir qué tipo de dato contiene cada columna de cada archivo que haya subido.
Pre-condición	Haber subido uno o varios archivos al sistema y haber elegido el archivo principal.
Post-condición	Quedarán registrados el tipo de dato que contenga cada columna de cada archivo.
Prioridad	Alto
Escenario principal	
1.El usuario selecciona el archivo principal 2.El sistema muestra al usuario todas las cabeceras de todos los archivos subidos, dando la opción de elegir qué tipo de dato contiene cada una mediante un seleccionable. 3. El usuario selecciona todos los tipos de datos que contienen las columnas 4. El sistema guarda la información y muestra la siguiente pantalla.	
Escenario alternativo	
2.b Los archivos introducidos no contienen cabeceras y el sistema lanza un error.	

Título	Relacionar datos
Descripción	El usuario podrá seleccionar qué cabeceras va a querer tratar de cada archivo, y además con qué cabeceras se van a combinar.
Pre-condición	Haber subido uno o varios archivos al sistema, haber seleccionado el archivo principal y haber elegido el tipado de cada uno de ellos.
Post-condición	Los datos a tratar quedarán completamente relacionados y guardados en el archivo principal.
Prioridad	Alta
Escenario principal	

1.El usuario selecciona el tipado de los archivos 2.El sistema guarda la información y muestra una pantalla las cabeceras de todos los archivos, dando opción al usuario a relacionarlas entre ellas. 3. El usuario realiza todas las relaciones que crea convenientes. 4. El sistema guarda la información y muestra el resultado final por pantalla.
Escenario alternativo
3.b El usuario solo ha introducido un archivo y el sistema muestra directamente por pantalla el contenido de ese archivo.

Título	Unir datos iguales
Descripción	El usuario tendrá la capacidad de unir los datos que estén duplicados de cualquier columna a elegir, calculando el promedio o la suma de sus datos numéricos asociados.
Pre-condición	Haber guardado un fichero en el sistema.
Post-condición	El fichero quedará modificado con los cambios realizados en el sistema.
Prioridad	Media
Escenario principal	
1.El usuario accede a un archivo registrado en su historial de ficheros. 2.El sistema muestra los datos por pantalla y la opción de combinar los datos. 3.El usuario selecciona la opción de combinar los datos. 4.El sistema modifica los datos, eliminando los datos duplicados y juntándolos en una sola fila, asociando el promedio o la suma a esta fila de los datos numéricos que pertenecieran a cada dato único.	
Escenario alternativo	
4.b El fichero no contiene datos duplicados y el fichero tras el procesamiento contiene la misma información.	

Titulo	Realizar Login
Descripción	Permite al usuario validar su identidad y acceder al sistema.
Pre-condición	El usuario debe introducir usuario y contraseña.
Post-condición	El sistema permite acceso a la aplicación
Prioridad	Alta
Escenario Principal	
1. El usuario introduce usuario y contraseña en el login. 2. El sistema verifica que el usuario se encuentra registrado. 3. El sistema verifica que la contraseña coincide con el usuario. 4. El sistema permite que el usuario acceda y le redirige a la página inicial.	
Escenario alternativo	
2.b El usuario no se encuentra registrado. 2.b1 El sistema lanza un error comunicándole que el usuario no existe. 2.b2 Regresa al paso 1. 3.b La contraseña es incorrecta. 3.b1 El sistema lanza un error comunicando que la contraseña no es correcta. 3.b2 Regresa al paso 1.	

Titulo	Realizar Logout
Descripción	El usuario podrá cerrar su sesión
Pre-condición	El usuario debe tener su sesión abierta
Post-condición	El sistema cierra su sesión con éxito.
Prioridad	Alta.
Escenario Principal	
1. El usuario selecciona la opción de cerrar sesión 2. El sistema cierra la sesión y redirige a la página de login.	
Escenario alternativo	

Titulo	Subir Archivos
Descripción	El usuario podrá subir uno o más archivos al sistema para procesarlos.
Pre-condición	El usuario debe de tener la sesión iniciada.
Post-condición	El archivo será subido al sistema
Prioridad	Muy Alta
Escenario Principal	
1. El usuario hace clic en subir archivo. 2. El sistema muestra un explorador de archivos. 3. El usuario selecciona uno o más archivos a procesar. 4. El sistema redirige al usuario para que seleccione el archivo principal.	
Escenario alternativo	
3.b El usuario no selecciona archivos. 3.b1 El sistema cierra el explorador de archivos.	

Titulo	Ver resultado.
Descripción	El usuario podrá ser capaz de ver el resultado de sus archivos procesados.
Pre-condición	El usuario debe haber subido uno o varios archivos, elegido archivo principal, y elegido cabeceras a tratar.
Post-condición	El archivo resultante es mostrado con éxito
Prioridad	Alta
Escenario Principal	
1. El usuario llega hasta la página de ver resultado. 2. Visualiza el resultado de sus operaciones.	
Escenario alternativo	

Titulo	Buscar en resultado
Descripción	El usuario podrá realizar una búsqueda dentro del resultado obtenido
Pre-condición	Haber procesado uno o más archivos
Post-condición	Obtener el resultado de la búsqueda
Prioridad	Media
Escenario Principal	
1. El usuario hace clic en el input de búsqueda 2. El usuario selecciona si quiere una búsqueda que contenga la palabra o que sea igual a la palabra. 3. El sistema busca según los parámetros. 4. El sistema muestra el resultado dinámicamente.	
Escenario alternativo	
4.b El sistema no encuentra ningún dato con esos parámetros. 4.b1 El resultado mostrado será la tabla vacía.	

Titulo	Guardar archivo resultante
Descripción	El usuario podrá guardar el archivo resultante en el sistema para su posterior consulta
Pre-condición	Haber subido y procesado uno o más archivos
Post-condición	El archivo resultante es guardado en la base de datos.
Prioridad	Alta
Escenario Principal	
1. El usuario visualiza el resultado obtenido. 2. El usuario introduce un nombre para el archivo. 3. El usuario hace clic en guardar. 4. El sistema guarda el archivo con el nombre introducido. 5. El sistema redirige al usuario a la página inicial.	
Escenario alternativo	
2.b El usuario no introduce un nombre para el archivo. 2.b1 El sistema lanza un aviso de que no puede guardar el archivo sin nombre.	

Titulo	Descargar archivo resultante
Descripción	El usuario podrá descargar el archivo a su dispositivo.
Pre-condición	Haber subido y procesado uno o más archivos
Post-condición	El archivo resultante es descargado con éxito.
Prioridad	Alta
Escenario Principal	
1. El usuario visualiza el resultado obtenido. 2. El usuario introduce un nombre para el archivo. 3. El usuario hace clic en descargar. 4. El sistema descarga el archivo con el nombre introducido. 5. El sistema redirige al usuario a la página inicial.	
Escenario alternativo	
2.b El usuario no introduce un nombre para el archivo. 2.b1 El sistema lanza un aviso de que no puede descargar el archivo sin nombre.	

Titulo	Ver historial
Descripción	El usuario podrá ver un historial con los archivos procesados por él mismo.
Pre-condición	Haber subido, procesado y guardado uno o más archivos.
Post-condición	Mostrar una lista con todos los archivos procesados.
Prioridad	Alta
Escenario Principal	
1. El usuario se encuentra en la página inicial. 2. El sistema muestra un listado con los archivos procesados. 3. El usuario puede clicar sobre uno de los archivos y podrá visualizarlo.	
Escenario alternativo	
2.b El usuario no ha introducido ningún fichero en el sistema y la lista se encuentra vacía.	

Titulo	Mostrar gráficas
Descripción	El usuario podrá ver gráficas con representaciones sobre sus datos.
Pre-condición	Haber procesado y guardado uno o más archivos. Tener al menos una cabecera del tipo “enum” y al menos otra del tipo “numeric”.
Post-condición	La gráfica muestra los datos.
Prioridad	Alta
Escenario Principal	
1. El usuario selecciona que tipo de gráfica quiere visualizar. 2. El usuario hace clic sobre el botón visualizar. 3. El sistema le muestra la gráfica interactiva con sus datos correspondientes.	
Escenario alternativo	
3.b El fichero que el usuario desea visualizar no contiene datos suficientes o bien tipados para poder visualizarlos.	

5.2 Modelo de clases

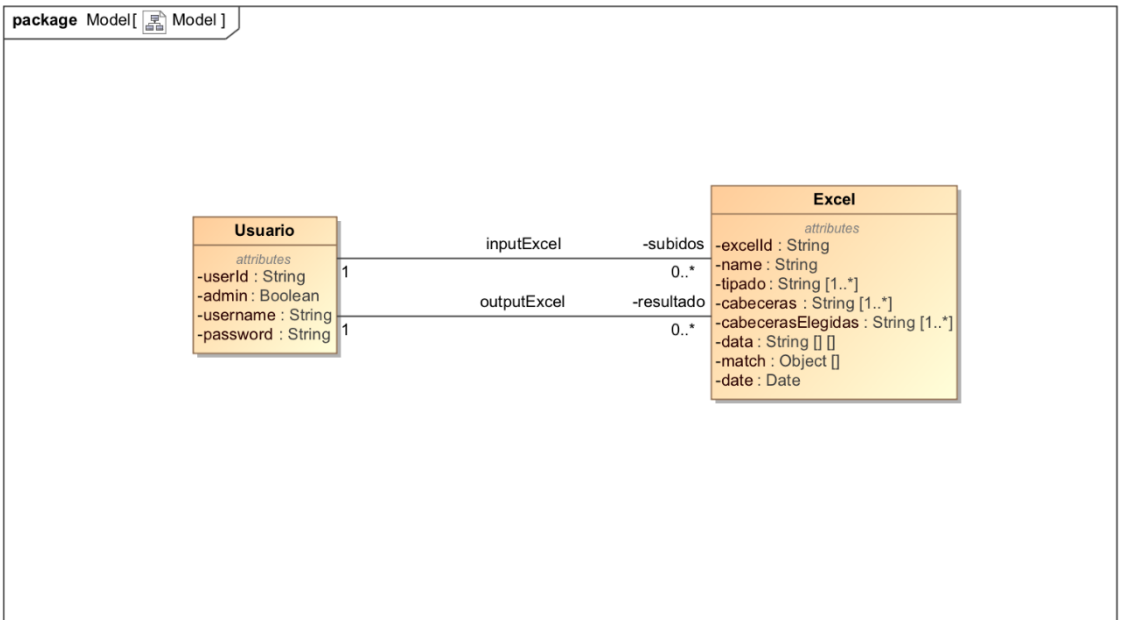


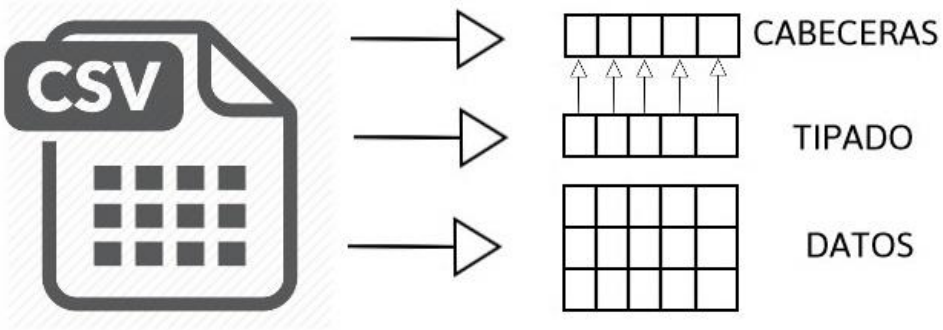
Ilustración 5.1: Diagrama de clases del sistema

