

figure.Alph2.1



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería en Informática



**TFG del Grado en Ingeniería Informática**

**título del TFG  
Documentación Técnica**



Presentado por nombre alumno  
en Universidad de Burgos — 21 de junio de 2017  
Tutor: nombre tutor

---

# Índice general

---

<b>Índice general</b>	<b>I</b>
<b>Índice de figuras</b>	<b>III</b>
<b>Índice de tablas</b>	<b>IV</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>1</b>
A.1. Introducción . . . . .	1
A.2. Planificación temporal . . . . .	1
A.3. Estudio de viabilidad . . . . .	8
<b>Apéndice B Especificación de Requisitos</b>	<b>9</b>
B.1. Introducción . . . . .	9
B.2. Objetivos generales . . . . .	9
B.3. Catalogo de requisitos . . . . .	10
B.4. Especificación de requisitos . . . . .	11
<b>Apéndice C Especificación de diseño</b>	<b>12</b>
C.1. Introducción . . . . .	12
C.2. Diseño de datos . . . . .	12
C.3. Diseño procedimental . . . . .	13
C.4. Diseño arquitectónico . . . . .	13
<b>Apéndice D Documentación técnica de programación</b>	<b>14</b>
D.1. Introducción . . . . .	14
D.2. Estructura de directorios . . . . .	14
D.3. Manual del programador . . . . .	14
D.4. Compilación, instalación y ejecución del proyecto . . . . .	14
D.5. Pruebas del sistema . . . . .	14

<b>Apéndice E Documentación de usuario</b>	<b>15</b>
E.1. Introducción . . . . .	15
E.2. Requisitos de usuarios . . . . .	15
E.3. Instalación . . . . .	15
E.4. Manual del usuario . . . . .	15

---

# Índice de figuras

---

---

# Índice de tablas

---

---

# Plan de Proyecto Software

---

## A.1. Introducción

## A.2. Planificación temporal

Primer Sprint:

Este tramo inicial fue dedicado a realizar investigación sobre las herramientas que se usarían para desarrollar el producto final. Se buscó información sobre 3 elementos o partes distintas: • Herramienta para desarrollar la GUI: Concretamente, se debatió acerca de si sería más adecuado crear una aplicación web o una aplicación interna. Para tomar una decisión, se tuvieron en cuenta 4 herramientas. Por la parte de aplicación web, las candidatas fueron Jupyter y Flask; y por la parte de aplicación de escritorio, PyQt y Kivy. Tras practicar con ambas herramientas, se tomó la decisión de programar la GUI en un notebook de Jupyter para posteriormente publicarlo en un Dashboard. • Base de datos para almacenar la información: Como se debía almacenar numerosa y variada información sobre las noticias elegidas, se necesitaba de una estructura donde almacenar todos esos datos. La decisión más relevante en este apartado era si usar una base de datos relacional o no relacional, o dicho de otra manera, SQL o NoSQL. Las alternativas fueron SQLAlchemy (relacional), PyLucene (no relacional) y PyMongo (no relacional). • Herramienta para escribir la documentación: Aquí sólo se plantearon dos alternativas inicialmente: OpenOffice o Latex. Por cuestiones de formato y de preferencia de los miembros del proyecto, se optó por Latex, aprovechando la plantilla para Trabajos de Fin de Grado que ya había sido creada y que estaba a disposición de los alumnos.

Segundo Sprint:

La prioridad de este sprint era finalizar la investigación y búsqueda de

herramientas para desarrollar la aplicación, que concluyó con la elección de Pymongo como base de datos, y los notebook de Jupyter como herramienta de interfaz. Una vez elegidas las diferentes herramientas necesarias para el desarrollo del proyecto, el siguiente paso consistió en crear un pequeño programa que probara la librería PDFMiner y que contuviera algo de funcionalidad básica como contar la frecuencia de una palabra dentro de un documento. Aprovechando este script, se realizaron pruebas con un snippet llamado fileupload, que permite cargar ficheros en el código usando una GUI clásica en la que puedes navegar por el equipo buscando la ubicación del mencionado fichero. El resultado final era un programa que permite al usuario buscar un fichero en el equipo y, si este archivo tiene una extensión pdf, se permitirá buscar la frecuencia de las palabras que introduzca el propio usuario.

