



Escuela
Politécnica
Superior

Título del Trabajo

Fin de Grado/Master



Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:

Nombre y Apellidos

Tutor/es:

Nombre y Apellidos

Marzo 2014



Universitat d'Alacant
Universidad de Alicante

Contrast

Subtítulo del proyecto

Autor

Adrián Tomás Vañó

Directores

Gustavo Candela Romero

Departamento de Lenguajes y Sistemas Informáticos

María Dolores Sáez Fernández

Departamento de Lenguajes y Sistemas Informáticos



GRADO EN INGENIERÍA MULTIMEDIA



Escuela
Politécnica
Superior



Universitat d'Alacant
Universidad de Alicante

ALICANTE, 21 de mayo de 2019

Resumen

Se debe incluir en la memoria un resumen de máximo 500 palabras, mejor se hace al final del trabajo con la visión global del mismo. Es obligatorio presentarlo para el tribunal. Se entrega en un fichero a parte y también lo podemos incluir en la memoria. Puede estar en las tres lenguas cas val y en

Preámbulo

Poner aquí un texto breve que debe incluir entre otras:

“las razones que han llevado a la realización del estudio, el tema, la finalidad y el alcance y también los agradecimientos por las ayudas, por ejemplo apoyo económico (becas y subvenciones) y las consultas y discusiones con los tutores y colegas de trabajo. [AENOR, 1997]”

*A mi esposa Marganit, y a mis hijos Ella Rose y Daniel Adams,
sin los cuales habría podido acabar este libro dos años antes*¹

¹Dedicatoria de Joseph J. Roman en “An Introduction to Algebraic Topology”

*Si consigo ver más lejos
es porque he conseguido auparme
a hombros de gigantes*

Isaac Newton.

Índice general

Resumen	v
1. Introducción	1
2. Modelo de negocio	3
2.1. Lean Canvas	3
3. Objetivos	5
3.1. Generales	5
3.2. Específicos	5
4. Marco Teórico	7
4.1. Servicios de contenido de medios digitales	7
4.1.1. Agregadores RSS	7
4.1.2. Servicios de seguimiento de prensa digital	8
4.2. Marco legal	8
4.2.1. Explotación y uso de contenido de medios digitales	8
4.2.2. Conflicto AEDE con Google Noticias	8
4.3. Tecnologías para el desarrollo	9
4.3.1. Lenguajes de programación	9
4.3.2. Tecnologías para la implementación del back-end	10
4.3.3. Tecnologías para la implementación del front-end	14
5. Metodología	17
5.1. Hitos e iteraciones	17
5.1.1. Hito 1	17
5.1.2. Hito 2	18
5.2. Herramientas de administración del proyecto	19
5.3. Herramientas de desarrollo	20
6. Análisis y especificación	23
6.1. Especificación requisitos funcionales	23
6.2. Casos de uso	23
7. Diseño	25
7.1. Arquitectura seleccionada	25
7.2. Tecnologías	25
7.3. Diagrama de clases	26

7.4. Mockups	28
7.4.1. Hito 1	28
7.4.2. Hito 2	29
8. Implementación	31
8.1. Hito 1	31
8.1.1. Iteración I	31
8.1.2. Iteración II	31
8.1.3. Iteración III	36
8.1.4. Iteración IV	39
8.2. Hito 2	41
8.2.1. Iteración VI	42
8.2.2. Iteración VII	43
9. Pruebas y validación	47
10. Resultados	49
11. Conclusiones	51
11.1. Conclusiones	51
11.2. Líneas de trabajo futuras	51
Bibliografía	53
A. Anexo A. Esquema de base de datos	55
B. Anexo B. Requisitos funcionales	57
C. Anexo C. Archivos Docker	65

Índice de figuras

2.1.	Lean Canvas de Contrast	4
4.1.	Lenguajes más populares en la encuesta de 2018 de Stack Overflow	9
11figure.4.2		
13figure.4.3		
5.1.	Espacio de trabajo para Contrast	19
5.2.	Tablero para Contrast	20
5.3.	Integración IFTT de Todoist con Trello	21
6.1.	Casos de uso.	23
7.1.	Stack tecnológico utilizado por Contrast.	25
7.2.	Diagrama de clases del backend.	27
7.3.	Diagrama de clases del frontend.	27
7.4.	Prototipo de interfaz para página de inicio y resultado búsqueda.	28
7.5.	Prototipo de interfaz para noticia leída en modo leer aparte.	28
7.6.	Diseño final de interfaz para página de inicio y resultado búsqueda.	29
7.7.	Diseño final de interfaz para noticia leída en modo leer aparte.	30
8.1.	Interfaz de usuario gráfica de Apache Solr	34
8.2.	Demostración de extracción de entidades en CoreNLP	34
8.3.	Demostración de extracción de entidades en Freeling	34
8.4.	Diseño final implementado de Contrast	44