El modelo de clases de nuestro sistema es muy sencillo, simplemente se compone de dos clases, una clase Usuario y una clase Excel (Ilustración 5.1.).
En la clase **Usuario** guardaremos sus datos básicos, como su nombre, su contraseña (encriptada), un ID único y un atributo booleano que determinará si es administrador o no. Este control de permisos lo hemos podido realizar de

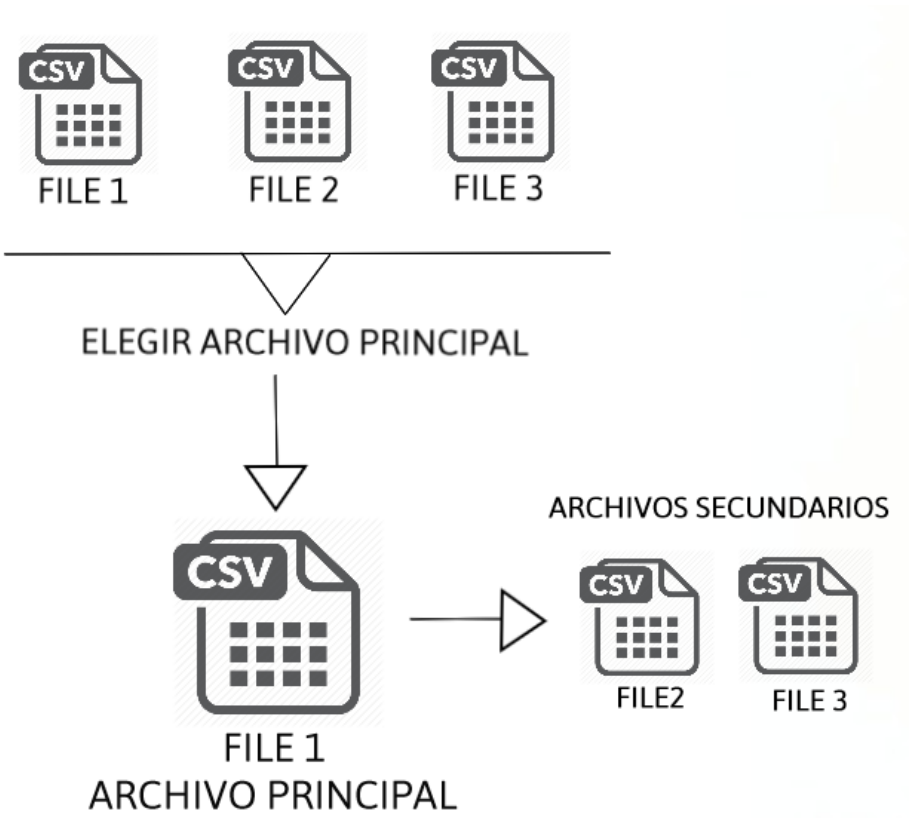
esta forma puesto que en nuestra aplicación no hay más de dos tipos de usuarios, y por tanto solo podemos encontrar dos tipos de privilegios, siendo un atributo booleano suficiente para controlar esto.
La clase **Excel** es la que representará a cada uno de los archivos que el usuario suba a la plataforma. Se compone de varios atributos que servirán para tratar cada uno de los ficheros y poder analizarlos.

- excelID : ID único para identificar cada archivo.
- name : El mismo nombre que el archivo que ha subido el usuario.
- tipado [] : Array que contendrá el tipo de dato (Numérico, String o Enumerado) que compone cada columna del archivo. Este array coincidirá en índice con el de cabeceras y con la matriz data.
- cabeceras [] : Array que contendrá las cabeceras del archivo. Necesariamente debe haber alguna para poder procesar el archivo.
- cabecerasElegidas [] : Array auxiliar que contendrá las cabeceras que se han elegido en la parte de realizar el match con el archivo principal. Su única función es la de facilitarnos el trabajo a la hora de la implementación.
- data [][] : Matriz de datos que contiene toda la información del fichero (incluidas las cabeceras).
- date : Fecha de inserción del archivo en la plataforma.
- match : Este atributo es uno de los más importantes y sobre todo complejos de entender de la aplicación. Por ello, su explicación vendrá acompañada de ilustraciones para facilitar el entendimiento de la estructura de datos utilizada, el por qué de la misma, y cómo se utiliza para conectarla con el resto de atributos.

Para empezar, un objeto tipo Excel/CSV contendrá básicamente la siguiente información, ya explicada previamente.



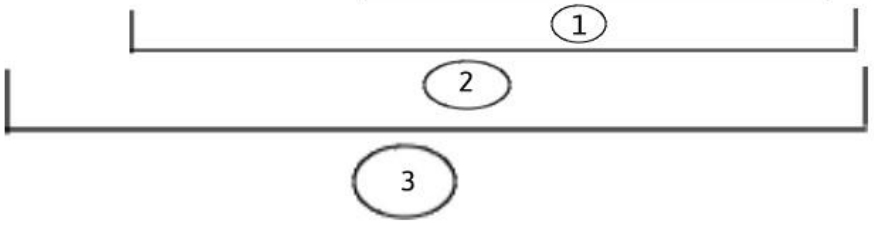
A partir de aquí, las ilustraciones mostrarán un ejemplo básico de tres archivos que un usuario sube a la aplicación para tratarlos conjuntamente.



Teniendo en cuenta que File 1 actuará como archivo principal, y el resto como secundarios, se explicará qué estructura se ha utilizado para tratar los datos de todos los archivos y los pasos a seguir para poder llevarlo a cabo.

Estructura de datos elegida para relacionar los archivos

```
private LinkedList<LinkedHashMap<String, LinkedHashMap<String, LinkedList<String>>>> match;
```



1 LinkedHashMap<String,LinkedList<String>>

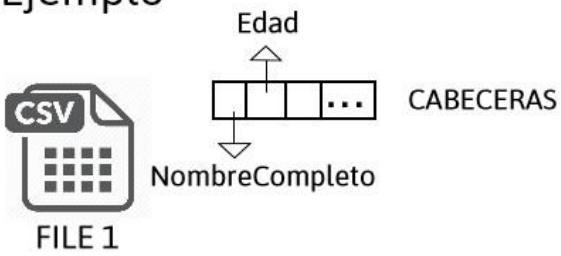
ID CABECERAS ELEGIDAS

Ejemplo:
(<IDFILE2,{Nombre, Apellido}>,< IDFILE3,{NombreTotal}>)

2 LinkedHashMap<String,LinkedHashMap<String,LinkedList<String>>>

CABECERA ARCHIVO PRINCIPAL

Ejemplo



Se hace match de la cabecera NombreCompleto, y con lo generado en 1 :

(NombreCompleto, (<IDFILE2,{Nombre,Apellido}>,<IDFILE3,{NombreTotal}>))

De este modo, ya sabemos qué cabeceras del archivo principal
están relacionadas con qué cabeceras del resto de archivos

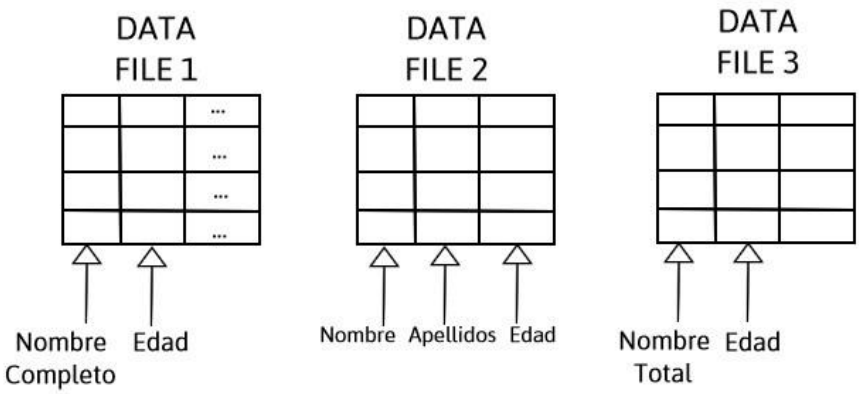
(3) LinkedList<LinkedHashMap<String,LinkedHashMap<String,LinkedList<String>>>>

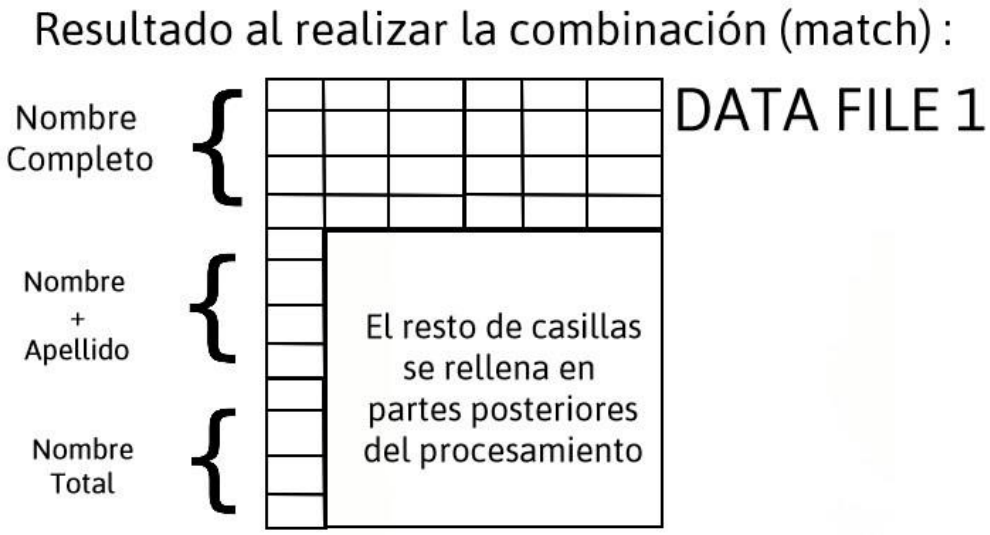
Con esta última lista recogemos todos los match del archivo principal

Ejemplo

{(NombreCompleto,(<IDFILE2,{Nombre,Apellidos}>,<IDFILE3,{NombreTotal}>)),
(Edad,(<IDFILE2,{Edad}>,<IDFILE3,{Edad}>))}

Hasta aquí, ya hemos logrado ver cómo se relacionan las cabeceras (que representan a una columna entera de datos) juntando unas con otras.
Ahora, vamos a ver cómo utilizamos esta estructura ya formada para leerla y procesar los datos.





Finalmente, este sería el último paso del procesamiento. Como muestra la imagen, se añaden las correspondientes casillas a la matriz de datos del *File 1.* El ejemplo solamente muestra cómo añadir la columna del nombre, a partir de aquí, básicamente sería realizar lo mismo con el resto de columnas seleccionadas (como la edad, que aparece también en el ejemplo).
En el caso de que haya columnas de los archivos secundarios seleccionadas pero que no hayan sido marcadas para realizar el match, simplemente se añadirán a la derecha de la matriz de datos del archivo principal en su correspondiente posición.

5.3 Arquitectura del sistema

El proyecto trata de crear una aplicación de escritorio completa, implementando un servidor REST basado en microservicios donde un cliente realizará peticiones para el tratamiento de los datos.
El servidor se ha implementado en Java utilizando el framework Spring Boot, del que se ha utilizado su estructura propia basada en el patrón arquitectónico Modelo-Vista-Controlador.