Tercer Sprint:

Habiendo finalizado la elección de todas las herramientas necesarias, y probado el funcionamiento básico para lectura de PDF subiendo archivos a mano, la siguiente prioridad fue crear un primer prototipo de web scrapping. Para lograr esto, se dividió la funcionalidad en dos etapas: • Un primer script que realiza la lectura web de noticias en diferentes medios (para este prototipo, se leían noticias sólo del diario “Público” y, más en concreto, de la sección de Igualdad), almacenando en la base de datos estructuras formadas por el titular, autor, fecha, texto de la noticia, procedencia y un link a la misma. Para recoger el texto html de las páginas web de los diarios, se usaron las librerías de url2lib y BeautifulSoup; la primera para hacer una llamada a la url indicada y obtener todo (o casi todo, en las próximas etapas explicaremos qué faltaba) el texto HTML de la página de cada noticia, y la segunda para poder manipular todo el HTML recibido previamente de forma más sencilla sin tener que hacer una división manual por etiquetas, simplemente indicamos al contenido de qué etiquetas queremos acceder.

• Un segundo script que, en función de una palabra clave introducida por el propio usuario, consulte en la base de datos y extraiga todas las noticias que contengan esa palabra clave, en un formato de tabla HTML y mostrando la información más importante de la noticia junto con el número de apariciones que hace esa palabra en el texto de las mismas. Para poder mostrar esto en la salida del notebook, necesitamos las librerías de pandas, ipywidgets y qgrid. Con la librería de pandas podemos organizar la información devuelta en la consulta de la base de datos a un dataframe, que puede ser convertido a una tabla html. Con los widgets, podemos mostrar controles básicos como botones, entradas de texto, listas desplegables, etc... sin necesidad de escribir a mano el código HTML. En esta versión, se usó la librería qgrid para transformar el dataframe en una tabla interactiva, que permitía reordenar el orden de las filas dando prioridad a unas columnas u otras. No obstante, no permitían formatear las celdas por separado, lo cual nos hizo descartarlo para posteriores versiones.

#### Cuarto Sprint:

Habiendo realizado una primera aproximación tanto a la lectura de PDF como de web, este sprint se centró en mejorar tres apartados: • En la parte de PDF, se mejoró el script, aunque a costa de retirar temporalmente la librería que permitía seleccionar un fichero de forma manual, para que recorriera y almacenara todos los archivos PDF, tanto del directorio que se pasaba como argumento como de todos los que colgaban de él. Una vez recorridos todos, se guardan las noticias en la base de datos, de forma similar a como se hacía con las noticias en la web pero cambiando el atributo de la procedencia. • En la parte de web scrapping, se actualizaron dos aspectos importantes: o Por una parte, usando Selenium además de BeautifulSoup se consiguió obtener los comentarios de las noticias, que en el caso de “Público”, se generaban con una API de terceros que impedía acceder a su HTML de forma normal como se hacía con el resto de la noticia. El método era algo pesado, ya que selenium necesita abrir en el ordenador que se esté ejecutando las pestañas a las que accede, pero como ya se ha mencionado, fue la alternativa que se encontró más idónea para poder acceder al HTML de los comentarios de las noticias.

Por otro lado, se cambió el formato de la tabla, dejando de usar qgrid para el DataFrame y mostrándola en un HTML normal, al que luego se aplicaba estilo manualmente usando un script de Javascript. Ésto es posible ya que uno de los widgets de Ipython/Jupyter permite introducir código HTML de forma manual, y es la vía que también utilizamos para introducir código JavaScript dentro del output del notebook. Con este cambio, tenemos flexibilidad total para manipular las celdas de la tabla para, por ejemplo, formatear como hiperenlaces las url de las noticias que se muestran en la tabla.