Índice de tablas

Índice de Listados

8.1.	Universal Feed Parser	32
8.2.	Rome Tools	32
8.3.	rss-parser	33
8.4.	Declaración de un botón en Vuetify	35
8.5.	Declaración de un botón en Bootstrap	35
8.6.	Funciones de Rome para la consecución del esquema mínimo de datos . .	36
8.7.	Funciones de Rome para la extracción de los campos adicionales para el esquema de datos óptimo	36
8.8.	Eliminación de etiquetas HTML del cuerpo de la noticia	36
8.9.	Definición del modelo de datos	37
8.10.	Interfaz del repositorio Solr	38
8.11.	Configuración de conexión a cliente Solr	38
8.12.	Controlador de la API REST de Contrast	38
8.13.	Proxy inverso del frontend al backend	39
8.14.	Clase de reconocimiento de entidades	40
8.15.	Uso del extractor de entidades y convergencia de ambos datos	41
8.16.	Campo de datos personalizado	42
8.17.	Endpoints finales de obtención de noticias	43
8.18.	Clase final extractora de entidades	44
A.1.	Esquema de datos mínimo utilizado	55
A.2.	Esquema de datos óptimo utilizado	55
C.1.	Archivo docker-compose.yml	65
C.2.	Dockerfile de Solr	66
C.3.	Dockerfile de Vue.js	66
C.4.	Dockerfile de Java	66

1. Introducción

En la introducción es donde se hará énfasis a la importancia de la temática, su vigencia y actualidad; se planteará el problema a investigar, así como el propósito o finalidad de la investigación, la aportación que supone este TFG al dominio/sector que se ha investigado. Debemos explicar el contexto donde se ubicará el trabajo.

No hay una longitud estimada, pero se debe tener en cuenta que es lo primero que lee una persona que debe evaluar el trabajo.

Se pueden establecer secciones si así lo requiere la memoria.

2. Modelo de negocio

2.1. Lean Canvas

Esta sección es opcional y depende de la tipología del trabajo a realizar. El siguiente apartado forma parte del estudio de viabilidad del proyecto, en él se debe reflejar y justificar el porqué surge el trabajo. Es importante pensar y plasmar en el trabajo cómo nuestro proyecto va a aporta valor a los clientes, conocer sus necesidades o problemas y cómo les damos solución y cómo estos clientes van a pagar por ese valor que reciben. Además, es importante conocer el tipo de mercado en el que operamos y qué otros factores influyen en él. Se analizan qué alternativas existen hoy en día en ese sector que solucionen total o parcialmente el problema y que afecten a los cliente o bien se ha detectado que no existe una solución (aunque sea comercial) a dicho problema.

Importante detectar problema o necesidad, a quién afecta, cómo se resuelve y qué se va a aportar, es decir cómo se va a solucionar (TFG).

Se pueden incluir análisis de DAFO y análisis de los riesgos

PROBLEM	SOLUTION	UNIQUE VALUE PROPOSITION
Dificultad para leer noticias de distintos periódicos en un mismo lugar Dificultad para contrastar información entre periódicos El seguimiento de medios para usos no-comerciales es caro	Agregador de prensa digital de diferentes fuentes Visor simultáneo de noticias Seguimiento de medios básico, agrupando noticias según temática	Visor para lectura simultánea de prensa digital
EXISTING ALTERNATIVES	KEY METRICS	HIGH-LEVEL CONCEPT
Agregadores de medios que permiten la lectura de noticias Para contrastar la información hay que acceder a cada noticia por separado El seguimiento de medios se hace a través de empresas	Búsquedas Accesos a la aplicación	Comparador de opinión en prensa
COST STRUCTURE		REVENUE
Alojamiento en Heroku Difusión Horas de desarrollo		Opción Pre realizar rec Venta de a

Figura 2.1.: Lean Canvas de Contrast

3. Objetivos

3.1. Generales

El objetivo general del proyecto es crear un lector de prensa digital capaz de relacionar noticias según su contenido, de tal manera que, partiendo de una noticia determinada, se realice una selección de noticias ordenadas por similitud.

3.2. Específicos

A continuación se detallan los objetivos específicos que parten del objetivo general. Se han desgranado en ítems independientes entre sí en la mayor medida posible, de tal manera que su consecución sea calificable.

1. Construir una API REST que actúe como recolector de información del motor de búsqueda seleccionado. Dicha API debe implementar un recuperador y analizador de contenido RSS que sea utilizado para obtener los medios de los que se vale la aplicación.
2. Utilizar un motor de búsqueda para indexar las noticias obtenidas mediante RSS. El motor debe tener un esquema de datos específico.
3. Utilizar una herramienta de procesamiento de lenguaje natural que proporcione los extractores necesarios para obtener información relevante que permita determinar la similitud entre dos o más noticias.
4. Diseñar un visor de noticias que permita la lectura de estas a través de su comunicación con la API. Dicho visor debe implementar diferentes opciones para el filtrado de las noticias, así como tener una interfaz que facilite la usabilidad.