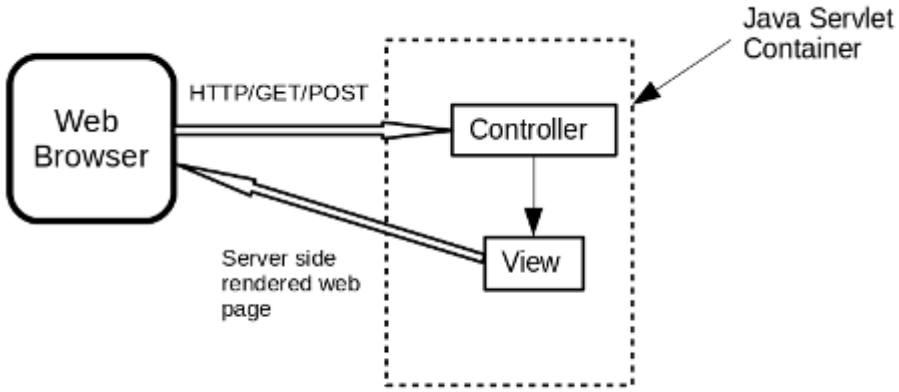


Ilustración 5.2: Modelo arquitectónico de Spring Boot

Por tanto, todas las funcionalidades descritas en los requisitos del sistema, estarán implementadas únicamente en el Controlador del sistema, el cual se encargará de recoger las peticiones del usuario, procesarlas y comunicarse con el modelo del sistema para llevarlas a cabo.

5.4 Base de datos MongoDB

Con el fin de realizar una mejor y más completa aplicación decidimos almacenar datos relacionados con los usuarios y las operaciones que realiza en la aplicación. Spring podía relacionarse con base de datos tanto relacional como no relacional por lo que estudiamos cual nos convendría mejor en nuestra aplicación.

Nos decidimos por MongoDB por varios motivos: en primer lugar los datos almacenados por MongoDB son de fácil lectura para el usuario debido a que son almacenados en formato JSON, en segundo lugar vimos que MongoDB tenía un comportamiento excelente para trabajar con sistemas de gran volumen. Por otro lado, vimos algunas desventajas como que los documentos de MongoDB nos permiten almacenar toda la información que queramos, lo que nos podría

ocasionar problemas en la consistencia de datos. La integración con la Spring, la tecnología utilizada en nuestra aplicación se presenta bastante fácil y sencilla. Para conectar la aplicación con la base de datos se han tenido que hacer algunas configuraciones.
En primer lugar, definir las clases java con todos los atributos a contener para guardarla en la base de datos. En nuestro caso, tendremos la clase Usuario y la clase Excel.

```
@Document(collection="Excel")
public class Excel {
    private ExcelRepository excelRepository;
    @Id
    private String id;
    @DBRef
    private Usuario usuario;

    private Date fecha;
    private String[] tipado;
    private String nombre;
    private String[] cabeceras;
    private LinkedList<String> cabecerasElegidas;
    private String[][] data;
    private LinkedList<LinkedHashMap<String, LinkedHashMap<String, LinkedList<String>>>> match;
```

Ilustración 5.4.1: Código Java que definen los atributos de una clase

Se observa cómo están definidos todos los atributos que la clase deberá tener para el correcto funcionamiento de la aplicación.
En concreto, destacar las siguientes etiquetas:

- Id : esta etiqueta se coloca en el atributo que actuará como ID en la base de datos. MongoDB lo añade de forma automática.
- DBRef : esta etiqueta se coloca en los atributos que actúan como referencias a otros objetos u atributos. En este caso, indicamos que cada objeto tipo Excel tendrá un Usuario.
- Document : esta etiqueta define a una clase como apta para poder ser guardada y tratada en la base de datos.

Por otro lado, es necesario crear otra clase Java que actúe como enlace con la base de datos y el modelo de la aplicación.

```
@Repository
public interface ExcelRepository extends MongoRepository<Excel, String> {

    Excel findByNombre(String nombre);

}
```

Ilustración 5.4.2: Código que define el repositorio para conectar con Mongo

Simplemente deberemos crear una interfaz en Java que extienda de la clase MongoRepository, indicándole qué clase se va a tratar y de qué tipo es el ID del objeto, y la etiqueta @Repository para indicar que es un repositorio de datos. El resto de configuraciones ya vienen integradas en esta clase y se añaden al extenderla de una forma muy sencilla.
Una vez creada esta interfaz y definidas las clases, toca comenzar a realizar operaciones con la base de datos.
Todas las operaciones se realizarán en la clase Controller, la cuál contiene todos los métodos de la aplicación. Para ello, se define un objeto en la clase de tipo ExcelRepository

```
@Controller
@SessionAttributes("map")
public class ExcelController {

    @Autowired
    private ExcelRepository db;
    @Autowired
    private UsuarioRepository dbUser;
```

Ilustración 5.4.3: Atributos de repositorios para interactuar con Mongo

Como se observa en la imagen, también hay un objeto de la clase UsuarioRepository. Este objeto es similar al explicado con ExcelRepository, a diferencia que trata objetos de tipo Usuario.
Es necesario definirlas con la etiqueta @Autowired, para que se definan automáticamente las propiedades del bean en el XML de configuración y todo funcione correctamente.
Realizado todo esto, se utilizan las operaciones que nos aporta la librería para tratar con la base de datos.

- Save : para guardar o actualizar un objeto en la base de datos
- Find : para obtener un objeto en la base de datos (puede ser buscado por ID, o por otro criterio definido).
- FindAll : para obtener una colección de objetos del mismo tipo almacenados en la base de datos.

6

Implementación y pruebas

6.1 Gestión de usuarios

En esta sección trataremos los problemas y soluciones que hayamos encontrado para tratar la gestión de usuarios en el sistema.

6.1.1 Registrar usuario

El proceso de registrar a un usuario según se ha planteado requiere de algunos procedimientos previos.

Normalmente, en cualquier web, un usuario tiene la capacidad de pulsar un botón para registrarse en la plataforma, donde solamente deberá rellenar sus datos.

En nuestro caso, como no queremos permitir el acceso a la plataforma a cualquiera, una cuenta debe estar autorizada y añadida previamente en el sistema por un usuario con el rol de administrador.

Entrando más en profundidad en esta parte, cuando un administrador decide añadir a un usuario al sistema, no posee ningún tipo de ID de usuario para hacer comprobaciones sobre si el usuario existe ya en el sistema o no. Por tanto, es obligatorio buscar a los usuarios por su nombre de usuario, y esto implica que los nombres de usuarios sean únicos en el sistema.

Para evitar nombres duplicados en el sistema, antes de añadirlo se realiza esta comprobación.

```
@RequestMapping("/createUser")
private String createUser(@RequestParam("username") String username, HttpSession sesion, Model model) {
    Usuario myUser = getCurrentUser(sesion);
    model.addAttribute("mostrarBack", true);
    Usuario aux = dbUser.findByUsername(username);
    if (aux != null) {
        model.addAttribute("back", false);

    } else {

        model.addAttribute("back", true);
        Usuario u = new Usuario();
        u.setUsername(username);
        dbUser.save(u);

    }
```

Ilustración 6.1: Código utilizado para añadir un nuevo usuario al sistema

Como podemos observar, en caso de que el usuario que queremos añadir no exista en el sistema, se crea un nuevo usuario y se añade al sistema sin contraseña.
Cumplida la labor del administrador, a continuación vendrá la parte donde el usuario se registra definitivamente en el sistema añadiendo su contraseña personal.

Cuando un usuario sin contraseña decide acceder al sistema, se le redirige directamente a una pantalla donde se le pide que añada su contraseña.
Como la contraseña del usuario debe ser privada, esta tiene que ser introducida en la base de datos encriptada, de modo que solamente el usuario al que pertenece dicha cuenta sepa cuál es y sea el único capaz de desencriptarla.
Para la encriptación/desencriptación de la contraseña, hemos utilizado una librería de Java llamada Jasypt, la cual contiene funciones básicas para realizar estas operaciones sin mayor esfuerzo.
Por tanto, como se muestra en la imagen, al registrar un nuevo usuario, se busca en la base de datos por su nombre de usuario (ya que estamos seguros de que estará), y se le añade la contraseña encriptada.

```
@RequestMapping("/registrar")
public String registrar(Model model, @RequestParam("pass") String password, HttpSession sesion) {

    BasicTextEncryptor textEncryptor = new BasicTextEncryptor();
    String privateData = password;
    textEncryptor.setPasswordCharArray(password.toCharArray());
    String myEncryptedText = textEncryptor.encrypt(privateData);
    String name = (String) sesion.getAttribute("nameUser");
    Usuario u = dbUser.findByUsername(name);
    u.setPassword(myEncryptedText);
    dbUser.save(u);
    return "upload";
}
```

Ilustración 6.2: Código utilizado para registrar un nuevo usuario

Como apunte, destacar que esta librería contiene también algunos métodos para cambiar el algoritmo de encriptación, pero en nuestro caso, como la seguridad en el sistema no era especialmente prioritaria, decidimos utilizar el algoritmo de encriptación que posee por defecto.

6.1.2 Realizar login

Ante el acceso al sistema de un usuario, podemos encontrar los siguientes casos:

1. El usuario existe e introduce de forma correcta la contraseña.
2. El usuario existe y no introduce bien la contraseña.
3. El usuario está añadido el sistema pero aún no tiene contraseña (debe registrarse).
4. El usuario no existe.

Para descartar el caso 4, la primera comprobación que realizamos en este método es comprobar si el usuario existe.
Después de esta comprobación, verificamos si el usuario tiene contraseña. En caso de no tenerla, se redirige al usuario a la pantalla de registro.
Y en el caso de tenerla, comienza la parte más complicada del método.
Como hemos nombrado en la explicación del método registrar usuario, la contraseña se introduce encriptada en la base de datos. Para autenticar al usuario, lo que hacemos en este caso es intentar desencriptar está clave con la contraseña que el usuario ha introducido por el formulario.
La librería de Jasypt que hemos utilizado, en caso de que la palabra que desencripta un mensaje no sea la misma que la que lo encriptó, lanza una Excepción, por eso observamos que está metido dentro de un try/catch.
En caso de entrar por el try, la contraseña sería correcta, y en caso contrario, la contraseña sería fallida.

```
public String login(Model model, @RequestParam("username") String usuario, @Requ
    Usuario user = dbUser.findByUsername(usuario);
    model.addAttribute("errorPass", false);
    model.addAttribute("error", false);
    if (user != null) {
        if (user.getPassword() != null) {
            try {
                String pass = user.getPassword();
                BasicTextEncryptor textEncryptor = new BasicTextEncryptor();
                textEncryptor.setPasswordCharArray(password.toCharArray());
                String plainText = textEncryptor.decrypt(pass);
                if (plainText.equals(password)) { // login ok
                    sesion.setAttribute("usuario", user);
                    model.addAttribute("listaOutput", user.getOutputExcels());
                    model.addAttribute("admin", user.getAdmin());
                    model.addAttribute("mostrarBack", false);
                    return "upload";
                }

            } catch (EncryptionOperationNotPossibleException e) {
                model.addAttribute("errorPass", true);
                return "index";
            }

        } else { // No existe el usuario y se registra
            sesion.setAttribute("nameUser", usuario);
            sesion.setAttribute("usuario", user);
            return "registrar";
        }
    } // no existe el usuario
```

Ilustración 6.3: Código para realizar el login al sistema

6.2 Tratamiento de ficheros

En esta parte se analizará de forma general cómo hemos decidido resolver el tratamiento de los ficheros y se entrará más en detalle en las partes más específicas y con mayor dificultad que necesiten aclaración explícita.