#### Quinto Sprint:

En esta etapa se decidió que en el muestreo de resultados, sería interesante incluir gráficos que acompañaran al resto de la información, para poder mostrar la mayor cantidad de datos posibles de diversas formas. Por tanto, una de las tareas fue elegir una librería de gráficos adecuada a nuestro contexto. Las candidatas iniciales fueron Bokeh y SeaBorn. Se probó Bokeh y, debido a unos problemas relacionados con la generación de archivos HTML, se decidió descartar. Por falta de horas durante esa semana, se trasladaron las pruebas con SeaBorn al siguiente sprint. La otra gran tarea a la que se dedicó tiempo durante este sprint fue a aumentar los medios a los que se realizaba web scrapping, creando un script para recorrer y almacenar las noticias de ElDiario. En este caso, los comentarios no se generaban a partir de una API de terceros, pero debido a la complejidad del código HTML obtenido, y visto necesario el uso de Selenium una vez más, se decidió separar la obtención de comentarios y aplazar esa parte al siguiente sprint.

#### Sexto Sprint:



La mayor parte de esta semana se dedicó a terminar tareas de la semana anterior que habían sido subestimadas en horas, y separadas en tareas entre este sprint y el anterior. En el apartado de los gráficos, se realizaron pruebas con la otra librería candidata, Seaborn, y por su comodidad y sencillez para mostrar gráficos simples (Como gráficos de barras para evaluar las menciones a una palabra por autor o día de la semana), se decidió que por el momento sería la alternativa elegida para acompañar en la interfaz. Respecto al web scrapping de ElDiario, se implementó el código necesario junto con el uso de Selenium adecuado para poder obtener los comentarios de las noticias de ese medio, que era la funcionalidad que no dio tiempo a terminar en la anterior semana. Aparte de estas tareas, se dedicó cierta cantidad de tiempo a investigar formas de publicar esta aplicación web en algún servidor. Para comenzar, se realizó cierta investigación sobre Docker, además de ver vídeos de la PyCon de Londres 2016, donde se trataba el tema de subir notebook a un servidor y poder visualizarlos como un dashboard. Por problemas de límite de tiempo, no se pudo avanzar mucho más este aspecto del proyecto.

#### Séptimo Sprint:

En este sprint se dedicó parte del tiempo a solucionar un problema recurrente en el que se mostraban sucesivamente todas las tablas de resultados de todas las palabras buscadas, provocando que la interfaz resultante se llenara de tablas innecesarias. De la misma manera, con los gráficos de Seaborn, que usa Matplotlib para funcionar, sucedía algo similar, provocando que todos los gráficos generados se fueran acumulando en una misma zona de dibujo que comparten por requisitos de la librería. Para solucionar esto hicieron falta dos medidas distintas. Para la parte de las tablas html, el problema se solucionó desde el código de Python, cambiando el planteamiento de la función que se ejecutaba al pulsar en el botón de búsqueda, y haciendo globales algunas variables necesarias para la generación de estas tablas. Para la parte de los gráficos, como este problema se estaba produciendo por limitaciones de la librería, hubo que buscar una solución más “rudimentaria” y controlar, por medio de una función JavaScript (introducida en un widget HTML), la visualización de la tabla requerida y la no visualización del resto de tablas obsoletas. El resto del tiempo de esa semana se dedicó a la investigación de la herramienta TextBlob, una opción muy interesante para realizar minería de datos de texto. En términos generales, es una herramienta que, a partir de los textos recibidos, puede traducirles, detectar idiomas, corregir palabras, dividir párrafos en frases, y frases en palabras, devolver etiquetas de las frases o palabras... etc. Aparte de toda esta funcionalidad, muy útil para casos como el de este proyecto, en el que se manejan textos, tiene sus propios clasificadores de datos. Para usar esta herramienta, se necesita descargar el NLTK (Natural Language ToolKit), que contiene todos los conjuntos de datos necesarios para poder llevar a cabo estas manipulaciones de texto y clasificaciones de los mismos. Debido a problemas con la descarga y tamaño de los archivos, se

cerró la tarea en ese punto, y se retomó en el siguiente sprint. Al término de esta semana, se realizó una reunión con el cliente para enseñar los progresos alcanzados hasta ese punto y tomar decisiones acerca de qué pasos tomar a continuación.