4. Marco Teórico

El siguiente capítulo trata el estado de las tecnologías existentes que ofrecen una servicio similar al que plantea Contrast en alguna de sus funcionalidades. Por otra parte, se trata el contexto en el que se engloba la aplicación a desarrollar.

4.1. Servicios de contenido de medios digitales

En este apartado se detallan dos posibles conjuntos de servicios sobre los que Contrast comparte cierta similitud en cuanto a funcionalidad. Los servicios citados están disponibles en internet.

4.1.1. Agregadores RSS

Existe una gran cantidad de aplicaciones con diferentes características y funcionalidades, aunque similares en uso. Dichos servicios permiten la búsqueda de fuentes de contenido para su lectura y la suscripción a temas de interés, que a su vez suscribe a varias fuentes relacionadas con el tema.

- **Feedly**¹ es el lector de contenido más popular actualmente. Cuenta con una interfaz clara y simple y permite consumir contenido en diversos formatos: vídeo, noticias de periódicos, tweets, publicaciones de blogs, alertas Google Keywords y por supuesto, RSS. Posee herramientas de integración con servicios de terceros como Evernote o Trello y permite crear tableros compartidos con otros usuarios.
- **Flipboard**² ha sido el agregador de noticias más utilizado durante mucho tiempo y una red social en sí. Posee integraciones con las principales redes sociales para leer contenido de ellas y compartir noticias en las redes desde la aplicación. Tiene un diseño cuidado que asemeja el uso de un magazín físico en cuanto a estructura de las secciones y el deslizamiento de hojas.
- **Inoreader**³ se define entre los usuarios como la alternativa a los anteriores cuando hay algún aspecto entre ellos que no termina de encajar con el usuario. Este lector de contenido posee herramientas para etiquetar y organizar automáticamente la información entrante en busca de ahorro de tiempo por parte del usuario.

¹<https://feedly.com/>

²<https://flipboard.com/>

³<https://www.inoreader.com/>

4.1.2. Servicios de seguimiento de prensa digital

El seguimiento de medios consiste en la detección de toda información publicada sobre un tema concreto. Estos datos se analizan en pos de categorizarlos y mejorar la distribución de los mismos. Los clientes que contratan estos servicios buscan mejorar su presencia digital al mismo tiempo que ser notificados de cualquier mención a su empresa y a su reputación.

- **Pressclipping⁴:** esta empresa ofrece análisis de medios y evaluación de noticias, además del propio seguimiento de medios. Cuenta además con soluciones para la gestión de las relaciones públicas y del marketing estratégico para medir con precisión el alcance y reputación de una marca.
- **Puntonews⁵:** servicio que permite buscar noticias tanto de periódico, radio como de televisión y exportarlas a PDF.
- **MyNews⁶:** hemeroteca digital desde 1996. Dispone de seguimiento de formatos audiovisuales y redes sociales, al igual que análisis de presencia y reputación de marca.

4.2. Marco legal

4.2.1. Explotación y uso de contenido de medios digitales

4.2.2. Conflicto AEDE con Google Noticias

REHACER Google Noticias es un agregador y buscador de noticias de medios de prensa digital que cuenta con más de 70 ediciones en 35 idiomas y que genera más de 10.000 millones de clics al mes. Esta herramienta, nacida en 2002, no está disponible en España desde hace unos años.

El 5 de noviembre de 2014, el Partido Popular aprobó la nueva ley de propiedad intelectual (<https://www.boe.es/boe/dias/2014/11/05/pdfs/BOE-A-2014-11404.pdf>), causando que Google se viese obligado a pagar una tasa exigida por la Asociación de Editores de Diarios Españoles (AEDE). Dicho de otra manera, ahora tenía que pagar por mostrar cualquier fragmento del contenido de los editores españoles.

Cabe destacar que Google no tenía un modelo de negocio con este servicio, pues no mostraba anuncios, tan solo enlazaba los titulares con la noticia en el medio digital.

La ley aprobada entraba en vigor el 1 de enero de 2015 y el 16 de diciembre Google anunciaba que retiraba el servicio de noticias en España. Los editores contrastaron días después que la plataforma de Google les aportaba tráfico y por lo tanto ingresos por anuncios, por ello pidieron al gobierno y a la Unión Europea que interviniieran, abogando que el gigante tenía la posibilidad de negociar la tasa que debía pagar. Google fue tajante con su rehuso.

⁴<https://www.pressclipping.com/>

⁵<https://www.puntonews.com/es>

⁶<https://www.mynews.es/>

Actualmente la ley sigue en vigor.

4.3. Tecnologías para el desarrollo

En esta sección se detallan las tecnologías sobre las que puede servirse el proyecto para su desarrollo. Por un lado se abarcan los lenguajes de programación, por otro las herramientas pertenecientes al back-end y por último las que atan al front-end.

4.3.1. Lenguajes de programación

Se han analizado los lenguajes más populares en la última encuesta publicada en Stack Overflow,⁷ que permiten tanto el desarrollo front-end como back-end, aunque se elijan opciones distintas para cada apartado.