6.2.1 Subir ficheros

La subida de ficheros al sistema no ha supuesto una gran complicación. La única restricción que hemos impuesto al usuario es que solamente pueda subir archivos con formato .xlsx o .csv, algo que se resuelve añadiendo la siguiente opción en el input del html: multiple accept=".xlsx,.csv".

Después de esto, el tratamiento de los datos se ha realizado siempre mediante el formato en csv, por tanto, cuando nos encontramos en la aplicación con un archivo .xlsx, le realizamos la conversión a csv con la librería Apache POI. Esta librería trae integradas un montón de funcionalidades que nos ha permitido trabajar con comodidad con estos ficheros, pudiendo tener en cuenta casos como espacios en blanco, casillas NULL, etc... sin mucho problema.

6.2.2 Elegir el tipado

Elegir el tipado de un archivo es una parte importante del sistema. Básicamente, nos encontramos con que el usuario puede introducir un archivo donde una columna sea tipo numérica y otra simplemente sean cadenas de caracteres. Para poder hacer en un futuro un análisis de estos datos, es necesario tipar las columnas.

Esto se realiza mediante unos seleccionables donde el usuario debe introducir a mano el tipo de cada columna.

El único problema que encontramos en este método, es que cuando se introduce más de un archivo, la lista recogida por el html no es capaz de diferenciar a qué archivo pertenece la cabecera puesto que todos están en la misma lista.

Para solucionar este problema, simplemente comenzamos a utilizar LinkedList, una estructura donde somos capaces de guardar el orden según se han añadido los datos. Por tanto, fuimos iterando por LinkedList para asociar el tipado a las cabeceras de cada archivo.

La importancia de poder guardar este orden mediante la LinkedList ha sido vital para la resolución del proyecto, tanto en este método como en muchos que vienen a continuación.

6.2.3 Elegir cabeceras

Es posible que un usuario no desee tratar todos los datos contenidos en un fichero, sino unas cuantas columnas solamente. Con esta función, el usuario será capaz de seleccionar qué columnas desea mantener y cuáles no. La única dificultad encontrada es la misma que la encontrada al elegir el tipado, pero al haber encontrado ya la solución, no requirió gran esfuerzo.

6.2.4 Elegir el archivo principal

Como hemos comentado en el apartado de diseño, el archivo principal es una parte importante del programa, puesto que sobre él irán las relaciones con el resto de archivos.

El usuario será capaz de elegirlo entre todos los archivos subidos a la plataforma. No llevó gran dificultad.

6.2.5 Relacionar datos entre ficheros

Sin duda, el método principal y más costoso de la aplicación.

Surge de la necesidad de poder relacionar datos entre distintos ficheros, y poder combinarlo de forma automatizada sin necesidad de copiar y pegar.

Surgieron muchos problemas para llegar a la solución de esta función, ya que tuvimos que montar una estructura bastante compleja, donde cada detalle no contemplado llevaba a un error que costaba incluso encontrar.
Finalmente, la estructura elegida fue la que se ha explicado en el apartado de diseño.
La única parte a detallar, es que todas las estructuras sean Linked. Esto, al igual que con el tipado, es para poder preservar el orden en que se han introducido los datos en la estructura, hecho fundamental para que todo funcione correctamente. De no hacerlo así, a veces encontrábamos datos en posiciones incorrectas, ya que al tener que realizar todo el análisis mediante iteradores y bucles, de no guardar el orden, no teníamos en qué basarnos al iterar pues no sabíamos qué ítem podíamos encontrar.
Como ya se ha explicado mediante las ilustraciones el funcionamiento de este método, no se incluirán fragmentos de código, puesto que realmente, lo complejo fue construir la estructura de forma ordenada. Hecho esto, el código consiste en recorrer bien la estructura iterando por ella e ir relacionando los datos.

6.2.6 Mostrar resultado

Esta funcionalidad es la que muestra por pantalla al usuario el resultado del tratamiento total de los datos.
Otra parte donde tampoco encontramos gran problema, puesto que solamente había que recoger la matriz de datos del archivo principal, pasarla al html y mostrarla. El único problema que pudimos encontrar fue aprender a iterar una matriz con la tecnología de Spring.
Algo que solucionamos rápidamente al descubrir el parámetro th:each, que nos permitía iterar por listas con mucha facilidad.

```
<div id="divScroll">
    <table id="myBlock" class="table table-striped table-bordered" style="width:100%">
        <tr class="item" th:each="row: ${matrix}">
            <td class="name" th:each="value: ${row}" th:text="${value}" />
        </tr>
    </table>
</div>
```

Ilustración 6.4 : Código html para mostrar matriz

6.2.7 Guardar y descargar archivos

Cuando un archivo ha sido procesado de forma completa, se le muestra al usuario por pantalla una especie de borrador antes de realizar ninguna inserción en la base de datos. Si el usuario queda contento con el resultado, podrá guardar el resultado solamente en la base de datos, o podrá descargar el resultado en un fichero para almacenarlo en su ordenador personal y poder transferirlo.

Esto no llevó grandes problemas, uno pudo ser el hecho de diferenciar estas dos acciones, puesto que primeramente solamente implementamos la descarga, pero pensamos que no era del todo cómodo para el usuario. El otro fue guardar los datos en un formato correcto, bien separado por comas, y tratar casos especiales como espacios en blanco, dobles comillas, etc...

6.2.8 Buscador en resultado

Filtro que permite al usuario realizar búsquedas dentro de la matriz de datos resultantes.
Implementado al 100% en JavaScript. Esto supuso uno de los problemas a la hora de realizarlo, puesto que nuestra formación en este lenguaje era un poco escasa.
El segundo gran problema que surgió, fue el hecho de conectar Java Spring con JavaScript, ya que en internet hay muy poca información sobre esto, y pasar un objeto completo desde Java y leerlo sin problema en JavaScript no es algo intuitivo.
Finalmente, tras muchas búsquedas, se encontró la solución.

```
<script th:inline="javascript" defer>
    var matriz = /*[[${matrix}]]*/;
    var tipado = /*[[${tipado}]]*/;
    var header = /*[[${header}]]*/;
```

Ilustración 6.5: Código JavaScript para recoger datos de Java

De este modo, aunque parezca un comentario, es como se recoge un objeto completo pasado desde Spring en JavaScript.
A partir de ahí, el tratamiento de datos y el filtro no fue más que realizar bucles y mostrar y ocultar elementos para obtener el resultado esperado.

6.2.9 Combinar datos

Este método surgió de la idea de poder dotar de un poco de inteligencia a la aplicación.
Hasta ahora, con la función de relacionar datos, simplemente se podían copiar y pegar columnas una debajo de la otra según el usuario decidiera.
Con esto método hemos resuelto la necesidad de combinar los datos que pertenezcan a un mismo "ítem".
Para facilitar la comprensión, se describirá mediante un ejemplo (Ilustración 6.6.1).
Como se observa, hemos combinado dos ficheros donde en la columna Grado aparecen los mismos elementos más de una vez.
Gracias al método de combinar datos, vamos a ser capaces de combinar los datos de estos elementos en una sola fila, eligiendo si realizar la suma o el promedio de sus datos numéricos. En este caso hemos elegido realizar la suma.(Ilustración 6.6.2)

A priori, puede parecer un método complejo de realizar, de hecho lo es, pero gracias al haber tratado los datos con estructuras como el mapa al relacionar los datos, el tratado de los mismos en este método fue muy similar y se agilizó mucho su realización.

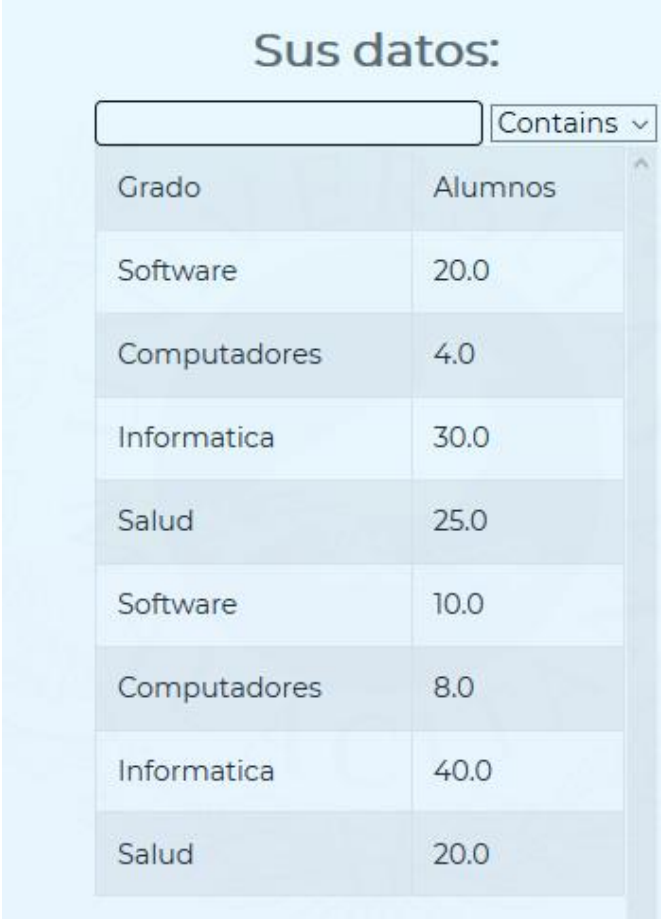


Ilustración 6.6.1 : Ejemplo de datos de fichero sin combinar

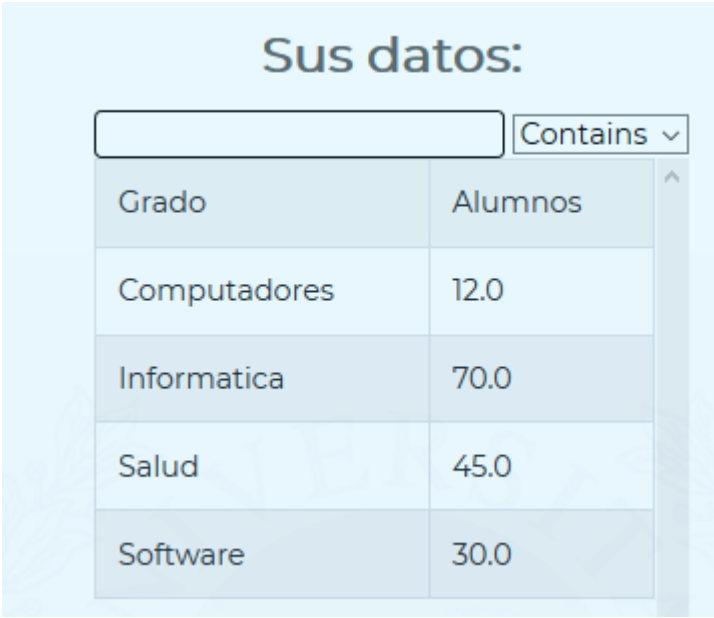


Ilustración 6.6.2 : Ejemplo de datos de fichero combinados

6.2.10 Ver historial de ficheros

La necesidad de tener un registro sobre qué archivos se han tratado desde que el usuario accedió a la plataforma se resuelve con esta funcionalidad.

Se ha implementado de modo que se muestra una lista donde cada archivo es enlace y redirige a la página donde se muestra el resultado realizando una petición GET.