#### Octavo Sprint:

La tarea más importante realizada esa semana fue la de hacer otra aproximación al web scrapping basada en obtener los datos a partir de los RSS de los medios en los que, hasta el momento, se estaban leyendo desde las páginas web convencionales. El resultado fue un pequeño programa que te permitía seleccionar un medio de una lista de diarios y una palabra clave y, en tiempo real, realizar el web scrapping del RSS de ese medio y almacenarlo en la base de datos a la vez que se mostraba en una tabla los resultados obtenidos. Para ello se usó la librería feedparser. Este acercamiento era mucho más rápido que el web scrapping convencional, pero a cambio sacrifica la posibilidad de poder acceder a los comentarios de las noticias, y los casos de algunos medios, no todo el texto de la noticia es accesible a través del RSS. Como ha sido mencionado en el resumen del sprint anterior, debido a la falta de capacidad en la máquina virtual en la que se estaba trabajando, no se podía descargar todo el conjunto de datos de NLTK de forma correcta, lo cual se intentó solucionar inicialmente clonando la máquina virtual a una de almacenamiento dinámico y editando la capacidad de almacenamiento de las particiones de la misma. Esta solución no tuvo resultados positivos, por tanto, se reinstaló desde el principio una nueva máquina virtual que evitara estos problemas y que además corrigiera la instalación de algunas librerías que habían producido problemas previamente. Por último, se probó una nueva librería de gráficos que sustituyera a Seaborn, denominada Bqplot, la cual se asemejaba en funcionalidad a Bokeh, pero evitando esa necesidad de archivos HTML que nos habían llevado a descartar Bokeh en un primer momento. Las pruebas fueron muy positivas en gráficos de barras y lineales.

Noveno Sprint: La duración de este sprint fue el doble de lo normal (dos semanas en vez de una) debido a que coincidió con las vacaciones de Semana Santa. Por tanto, el día que debía haberse realizado la reunión habitual no era lectivo, y hubo que posponerlo. Por otro lado, como había suficientes tareas/horas de trabajo como para dividirlo en más de un sprint, se dio por finalizado el sprint a los 14 días aunque la siguiente reunión fuera después de 21. En lo relacionado a tareas relacionadas con sprints anteriores, se consiguió solucionar el problema por el cual no se aplicaban los scripts de JavaScript, aunque para ello hubo que buscar una forma alternativa de implementarles. Además, se consiguió finalizar con éxito la instalación y pruebas con TextBlob, llevando a cabo la instalación del NLTK de una manera alternativa que ocupaba mucho menos espacio en memoria. Una vez instalado y probado, se hizo un sencillo prototipo con un conjunto de datos de prueba minúsculo de

diferenciador de frases machistas y feministas, tanto con TextBlob como con SciKitLearn, aprovechando código de scripts ya existentes. También se mejoraron las operaciones de almacenamiento en base de datos, evitando introducir noticias duplicadas, para que el usuario final no se tenga que preocupar de filtrar las que estén duplicadas de las que no. En lo respectivo a la publicación del notebook como aplicación web, se realizaron instalaciones de varias herramientas de Jupyter Dashboards y de Docker, pero debido a impedimentos de configuración de la BIOS y del anfitrión, no se pudo avanzar más en la máquina virtual, y se decidió repetir el proceso en el anfitrión en el siguiente sprint. Por último, se añadió el diario “El Mundo” a los disponibles en el web scrapping por RSS, para ir aumentando el rango ideológico de las alternativas disponibles.

#### Décimo Sprint:

En este periodo se continuó la investigación y las pruebas de despliegue de aplicación de los notebook en una máquina de Docker, aunque en esta ocasión se realizaron en el Sistema Operativo anfitrión (Windows) en vez de en la máquina virtual, para comprobar si al cambiar el entorno conseguíamos solucionar los problemas técnicos sufridos durante la configuración de Docker.

Se realizaron además las primeras pruebas de clasificación de textos con un dataset real, compuesto por un grupo de noticias de diferentes medios de comunicación tratando el debate de los vientos de alquiler, etiquetadas a partes iguales como <sup>A</sup> Favorz <sup>En</sup> Contra” del tema que hablaban. Las pruebas fueron realizadas tanto en TextBlob, dividiéndolas en sentencias, como en SciKit-Learn, dividiéndolas en palabras.

También se dedicó parte de la semana a la documentación del trabajo realizado hasta la fecha.