Figura 4.1.: Lenguajes más populares en la encuesta de 2018 de Stack Overflow

- **Javascript**⁸ fue el primer lenguaje de scripting para web. Es uno de los lenguajes más usados en web tanto para el back-end como para el front-end, integrado en la gran cantidad de frameworks que se han construido sobre él. Con la salida de ECMAScript 6 se han solucionado muchos problemas que acarreaba el uso de Javascript y se han añadido nuevas funcionalidades que lo convierten en aún más interesante. Tiene una de las bases de usuarios más sólidas entre todos los lenguajes.

⁷<https://insights.stackoverflow.com/survey/2018/>

⁸<https://www.javascript.com/>

- **Java⁹** es un lenguaje con madurez y con muchos proyectos construidos sobre él que dependen de la instalación de Java en la máquina local. Se caracteriza por la solidez, confiabilidad, extensibilidad y rapidez. Existen multitud de frameworks y plugins que operan sobre Java y la documentación y comunidad es más que extensa. A este razonamiento se le suma el hecho de que es el lenguaje para el desarrollo en Android.
- **Python¹⁰** es un lenguaje en alza desde los últimos años. Está diseñado partiendo de la intuitividad y del aproximamiento al lenguaje humano. La popularidad del lenguaje se debe al crecimiento del big data y del internet de las cosas, ya que ofrece algunas herramientas para el prototipado y las analíticas. Es un lenguaje ideal para principiantes debido a su flexibilidad.

<https://www.quora.com/Which-language-has-the-best-future-prospects-Python-Java-or-JavaScript> <https://stackshare.io/stackups/java-vs-javascript-vs-python>

4.3.2. Tecnologías para la implementación del back-end

El siguiente apartado contiene un sumario de tecnologías que facilitan la implementación y puesta en funcionamiento del back-end del proyecto.

Analizador RSS

Un analizador RSS es una herramienta que recolecta contenido RSS de distintas fuentes y lo traduce en una clase u objeto sobre el que se puede interactuar para obtener información específica. En este caso, se ha analizado la herramienta más popular para este cometido en cada uno de los lenguajes citados anteriormente.

- **rss-parser¹¹** es una librería mínima para convertir contenido RSS XML en objetos Javascript. Se compone de una función asíncrona que utiliza en un capa inferior un conversor de XML a objeto Javascript llamado `xml2js`¹².
- **Universal Feed Parser¹³** es un módulo Python para descargar y parsear feeds. Es un plugin simple, autocontenido en un único fichero y con una única función primaria para parsear. Tiene una documentación extensa y funciones adicionales para sanitizar y parsear otros elementos contenidos en el feed. Trata también con el formato Atom.
- **Rome¹⁴** es un framework de Java de código abierto para tratar con RSS y Atom. Incluye analizadores, generadores y conversores entre ambos formatos. Los analizadores, el objeto de interés de Contrast, devuelve generalmente un objeto genérico

⁹<https://www.oracle.com/technetwork/java/javase/overview/index.html>

¹⁰<https://www.python.org/>

¹¹<https://www.npmjs.com/package/rss-parser>

¹²<https://www.npmjs.com/package/xml2js>

¹³<https://pythonhosted.org/feedparser/>

¹⁴<https://rometools.github.io/rome/>

que permite trabajar sin importar el formato de entrada o salida. Está compuesto por distintos módulos según se realice una acción concreta u otra. Posee una documentación sólida y es ampliamente utilizado para este fin.

Motores de búsqueda e indexadores

Los sistemas de recuperación de información con índice invertido,¹⁵ son sistemas caracterizados por tener una estructura de datos capaz de manejar grandes volúmenes de información orientados a texto. Esta funcionalidad está presente en los motores de búsqueda que se han investigado y que además son los tres más populares.

18 systems in ranking, December 2018										
Rank			DBMS	Database Model	Score			Dec 2018	Nov 2018	Dec 2017
Dec 2018	Nov 2018	Dec 2017			Dec	Nov	Dec			
1.	1.	1.	Elasticsearch	Search engine	144.70	+1.24	+24.92			
2.	2.	↑ 3.	Splunk	Search engine	82.18	+1.81	+18.39			
3.	3.	↓ 2.	Solr	Search engine	61.35	+0.47	-4.95			
4.	4.	4.	MarkLogic	Multi-model	14.28	+0.81	+3.13			
5.	5.	5.	Sphinx	Search engine	7.82	+0.46	+1.79			
6.	6.	6.	Microsoft Azure Search	Search engine	5.67	+0.36	+1.56			
7.	7.	7.	Algolia	Search engine	3.62	-0.13	+0.57			
8.	8.	↑ 9.	Google Search Appliance	Search engine	2.93	+0.03	+0.20			
9.	9.	↓ 8.	Amazon CloudSearch	Search engine	2.64	-0.11	-0.09			
10.	10.	10.	Xapian	Search engine	0.60	-0.08	-0.02			
11.	11.	11.	CrateDB	Multi-model	0.44	-0.06	-0.15			
12.	12.	12.	SearchBlox	Search engine	0.24	0.00	+0.01			
13.	13.	↑ 14.	searchxml	Multi-model	0.09	-0.01	+0.09			
14.	14.	↓ 13.	DBSight	Search engine	0.06	+0.00	+0.05			
15.	15.	↓ 14.	Manticore Search	Search engine	0.04	-0.01	+0.04			
16.			FinchDB	Multi-model	0.03					
17.	↓ 16.	↓ 14.	Exorbyte	Search engine	0.01	+0.01	+0.01			
18.	↓ 16.	↓ 14.	Indica	Search engine	0.00	±0.00	±0.00			