La única complicación encontrada en este caso, fue averiguar cómo pasar por parámetro el ID del archivo en la URL, ya que en Spring se realiza de una forma poco intuitiva.

```
<li th:each="archivo : ${listaOutput}" >
    <!-- ${category.idCategory} -->
    <a  th:href="@{~/getHistorial/{id}(id=${archivo.id})}"> <label  th:text="${archivo.nombre}"></label>
        <label  th:text="${archivo.fecha}"></label>  </a>
</li>
```

Ilustración 6.7 : Código html para pasar parámetros por URL

6.2.11 Gráficas de datos

Esta es otra parte de la aplicación implementada 100% en JavaScript, algo que en principio causó problemas en la velocidad a la que avanzábamos, por la poca información que encontramos para conectar Java Spring con JavaScript.

Para mostrar las gráficas de datos, hemos utilizado una librería de JavaScript llamada C3.js, la cual permite con mucha facilidad mostrar muchos tipos de gráficas modificando apenas unos pocos parámetros.

Como casi todas las gráficas, suele haber un eje X donde normalmente encontramos un parámetro enumerado y un eje Y donde encontramos un parámetro numérico o viceversa.

La forma en que la librería C3.js tiene de pasar estos datos, es la siguiente:

```
var chart = c3.generate({
    data: {
        columns: [
            ['data1', 30, 200, 100, 400, 150, 250],
            ['data2', 130, 100, 140, 200, 150, 50]
        ],
```

Ilustración 6.8.1 : Ejemplo datos C3.js

El mayor problema que encontramos, fue que estos datos están añadidos de forma manual, y al no conocer primeramente como utilizar esta librería, no sabíamos cómo cambiar de forma dinámica estos datos, en función de los datos encontrados

en el archivo del usuario. Finalmente encontramos la existencia de la siguiente opción:

```
var chart = c3.generate({
    data: {
        rows: [
            ['data1', 'data2', 'data3'],
            [90, 120, 300],
            [40, 160, 240],
            [50, 200, 290],
            [120, 160, 230],
            [80, 130, 300],
            [90, 220, 320],
        ]
```

Ilustración 6.8.2 : Ejemplo datos C3.js con matriz

Como vemos, se pasa por parámetro una matriz completa, donde cada columna representa un parámetro del eje X, y el resto de datos de su columna son los datos asociados al eje Y de ese parámetro.
Para poder incorporar la matriz tal y como la librería requiere, se ha tenido que realizar en JavaScript una especie de conversión mediante un mapa, para dejar las columnas únicas y asociar los valores justo debajo de donde corresponda.
Destacar también que se ha implementado de modo que los datos pueden cambiar de forma dinámica, donde el usuario elige mediante dos seleccionables qué columnas desea ver en los gráficos, donde una debe ser necesariamente tipo enumerado y otra numérica.

La siguiente imagen muestra un diagrama de barras, utilizando el ejemplo tratado en el apartado de combinar datos.

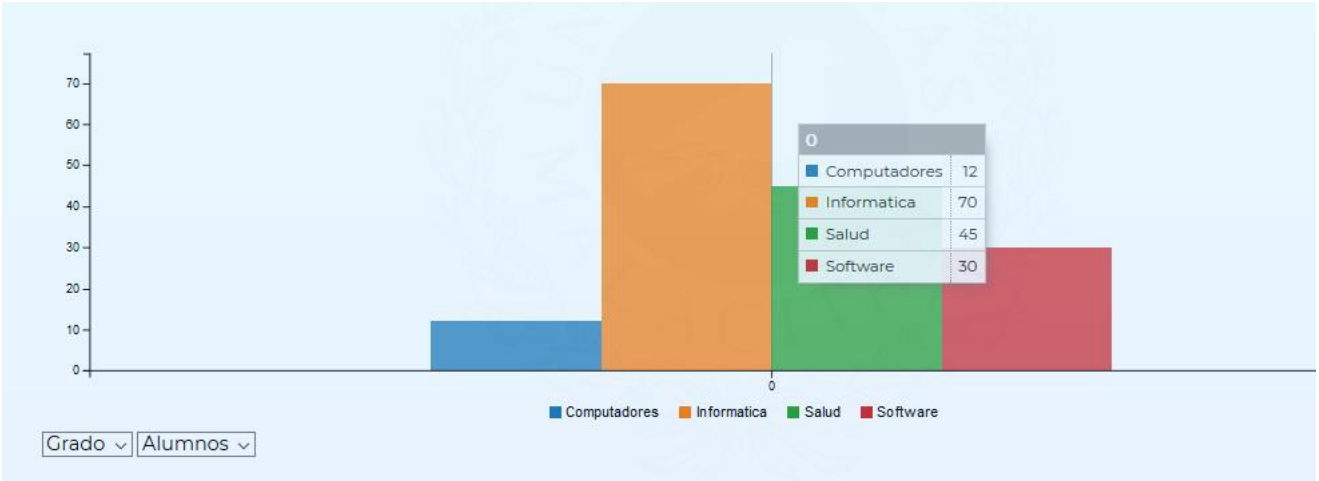


Ilustración 6.9 : Ejemplo de diagrama de barras de la aplicación

6.3 Diagramas de secuencia

En este apartado se incluirá un diagrama de secuencia que describe la funcionalidad principal del sistema que un usuario llevará a cabo cuando utilice la aplicación.

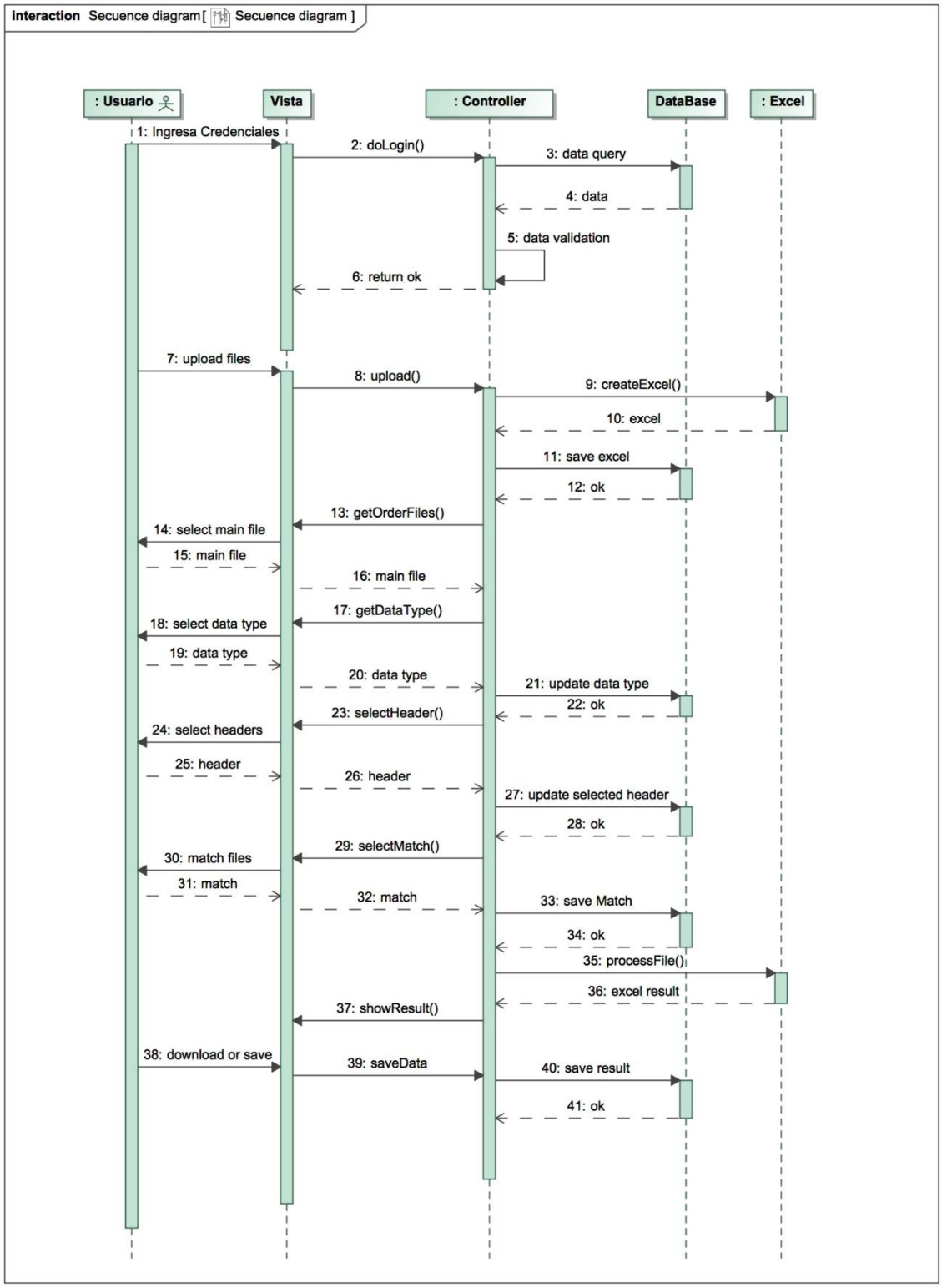


Ilustración 6.10 : Diagrama de secuencia principal del sistema

6.4 Pruebas

La parte destinada a las pruebas del sistema se han basado únicamente en depurar la aplicación. Como ingenieros, sabemos de la importancia de realizar también esta última parte de un modo más profesional y formal, pero la falta de tiempo no nos ha permitido hacer otra cosa.
Por tanto, lo único que se podría considerar como pruebas del sistema ha sido ir probando la aplicación poco a poco conforme se iba construyendo, y utilizar el depurador de Netbeans para encontrar errores de forma más fácil.

7

Conclusiones y líneas futuras

7.1 Resultado final y conclusiones

Como resultado final del proyecto, se ha conseguido realizar con éxito un proyecto donde se procesen archivos con formato .xlsx y .csv realizando acciones, que anteriormente se debían hacer de forma manual o a través de herramientas con cierta complejidad, en nuestro caso de forma automatizada.
Además, gracias a la velocidad de trabajo, también se ha conseguido tipar los datos y mostrar gráficos con la información contenida, algo que a priori no se sabía si iba a estar incluido.
Por tanto podría decirse que el resultado ha sido bastante satisfactorio por haber cumplido las expectativas iniciales.

En cuanto a las conclusiones finales del proyecto, voy a dividirlas en función de la etapa de desarrollo.

- Análisis: la parte más importante del proyecto sin duda, puesto que cualquier duda sobre qué quiere exactamente el cliente lleva a una pausa forzada en el avance del proyecto y contradicciones con mi compañero cuando cada uno de los dos había entendido una cosa diferente.

- Maquetación web: es la parte que personalmente más me ha costado puesto que no tenemos especial formación en esta rama, y me he dado cuenta de que una buena maquetación aporta mucha calidad y mucha seriedad a un proyecto.

- Diseño: esta parte ha sido la más complicada de llevar a cabo, por todo lo comentado anteriormente. Este proyecto trataba de realizar algo nuevo, y por tanto no teníamos apenas referencias para montar la estructura del sistema. A pesar de ser la más complicada, es la parte que más satisfactoria me ha resultado al conseguir los objetivos esperados.

- Documentación: esta ha sido la parte menos entretenida del proyecto y que supongo que a nadie le gusta hacer, aunque comprendo su importancia a la hora de exponer el producto ante un cliente.