#### Undécimo Sprint:

Tras los fracasos en la configuración sufridos con Docker, se investigó la alternativa de Cloud Foundry, plataforma open source basada en contenedores que da soporte a la subida de aplicaciones a la nube.

En lo respectivo a la clasificación del Dataset real, se continuaron las pruebas, en este caso probando distintos permutadores (Leave One Out, Leave P Out y ShuffleSplit), para hacer validación cruzada y separar las noticias que teníamos en conjuntos de entrenamiento y conjunto de prueba.

Además, se investigó la librería de Lime, que permite crear explicaciones detalladas de las predicciones, y se realizaron los primeros experimentos con la misma.

#### Duodécimo Sprint:

En este punto, se decidió dar más prioridad a crear una aplicación con funcionalidad completa que a seguir invirtiendo más tiempo en intentar con-

figurar un contenedor para poder subir los notebooks de Jupyter con formato de aplicación web.

Por ello, se comenzó el desarrollo de una aplicación con Flask que reuniera toda la funcionalidad implementada hasta el momento, que estaba repartida en varios archivos sin relación entre ellos, con objetivo de desplegarla en modo local en lugar de subirla a un servidor.

Se creó un esqueleto con la funcionalidad más básica de lectura de PDF, Web Scraping y persistencia en una base de datos NoSQL, sobre la que construir en las siguientes semanas.

Decimotercer Sprint:

Durante esta semana se continuó la construcción de la aplicación en Flask, como la implementación de las inserciones desde la web a la base de datos, el traslado del código de clasificación de texto de Jupyter a Flask y una mejora en la lectura de PDF.

Aparte de esto, se dedicó un tiempo a solucionar problemas más pequeños como correcciones en el uso de rutas literales, sustituir el uso de librerías obsoletas por sus alternativas vigentes, la inclusión de un fichero .txt con las librerías necesarias para ejecutar la aplicación, y completar la experimentación de Lime con los ejemplos de la documentación.

Decimocuarto Sprint:

Este sprint se centró en avanzar todo lo posible la funcionalidad relacionada con la clasificación de texto. Concretamente, se consiguió desde la librería de NLTK una lista de palabras irrelevantes en español a tener en cuenta para evitar ruido en los entrenamientos y predicciones.

En lo respectivo a la aplicación web, también se centraron esfuerzos en la vista de resultados de noticias y las posibilidades de etiquetado de las mismas, permitiendo que las noticias tuvieran varios datasets, que se pudiera añadir un dataset entero mediante un documento XML y se hizo una primera aproximación a un etiquetado manual, pero que sería sustituido por una alternativa más coherente en la siguiente semana.

Decimoquinto Sprint:

En esta semana se completó casi toda la funcionalidad restante en la tabla de resultados de noticias, incluyendo una barra de progreso que indicaba las probabilidades que tenía una noticia de pertenecer a una clase o a otra, la posibilidad de que el usuario realice un etiquetado manual a cada noticia, y la posibilidad de ver la explicación de la predicción en una pestaña aparte, acompañada de un gráfico para comprender mejor los resultados.

También se dedicó parte del tiempo a mejorar la coherencia de urls para separar distintas funcionalidades en distintas pestañas, para evitar sobrecargar

de complejidad las peticiones de los formularios, sobre todo en las pestañas de lectura de PDF y XML.

Decimosexto Sprint:

Teniendo gran parte de la funcionalidad implementada, en este Sprint se centró en mejorar el estilo gráfico de la interfaz de la aplicación. Para ello, se usó Bootstrap, primero experimentando con la versión para Flask y posteriormente se decidió permanecer con el modo de uso que tendría en un proyecto web normal, importando los archivos CSS y JS correspondientes. Aprovechando que había que utilizar JQuery para poder aplicar Bootstrap, se refactorizó el código existente de JavaScript en JQuery, reduciendo en un gran tamaño el mismo.

Además del estilo, también se actualizó el modo de lectura de los PDF, creando una estructura dentro de los archivos simple que facilitara la posterior lectura e inserción en base de datos, obligando a todos los documentos PDF que sigan la misma estructura para que puedan ser leídos.