Figura 4.2.: Motores de búsqueda más populares en db-engines¹⁶

- **Apache Solr**¹⁷ está construido sobre **Apache Lucene**¹⁸. **Lucene** es una potente librería de recuperación de información. Se encarga de construir el índice invertido mencionado anteriormente, una estructura especializada en emparejar documentos de texto con términos de consulta. **Lucene**, al igual que otros motores de esta clase, se distingue por la escalabilidad, el rápido despliegue, el manejo de grandes volúmenes de datos, la optimización orientada a búsqueda y esta a su vez, en textos. **Solr** por tanto es un servidor open source construido con la librería de **Lucene**, que incluye en una interfaz web de administración, APIs en distintos

¹⁵<http://www.sisorbe.com/ExamenFinalRI/invertidos.html>

¹⁷<https://lucene.apache.org/solr/>

¹⁸<https://lucene.apache.org/>

lenguajes para la realización de consultas y una serie de mejoras funcionales sobre las características base de **Lucene**.

- **Elasticsearch**¹⁹, también construído sobre **Lucene**, ha basado su modelo en una API REST Web al ser un motor de búsqueda más reciente y que por ello, ha ganado popularidad desde la última mitad de década. **Elasticsearch** tiene una destacable ejecución en las consultas analíticas y es apreciado por ser muy intuitivo, especialmente entre desarrolladores que se inician en este tipo de tecnologías. Esta tecnología es muy valorada por su escalabilidad.
- **Splunk**²⁰ es un motor de búsqueda y plataforma orientado a las analíticas para el Big Data, es por ello que se ha desestimado su análisis.

Herramientas de procesamiento de lenguaje natural

Las tecnologías de procesamiento de lenguaje natural son un conjunto de herramientas lingüísticas, que permiten extraer información relativa a características léxicas, morfo-sintácticas y semánticas de un determinado texto. Existen muchas propuestas en este ámbito, entre ellas destacan las siguientes.

- **CoreNLP**²¹ es un proyecto de la Universidad de Stanford que provee una serie de herramientas de análisis de lenguaje humano. Proporciona instrumentos para un amplio análisis gramático, además de extractores muy diversos. CoreNLP posee una documentación extensa ya que es ampliamente utilizado, complementándose además con actualizaciones frecuentes que mejoran el producto. Posee soporte oficial para seis idiomas (entre ellos el español), aunque no están disponibles todas las funciones, como se puede observar en la figura 4.3 de la página 13. La integración con otros proyectos se realiza a través de una serie de APIs en distintos lenguajes. Esta herramienta es utilizada por la Biblioteca Virtual Miguel de Cervantes²², concretamente el analizador sintáctico.
- **OpenNLP**²⁴ es un proyecto Apache, consistente en una librería de aprendizaje automático cuyo objetivo es el procesamiento del lenguaje humano en texto. Posee soporte para las tareas NLP más comunes. No tiene soporte al uso de distintos lenguajes, pero posee una serie de modelos entrenados en diferentes idiomas para propósitos determinados. En el caso del español encontramos cuatro modelos entrenados en reconocimiento de entidades: personas, organizaciones, localizaciones y misceláneo. Cualquier extractor de interés que no esté en los modelos tendría que ser entrenado para su propósito.

¹⁹<https://www.elastic.co/es/>

²⁰<https://www.splunk.com/>

²¹<https://stanfordnlp.github.io/CoreNLP/>

²²<http://data.cervantesvirtual.com/analizador-sintactico-automatico>

²⁴<https://opennlp.apache.org/>

ANNOTATOR	AR	ZH	EN	FR	DE	ES
Tokenize / Segment	✓	✓	✓	✓		✓
Sentence Split	✓	✓	✓	✓	✓	✓
Part of Speech	✓	✓	✓	✓	✓	✓
Lemma			✓			
Named Entities			✓	✓	✓	✓
Constituency Parsing	✓	✓	✓	✓	✓	✓
Dependency Parsing	✓	✓	✓	✓	✓	✓
Sentiment Analysis			✓			
Mention Detection	✓		✓			
Coreference			✓	✓		
Open IE			✓			

Figura 4.3.: Soporte oficial de idiomas en CoreNLP²³

- **Freeling**²⁵ es un proyecto de la Universitat Politènica de Catalunya. Se trata de una librería con una serie de herramientas open source para el análisis de lenguaje humano. Está escrita en C++ y posee una extensa variedad de utilidades, entre ellas las de interés para el proyecto. Tiene soporte para muchos idiomas y dialectos españoles, pero carece de una comunidad sólida que utilice la herramienta de manera asidua, al igual que una documentación no tan extensa como las propuestas anteriores.