Como conclusión final, me reitero en mi gran satisfacción con el resultado del proyecto. A nivel académico, me he dado cuenta de que todo lo aprendido a lo largo de estos 4 años no me ha formado como programador, sino como ingeniero. He aprendido a pensar, a cambiar la visión sobre las cosas, a tomar decisiones importantes e incluso a aprender tecnologías no estudiadas de forma autodidacta. He aprendido a que como ingeniero debo ser capaz de escuchar al cliente, entender su problema, y plantear una solución acorde al plazo y a las prestaciones existentes a pesar de que no exista nada parecido previamente.
Creo que tanto yo como mi compañero nos hemos compenetrado a la perfección y hemos sacado el mayor partido posible del proyecto.
Por último, me gustaría destacar que me he dado cuenta también, de que cuando avanzas en un proyecto durante tanto tiempo, y se hace de forma tan personal, la dedicación y las ganas de sacarlo adelante de la mejor forma posible son máximas, y por tanto, lamento decir que me hubiera gustado ampliar aún más la funcionalidad del mismo puesto que nunca me parecía suficiente. Supongo que este último apunte no indica más que me gusta lo que he estudiado y en lo que voy a dedicar el resto de mi vida.

7.2 Mejoras futuras

El avance futuro del proyecto podría ser prácticamente infinito. Las mejoras cercanas que se podrían plantear serían algunas de las siguientes:
- Mejora de la interfaz de usuario, haciéndola más usable, intuitiva y profesional.
- Añadir más tipos de gráficas para el procesamiento de datos.
- Permitir a la aplicación trabajar con otros formatos de archivos.
- Permitir la interacción entre los usuarios del sistema para compartir ficheros.
- Mejorar la eficiencia del código e implementar más casos de prueba para evitar posibles errores.
- Añadir más flexibilidad al usuario a la hora de manejar los datos una vez procesada toda la información.

Referencias

Valores separados por comas. (2019). Recuperado de https://es.wikipedia.org/wiki/Valores_separados_por_comas

Todo tipo de consultas. (2019). Recuperado de https://stackoverflow.com/

Librería para archivos Excel. (2019). Recuperado de https://poi.apache.org/

MongoDB. (2019). Recuperado de https://www.mongodb.com/

Librería Jasypt. (2019). Recuperado de http://www.jasypt.org/

Apache Maven. (2019). Recuperado de https://maven.apache.org/

Gráficas de datos. (2019). Recuperado de https://c3js.org/

HTML, CSS y JavaScript. (2019). Recuperado de https://www.w3schools.com/

Spring Boot. (2019). Recuperado de https://spring.io/projects/spring-boot

UML. (2019). Recuperado de https://www.nomagic.com/

Tutoriales. (2019). Recuperado de https://www.youtube.com/

Web para realizar ilustraciones. (2019). Recuperado de https://pixlr.com/

Apéndice A

Manual de Instalación

1. Menú principal

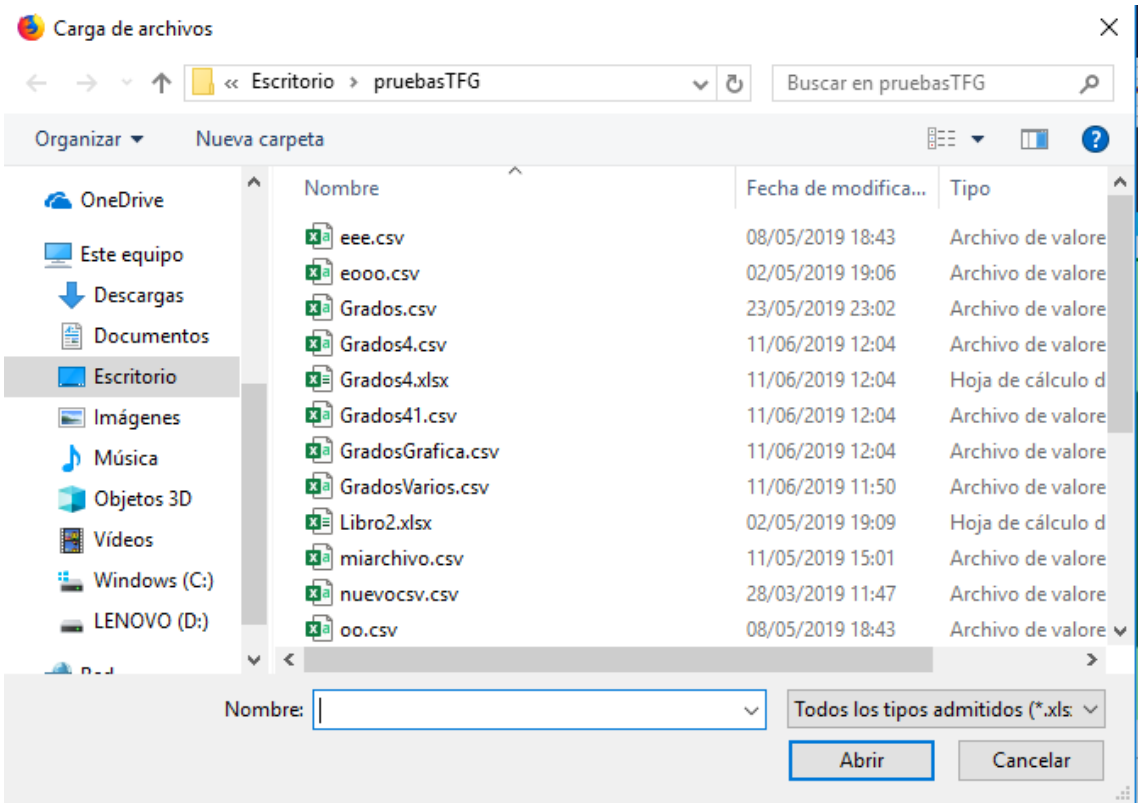
1.1 Opciones

Logout

Permite al usuario salir del sistema cerrando su cuenta.

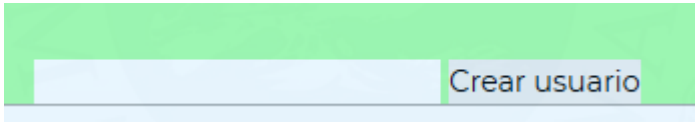
Subir ficheros

Para subir ficheros al sistema el usuario debe clicar en el icono y aparecerá el buscador de ficheros del sistema, seleccionando los que desea añadir.



Añadir usuarios al sistema

Opción disponible solamente para usuarios con permisos de Administrador.
Si se desea añadir un nuevo usuario al sistema, se debe introducir el nombre de usuario que tendrá y se pulsará el botón crear usuario .



Visualizar archivo del historial

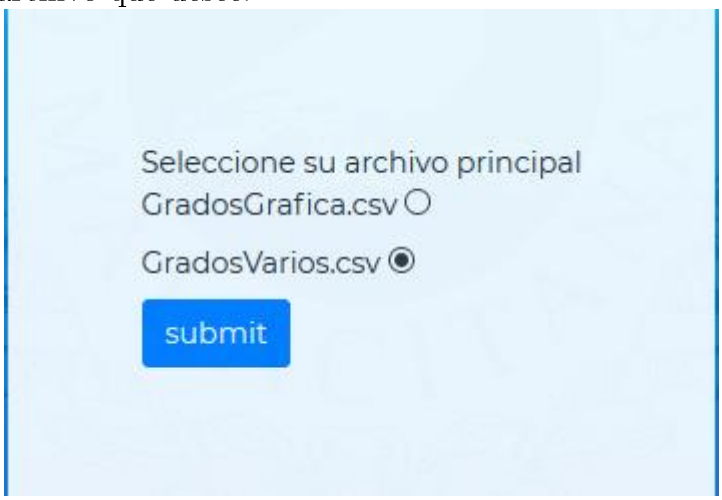
Al usuario le aparecerá una lista de ficheros que haya ido añadiendo al sistema con su trabajo.

Para visualizar uno de ellos, debe clicar encima del nombre.

2.Tratamiento de ficheros

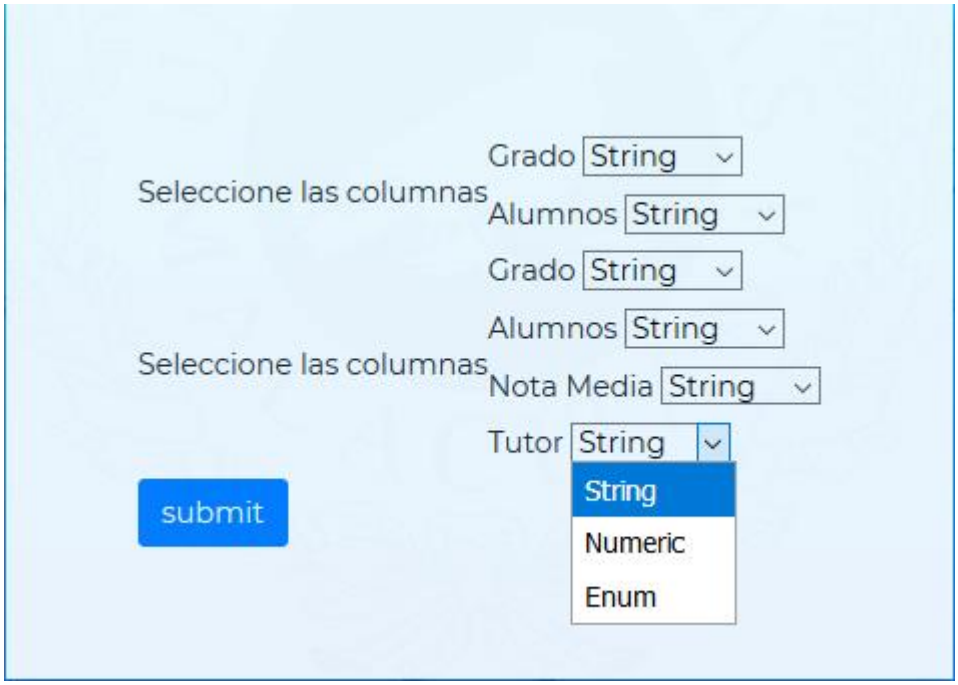
2.1 Elegir archivo principal

Una vez que el usuario haya añadido los ficheros que desea tratar, se encontrará en esta pantalla para elegir su archivo principal. Deberá seleccionar mediante los botones el archivo que desee.



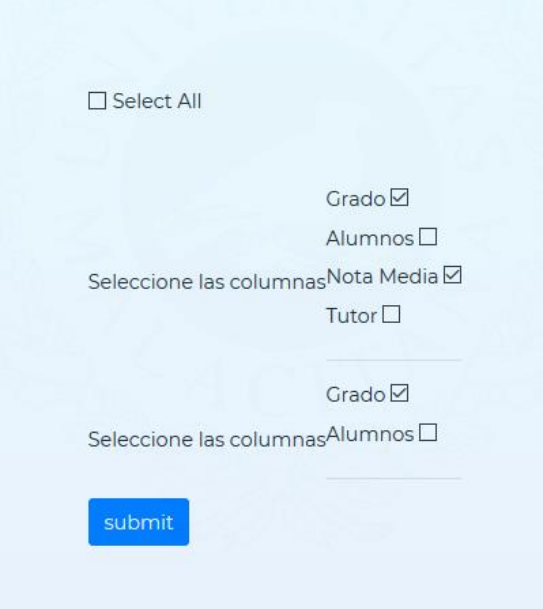
2.2 Elegir tipado

En esta pantalla el usuario elegirá el tipado de todas las columnas de sus archivos. Por defecto, vienen marcadas como tipo "String". En el caso de ser una columna que contenga datos numéricos deberá marcarla como "Numeric" y en caso de que sea una columna a tratar en una gráfica deberá ser marcada como "Enum".