También se modificó la vista de resultados de la explicación de Lime, aprovechando su característica de poder mostrar una vista de resultados propia exportada como HTML, sin tener que recurrir a librerías de gráficos de terceros.

### A.3. Estudio de viabilidad

#### Viabilidad económica

Para poder hacer un uso correcto de la aplicación, no hay que hacer ninguna inversión económica en software de ningún tipo, ya que todas las librerías que se han usado son open-source.

Más allá de lo comprendido por la propia herramienta, el único coste imprescindible sería el del propio equipo en el que se instale y configure. Aunque algunas funcionalidades tengan tiempos de carga más notables que otras, no debería ser impedimento para que la aplicación funcionara correctamente en equipos de diferentes prestaciones. Respecto al Sistema Operativo, como la aplicación se ejecuta en sistemas Linux, no es necesario ni realizar la compra de un Sistema Operativo de pago.

Cabe destacar que en el contexto de este proyecto la aplicación no está publicada en un servidor web, pero que en un futuro podría llevarse a cabo esta operación. De ser así, habría que evaluar diferentes alternativas de host compatibles y comprobar si alguna de estas alternativas son de pago o gratuitas.

#### Viabilidad legal

---

## Especificación de Requisitos

---

### B.1. Introducción

En esta sección se describirán las funciones para las que la herramienta fue desarrollada. Se hará una detallada definición tanto de objetivos como de sus requisitos (funcionales y no funcionales) y también se hará un estudio en profundidad de los casos de uso.

### B.2. Objetivos generales

Como ya se describió previamente en a memoria, hay un conjunto de objetivos en torno a los cuales se estructura el desarrollo del proyecto. A continuación se volverán a resumir los relacionados con la funcionalidad y las técnicas empleadas:

- El principal objetivo es crear una aplicación web en la que el usuario pueda almacenar noticias tanto por medio de web scraping como lectura de PDF.
- Se busca que el usuario pueda añadir etiquetas relacionadas con diferentes temas de debate a estas noticias, tanto manualmente como guardando una predicción realizada con aprendizaje automático.
- Otro objetivo es que el usuario pueda añadir temas de debate nuevos que contengan dos clases, para luego poder etiquetar estas noticias en función de estas nuevas clases.
- En caso de que el usuario ya tuviera un corpus preparado antes de comenzar a usar la aplicación, tiene la posibilidad de añadir un corpus mediante un archivo XML con todas las noticias que lo forman y sus etiquetas correspondientes.

- La aplicación debe ser responsive en los diferentes equipos en los que se use, y fácil de usar para nuevos usuarios.

### B.3. Catalogo de requisitos

#### Requisitos Funcionales

Estos son los requisitos funcionales que la herramienta debe cumplir:

- RF - 1: Los usuarios deben poder visualizar un conjunto de noticias a partir de la búsqueda por palabra clave.
- RF - 2: Los usuarios podrán etiquetar manualmente las noticias en función del tema de debate seleccionado.
  - RF - 2.1: Los usuarios podrán elegir de una lista de opciones el tema de debate mostrado en ese momento.
- RF - 3: Los usuarios pueden ver una explicación gráfica de la explicación para la predicción de la etiqueta de una noticia.
  - RF - 3.1: Los usuarios también podrán guardar esa predicción como la etiqueta final de la noticia.
- RF - 4: Los usuarios podrán guardar las noticias que se contuvieran en archivos PDF almacenados en una carpeta concreta del proyecto.
- RF - 5: Los usuarios podrán guardar nuevas noticias de los RSS de varios medios de comunicación españoles.
  - RF - 5.1: Los usuarios podrán elegir de una lista de opciones el medio de comunicación del que ver las nuevas noticias.
- RF - 6: Los usuarios podrán añadir un corpus de noticias entero, indicando el tema de debate y la etiqueta de cada noticia, a través de un archivo XML.
- RF - 7: Los usuarios podrán crear nuevos temas de debate, escribiendo el nombre del propio tema y el nombre de dos clases, que se usarán posteriormente de etiquetas para ese tema.