Frameworks de desarrollo

Para esta sección se ha detallado un framework por cada lenguaje de programación que se ha propuesto previamente. Esta selección de frameworks ha sido realizada bajo criterios de popularidad, mantenimiento, calidad en la documentación, extensibilidad e integración con el resto de herramientas propuestas.

- **Spring Boot**²⁶ es un framework basado en Spring para Java. Abstira el proceso

²⁵<http://nlp.lsi.upc.edu/freeling/node/1>

²⁶<https://spring.io/>

de elección de dependencias y despliegue del proyecto, realizando en cambio un proceso sencillo y automatizado. El objetivo principal de esta herramienta es proporcionar un conjunto de herramientas para construir rápidamente aplicaciones en el potente entorno Spring. Se trata de un framework maduro, con resultados profesionales e integraciones con otros productos de Spring que añaden más funcionalidades. Cabe destacar que posee una capa de abstracción²⁷ para el motor de búsqueda Solr, al igual que otra para Elasticsearch²⁸, aunque la primera está más consolidada.

- **Express.js**²⁹ es un framework rápido, minimalista y flexible para el entorno de ejecución Node.js para Javascript. Proporciona un conjunto sólido de características para realizar APIs a través de métodos y utilidades HTTP y middleware que garantizan una creación rápida y sencilla.
- **Django**³⁰ es un framework full-stack para Python que facilita rapidez y claridad, ya que se hace cargo de muchos conceptos que conlleva el desarrollo web. Es una herramienta altamente escalable debido a su flexibilidad y proporciona ayuda a los desarrolladores para evitar muchos errores comunes de seguridad.

4.3.3. Tecnologías para la implementación del front-end

El siguiente apartado contiene un sumario de tecnologías que facilitan la implementación del front-end y diseño de interfaces del proyecto. Por cada una de estas, se han detallado dos o tres propuestas en función de la popularidad, extensión y uso.

Frameworks de desarrollo

Prácticamente la totalidad de propuestas para el desarrollo del frontend se basan en Javascript, pues es un lenguaje web interpretado ejecutado en el lado del cliente. En este caso, se detalla un conjunto de frameworks con gran popularidad en la actualidad.

- **Angular**³¹ es una plataforma para diseñar aplicaciones multidispositivo creada por Google que facilita el desarrollo de aplicaciones web SPA, proporcionando herramientas para trabajar con los elementos de la página web de una manera óptima. Tiene una gran comunidad detrás y una documentación extensa, pero en cierto modo fuerza al usuario a aprender TypeScript.
- **Vue**³² es un framework progresivo con un ecosistema que se plantea cercano, versátil y con gran rendimiento. Esta herramienta, diseñada para construir interfaces de usuario reactivas, se creó con el propósito de ser fácil de integrar con

²⁷<https://spring.io/projects/spring-data-solr>

²⁸<https://spring.io/projects/spring-data-elasticsearch>

²⁹<https://expressjs.com/>

³⁰<https://www.djangoproject.com/>

³¹<https://angular.io/>

³²<https://vuejs.org/>

otras librerías o proyectos a través del desacople de las partes que lo conforman. Es un framework relativamente nuevo que mejora algunos conceptos de Angular y React, a través de una documentación clara y detallada.

- **React³³** es una librería para construir interfaces de usuario reactivas SPA. Esta herramienta de Facebook permite la creación de aplicaciones web potentes que requieran un intercambio de datos constante. Tiene una curva de aprendizaje algo elevada que recompensa con eficiencia y responsividad.

Librerías de diseño de interfaces y componentes

En este apartado se detallan tres librerías de diseño de interfaces, que complementan el front-end con animaciones y elementos preconstruídos. La selección ha sido realizada en base al número de estrellas que acumula cada proyecto en Github.

- **Bootstrap³⁴** es la librería de diseño más popular para construir aplicaciones web responsive pensadas en el mobile first. Dispone elementos que al utilizarlos aplican diferentes propiedades HTML, CSS y JS con un esfuerzo mínimo. Posee una gran comunidad detrás que además contribuye con plugins extra.
- **Foundation³⁵** es una librería open-source que incluye semántica. Su uso es intuitivo, complementándose con entrenamiento, soporte y consultas, por lo tanto garantiza una gran ayuda para los desarrolladores. Tiene además la capacidad de ser utilizado para diseñar plantillas de correo electrónico.
- **Semantic UI³⁶** posee el enfoque en realizar más semántico el proceso de construcción de una página. La principal funcionalidad consiste en utilizar principios del lenguaje natural para hacer el código más legible y fácil de entender. Posee una documentación bien organizada y con múltiples guías para empezar.