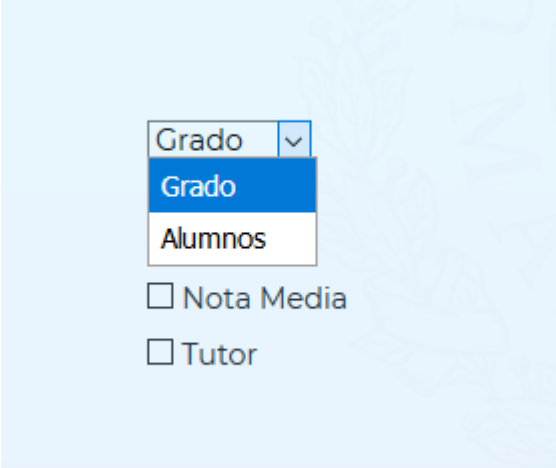
2.3 Elegir columnas

Puede ser que el usuario no desee tratar todas las columnas de un fichero. Después de haber elegido el tipado de su archivo, aparecerá la siguiente pantalla, donde el usuario seleccionará mediante unos checkboxs qué columnas desea tratar de cada archivo.

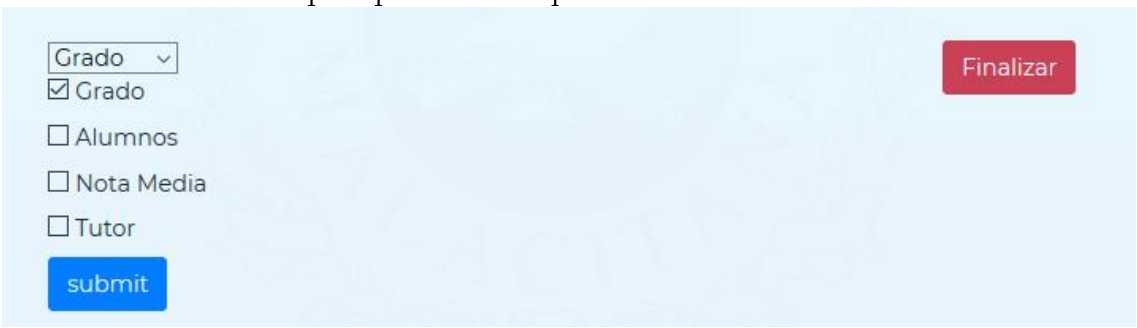


2.4 Relacionar datos

Una vez elegidas las columnas a tratar, llega la hora de relacionarlas en caso de que el usuario lo desee.
Para realizar esta acción, en primer lugar al usuario le aparecerá un seleccionable donde aparecerán todas las cabeceras del archivo principal.



Por cada cabecera que el usuario desee relacionar, deberá marcarla en el seleccionable, y acto después, deberá elegir con qué cabeceras de los otros archivos quiere relacionarlas. Para realizar esta acción, el usuario marcará las cabeceras mediante las casillas que aparezcan en pantalla.

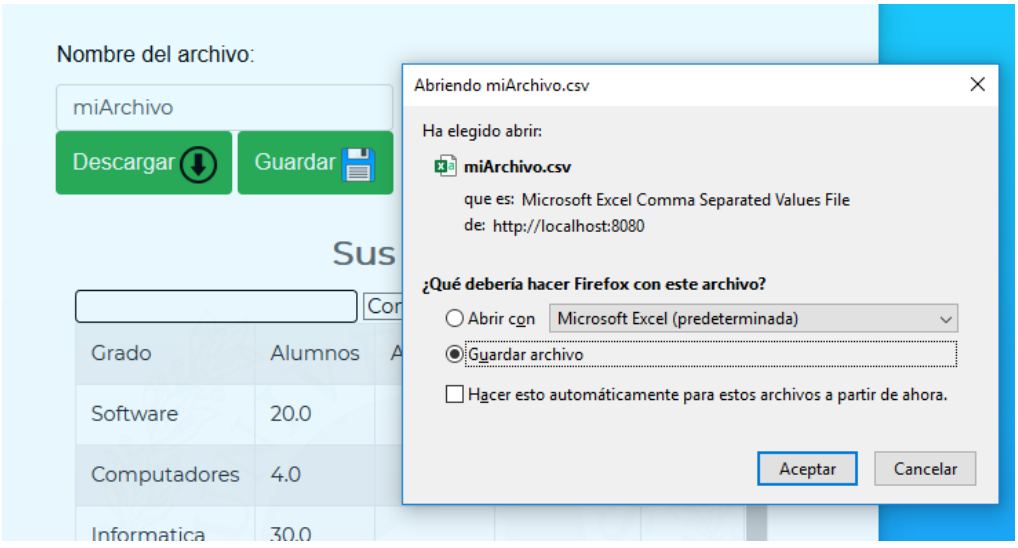


En caso de que el usuario quiera seguir relacionando otras cabeceras, deberá pulsar submit. En caso contrario, deberá pulsar Finalizar.

2.5 Descargar y/o guardar archivo

Cuando el usuario ha terminado de tratar el archivo, se le muestra el resultado por pantalla.
En caso de que el resultado sea satisfactorio, el usuario tendrá la opción de descargarlo en su propio ordenador como un archivo con formato csv, o simplemente guardarlo en la base de datos, para almacenarlo en la aplicación y poder acceder a él en un futuro.

En caso de que el usuario elija la opción Descargar, también se guardará automáticamente en la aplicación sin necesidad de pulsar el botón Guardar.

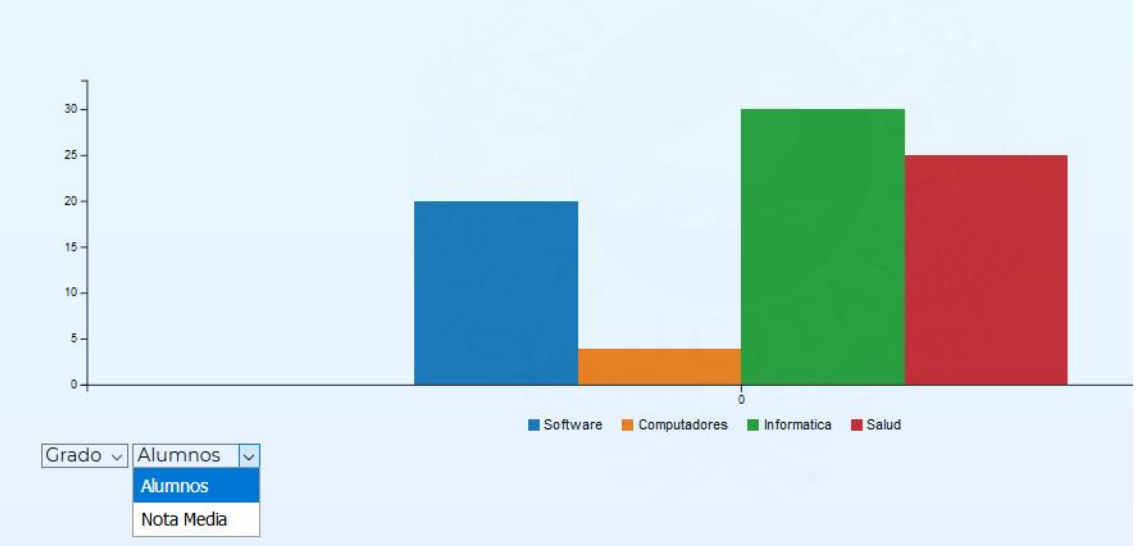


2.6 Visualizar gráficas de datos

Cuando un usuario ha guardado un archivo en el sistema, y desea acceder a su información, se le ofrece la opción de visualizar los datos mediante gráficas.
La aplicación actualmente dispone de 4 tipos de gráficas: de barras, circular, de puntos y de área.

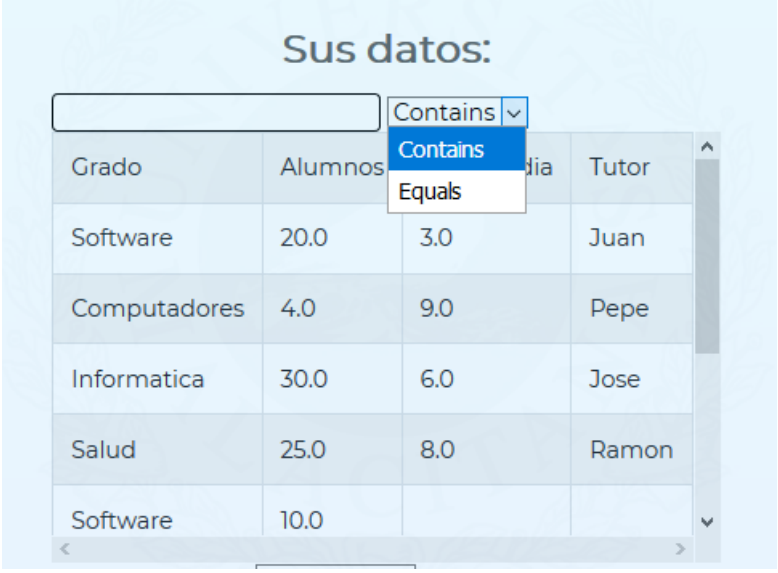
Para elegir entre una de estas opciones, el usuario debe marcarlo mediante los botones y pulsar submit.

Una vez dentro, el usuario será capaz de cambiar los datos dinámicamente por columnas.

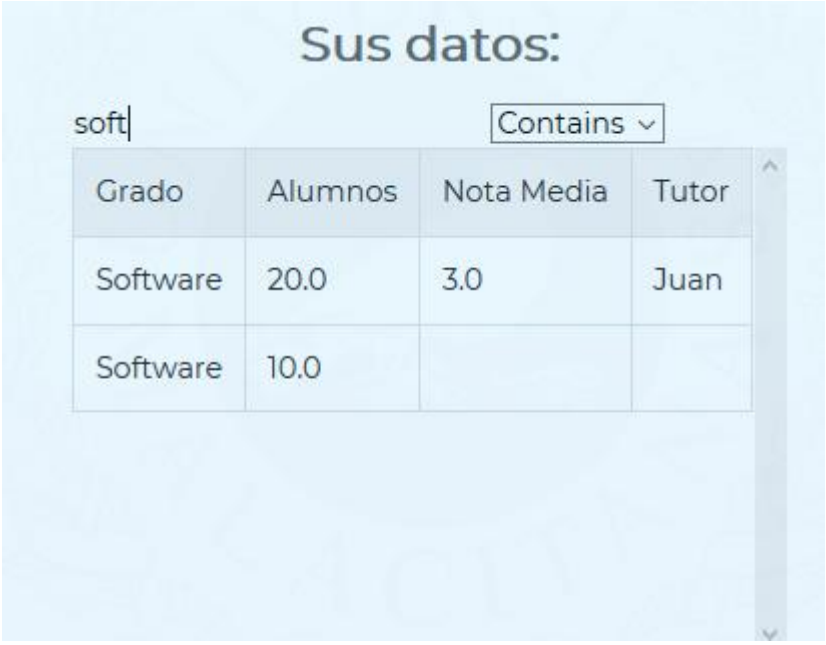


2.7 Filtrar datos

El usuario tendrá la opción de realizar búsquedas dentro de los datos de su fichero, para facilitar el tratamiento de los mismos y encontrar algo específico rápidamente.
Se han implementado dos opciones:
- Contains : al utilizar este filtro, se mostrarán por pantalla todas las filas que contengan la información que el usuario haya añadido
- Equals : al utilizar este filtro, se mostrarán por pantalla todas las filas que exactamente contengan la información que el usuario haya añadido.