#### Requisitos No Funcionales

Por otro lado, estos son los requisitos no relacionados con la funcionalidad que se deben tener en cuenta:

- RNF - 1: La interfaz de las pestañas no debe impedir su correcto funcionamiento en pantallas de diferentes tamaños.
- RNF - 2: A pesar de que se produzcan picos de carga de trabajo, el funcionamiento general no debe producir tiempos de carga excesivos.
- RNF - 3: La herramienta debe poder ejecutarse en cualquier navegador (siempre que soporten librerías como Bootstrap o JQuery).

## B.4. Especificación de requisitos

A continuación, se describirán con detalle los casos de uso de los que se compone el proyecto.

### Diagrama de casos de uso

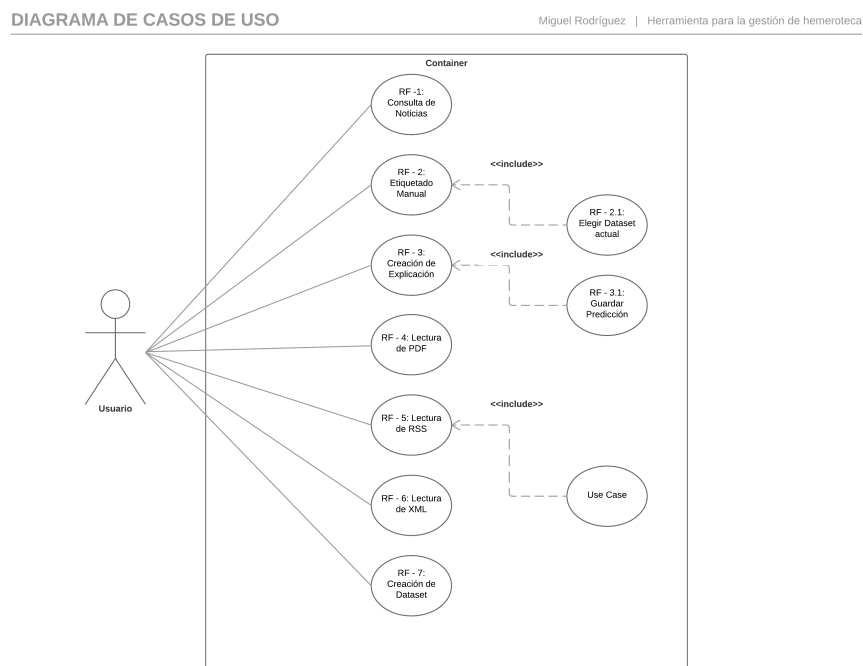


Figura B.1: Diagrama de casos de uso

### Descripción de casos de uso



## Especificación de diseño

---

### C.1. Introducción

En este apartado se analizará el diseño de la aplicación desde las diferentes perspectivas que componen el sistema. Se hablará del diseño de los datos, del flujo de ejecución de la aplicación y del diseño de la estructura de la misma.

### C.2. Diseño de datos

Como ya se ha mencionado en la memoria, la base de datos que se usa en la aplicación es NoSQL, lo cual quiere decir que no tendrá un sistema relacional de tablas como es costumbre.

También hay que destacar que, como la herramienta no está pensada para el uso abierto a cualquier usuario, si no a uno en concreto, no se consideró necesario hacer un sistema de registro de usuarios, con lo cual las colecciones/tablas habituales con la información de los usuarios aquí son inexistentes.

Sin embargo, si que existen dos colecciones (Se entiende por colección el equivalente a una tabla en SQL) en torno a las cuáles se estructura el almacenamiento de la información. A continuación describimos qué contienen cada una de ellas:

- **Noticias:** Es la colección que almacena los datos importantes de cada una de las noticias guardadas en la hemeroteca.
- **Datasets:** Es la colección que almacena los diferentes temas de debate sobre los que luego se podrá etiquetar y realizar predicciones con las noticias.

**C.3. Diseño procedimental**

**C.4. Diseño arquitectónico**

---

## **Documentación técnica de programación**

---

- D.1. Introducción
- D.2. Estructura de directorios
- D.3. Manual del programador
- D.4. Compilación, instalación y ejecución del proyecto
- D.5. Pruebas del sistema

## *Apéndice E*

---

# **Documentación de usuario**

---

- E.1. Introducción**
- E.2. Requisitos de usuarios**
- E.3. Instalación**
- E.4. Manual del usuario**