³³<https://reactjs.org/>

³⁴<https://getbootstrap.com/>

³⁵<https://foundation.zurb.com/>

³⁶<https://semantic-ui.com/>

5. Metodología

El proyecto ha seguido un modelo de desarrollo incremental mediante dos hitos compuestos por iteraciones cada dos semanas. Al finalizar cada iteración se ha realizado una revisión con los tutores del proyecto con el fin de analizar el avance, responder dudas surgidas durante el sprint y planificar el siguiente. Al finalizar el primer hito, con un desarrollo funcional de la mayoría de características, se empezó el segundo desde el principio evitando los errores realizados en el primer hito y mejorando la calidad del código. El trabajo realizado en cada hito e iteración se detalla en el siguiente apartado.

5.1. Hitos e iteraciones

Este apartado trata la duración y tareas propuestas en cada iteración. Se especifican las tareas a grandes rasgos, sin entrar en detalle.

5.1.1. Hito 1

El primer hito ha consistido en realizar una primera implementación del proyecto, probando las diferentes propuestas tecnológicas y analizando las interacciones e integraciones entre ellas.

Iteración I

La primera iteración ha analizado el problema a tratar así como las tecnologías para abordarlo y qué se quiere mostrar. Concretamente las tareas han sido:

- Estudiar el marco teórico y documentarlo.
- Establecer metodología de trabajo.
- Establecer objetivos.
- Describir requisitos funcionales.

Iteración II

La segunda iteración ha consistido en especificar ciertas bases del proyecto y construir una implementación inicial de algunas partes independientes probando diferentes tecnologías. Las tareas han sido:

- Prototipar interfaces.

- Probar y escoger tecnologías para el desarrollo y documentarlo.
- Construir analizador RSS.
- Definir casos de uso de la aplicación.

Iteración III

La tercera iteración ha pretendido conseguir una API REST funcional mínima que interactuase con el motor de base de datos. Las tareas han sido:

- Definir modelo de negocio.
- Construir API REST con CRUD mínimo.
- Definir esquema de datos provisional e integrar motor de búsqueda.

Iteración IV

En la cuarta iteración se ha integrado un framework de front-end que muestre los resultados y la herramienta de procesamiento de lenguaje natural. El objetivo era llegar a una primera versión semi-funcional.

- Construir front-end mínimo y funcional e integrarlo con la API REST para mostrar los resultados.
- Documentar implementación inicial del proyecto.
- Integrar herramienta de procesamiento de lenguaje natural.

5.1.2. Hito 2

El segundo hito ha tomado como referencia el resultado implementado en el primero, ha replanteado algunas de sus elecciones del stack tecnológico y ha partido de una base más sólida sobre la que desarrollar.

Iteración V

- Estudio e integración de tecnologías CI/CD y contenedores.
- Definir y documentar arquitectura final seleccionada.
- Definir y documentar diseño final de la aplicación.

Iteración VI

- Detallar esquema de datos final y optimizar la indexación.
- Crear endpoints finales de la API, integrando facetados y aspectos del motor de búsqueda.

Iteración VII

- Optimizar el uso de la herramienta de NLP para disminuir el coste computacional.
- Reconstruir el front-end para mostrar el diseño final.

5.2. Herramientas de administración del proyecto

A continuación se detallan las herramientas que se han empleado para la organización del proyecto:

- **Workona¹:** extensión del navegador Chrome que facilita la gestión de las pestañas. Su principal funcionalidad es crear espacios de trabajo en el que se guardan las pestañas actualmente abiertas de una sesión a otra. Esto permite cambiar el dispositivo de trabajo fácilmente sin perder el contenido que se estaba visitando en ese momento. Al mismo tiempo también permite cambiar el proyecto en el que se esté trabajando con un solo clic.