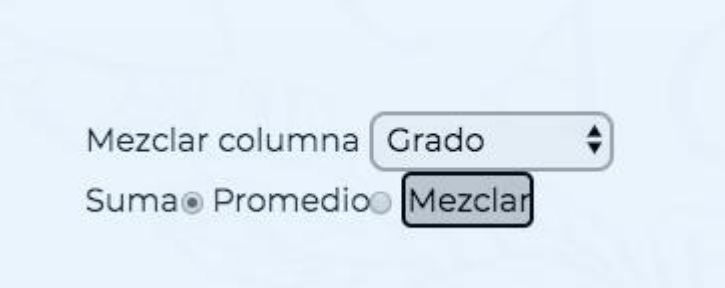


Así se mostrarían los datos después de filtrar con la opción "Contains" la palabra "soft".



2.8 Combinar datos

El usuario tendrá la opción de combinar los datos de su fichero por columnas en caso de que haya elementos repetidos, calculando el promedio o la suma de los datos numéricos asociados.
El usuario elegirá la columna a combinar mediante un seleccionable donde aparecerán todas las cabeceras. Además, deberá marcar también si desea calcular el promedio o la suma.
Cuando el usuario haya configurado la combinación deseada, pulsará el botón mezclar.



3.Registrarse en la aplicación

Para que un usuario se pueda registrar en la aplicación, primeramente un usuario con derechos de administrador debe haberlo habilitado para poder entrar al sistema.
Una vez que tiene estos permisos, para realizar su registro en el sistema, el usuario deberá seguir los siguientes pasos:
1º Se encontrará con el login principal de la aplicación. Aquí deberá ingresar su nombre de usuario (el asignado por el administrador correspondiente) y como contraseña, de nuevo su nombre de usuario. Acto seguido, pulsará el botón "Entrar". *(Ilustración A3.1).*
2º A continuación, aparecerá una nueva pantalla donde el sistema pedirá al usuario introducir su contraseña. Deberá hacerlo dos veces para evitar fallos, y el mismo sistema proporcionará un feedback informando al usuario sobre si ha escrito las mismas contraseñas o ha cometido algún error. Una vez introducidas de forma correcta, el usuario pulsará el botón "Registrar" y tendrá acceso a la aplicación. *(Ilustración A3.2).*

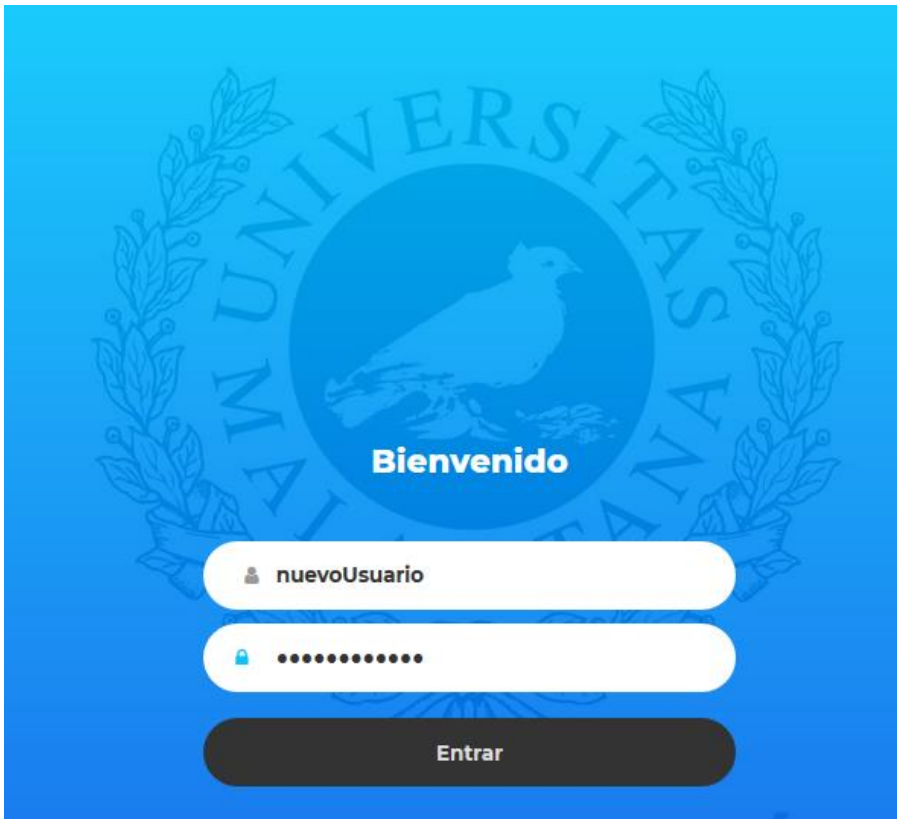


Ilustración A3.1 : Bienvenida usuario

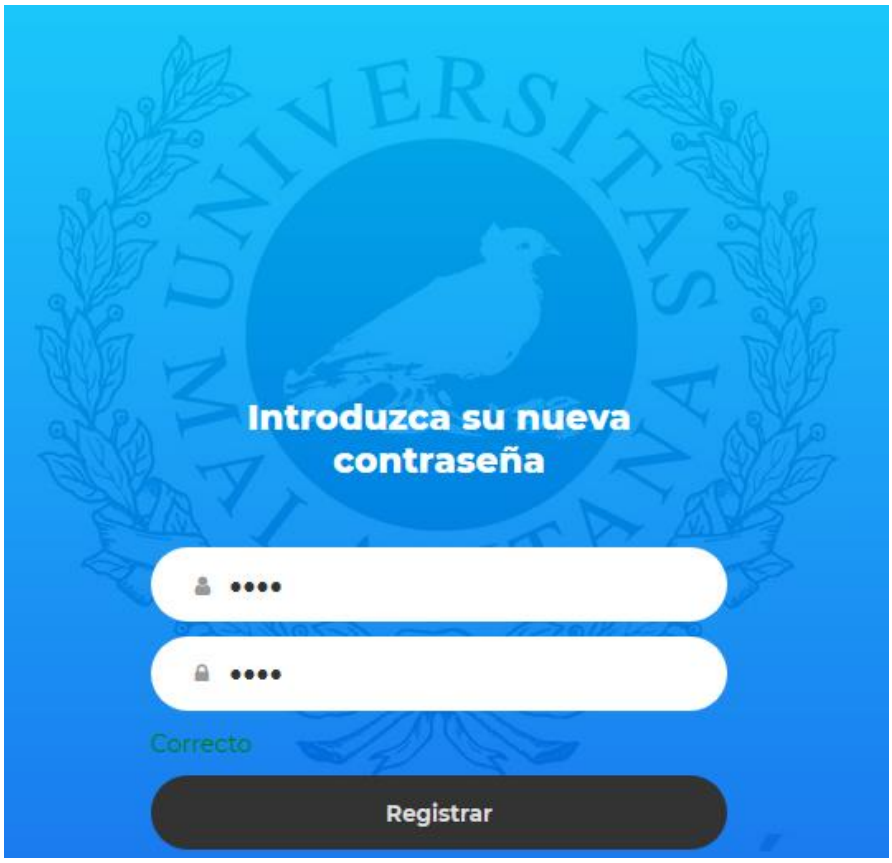


Ilustración A3.2 : Registrar usuario

Apéndice B

Manual de Instalación

El presente manual de instalación va destinado a usuarios tipo administrador que quieran instalar nuestra aplicación en un servidor.

1. Ejecutar la aplicación.

Para ejecutar nuestra aplicación es necesario instalar maven en nuestro equipo. En Mac OS podemos instalando mediante brew ejecutando el comando: "brew install maven".
Una vez instalado, entramos a la carpeta de nuestro proyecto y dentro hacemos una instalación limpia del proyecto. Para ello ejecutamos el comando: "mvn clean install". Al ejecutar este comando compilamos nuestra aplicación, y podemos ver que se crea un nuevo directorio llamado target. Este directorio es el que crea Spring por defecto como directorio de salida. El archivo que más nos interesa de esta carpeta es el llamado Procesamiento-CSV-1.0-SNAPSHOT.jar.

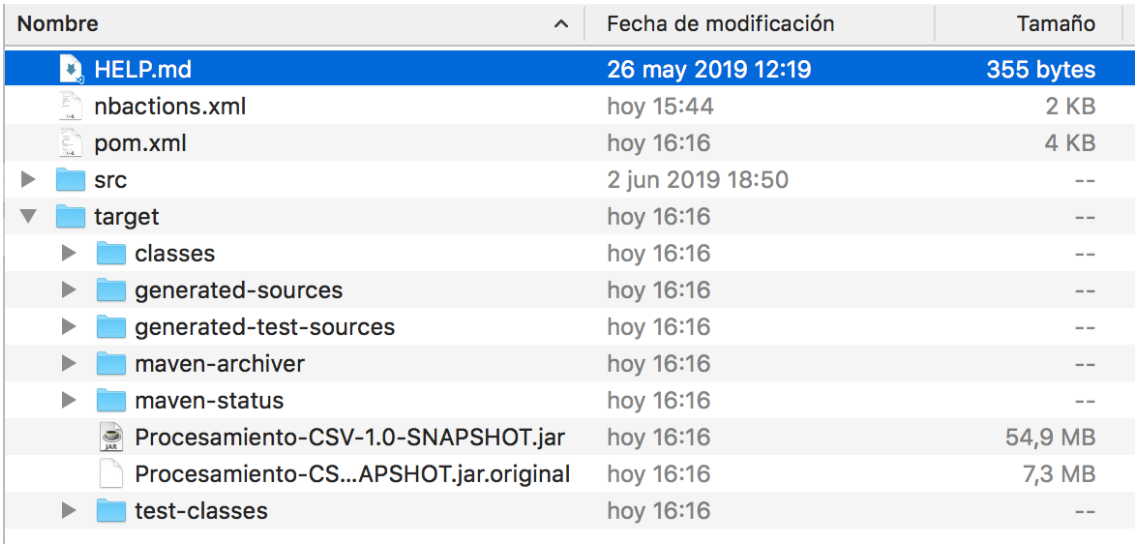


Ilustración MI 1: Directorio de la aplicación

Para ejecutar este archivo .jar podemos hacerlo haciendo clic sobre el o si estamos en un entorno sin interfaz gráfica podemos hacer uso del comando: "java -jar target/Procesamiento-CSV-1.0-SNAPSHOT.jar" que nos sirve tanto para unix, como para Mac Os, como para Windows. Una vez ejecutado el archivo de extensión .jar podremos acceder a la aplicación mediante la dirección

http://localhost:8080.

2. Conectar con MongoDB

Nuestra aplicación necesita de MongoDB para funcionar. Si no lo tenemos instalado, deberemos descargarlo de: https://www.mongodb.com.
La instalación se presenta bastante sencilla basta con ejecutar el ejecutable que nos descargamos y seguir las instrucciones del instalador. Tras la instalación nuestra aplicación se enlaza directamente con MongoDB. Es importante tener en cuenta que no sólo basta con instalar mongoDB, pues debemos iniciar un servidor mediante el comando "mongod" y crear una base de datos llamada "test".

3. Creación de Usuario Administrador

Para poder empezar a usar nuestra aplicación es necesario crear un usuario administrador, sin embargo, el usuario "admin" lo creamos por defecto en nuestra aplicación.
Si nunca has ingresado en la aplicación antes con este usuario, el sistema te pedirá que introduzcas una nueva contraseña. Con esto finalizaría la instalación y la aplicación estaría lista para ser usada.