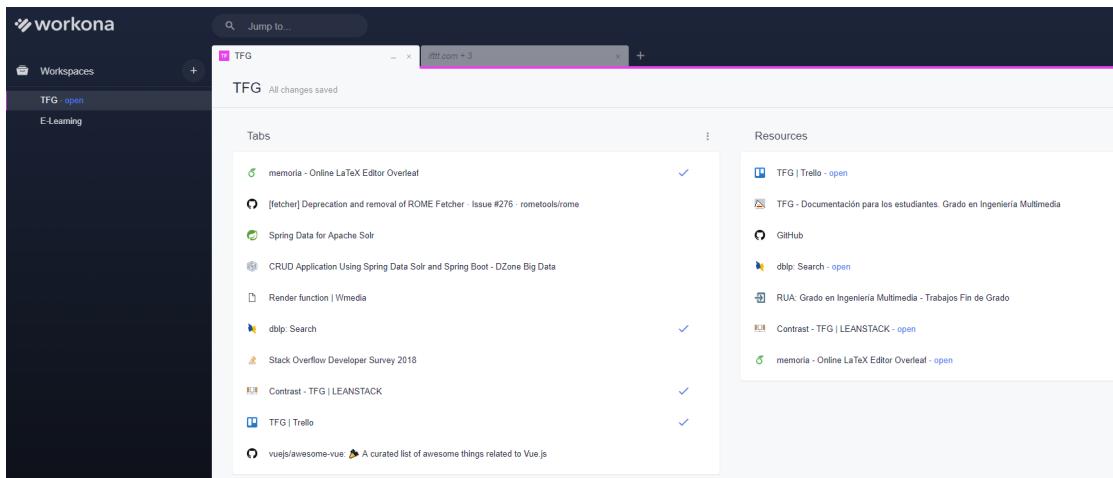


Figura 5.1.: Espacio de trabajo para Contrast

- **Trello²:** aplicación web que posibilita la creación de tableros con listas de tareas. Dichas tareas pueden ser etiquetadas y comentadas y se les pueden añadir archivos adjuntos. Los tableros permiten el flujo libre de tareas entre una lista y otra. En el caso del proyecto, se ha utilizado una metodología de gestión de trabajo Kanban, por lo tanto se han creado tres listas: por hacer, en proceso y finalizado. Se han añadido dos listas adicionales: una para acumular las tareas al finalizar una iteración y otra para definir tareas que surgen durante el proceso de desarrollo y así asignarlas en una iteración próxima.

¹<https://workona.com/>

²<https://trello.com/>

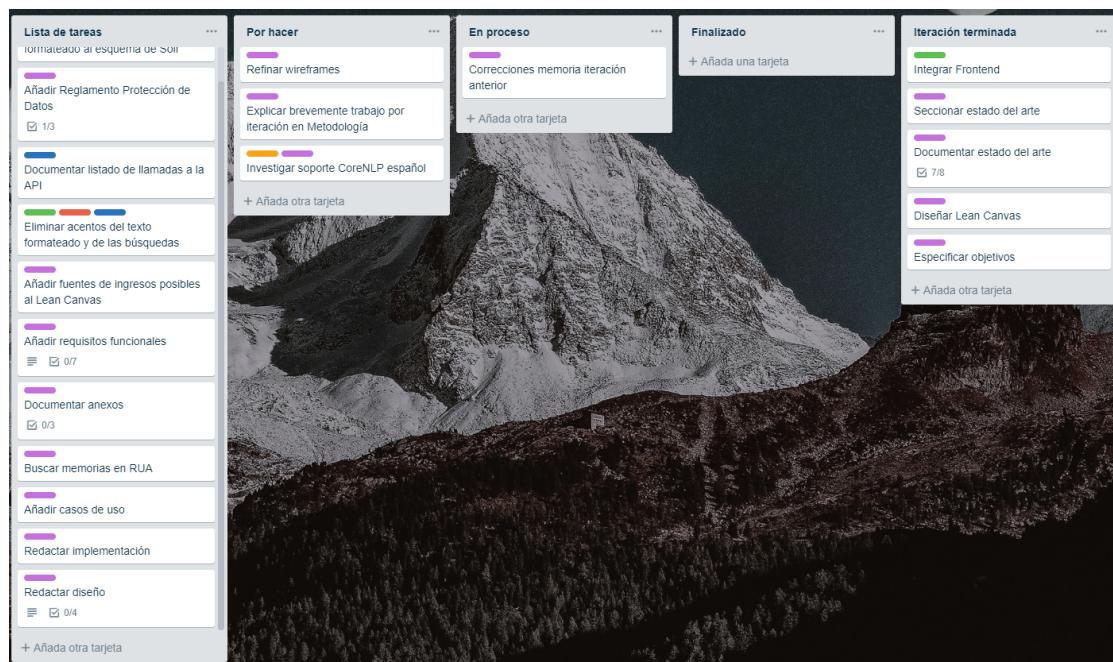


Figura 5.2.: Tablero para Contrast

- **Todoist³:** aplicación web que facilita la organización de tareas en el día a día. En el caso del proyecto es utilizada para organizar las tareas personales junto a las del trabajo de final de grado. Esta aplicación es utilizada junto a IFTT⁴ (If This Then That), que posibilita la integración de Todoist y Trello, de tal manera que cada vez que se cree una tarjeta en Trello, se añade una tarea en Todoist. La aplicación es utilizada bajo la filosofía GTD⁵ (Getting Things Done).

5.3. Herramientas de desarrollo

En esta sección se especifican las herramientas utilizadas para el desarrollo del proyecto:

- **Visual Studio Code⁶:** editor de código fuente multiplataforma, open-source que incluye muchas funcionalidades en forma de extensiones para trabajar con todos y cada uno de los lenguajes existentes. Su característica principal es el IntelliSense, que autocompleta las instrucciones mientras se escribe y destaca elementos con los que se interactuan.

³<https://todoist.com/>

⁴<https://ifttt.com/>

⁵https://es.wikipedia.org/wiki/Getting_Things_Done

⁶<https://code.visualstudio.com/>



Figura 5.3.: Integración IFTT de Todoist con Trello

- **GitKraken⁷:** cliente de Git⁸ con una interfaz visual que automatiza ciertos procesos internos de las instrucciones y permite un uso intuitivo de Git. Destaca por el uso de la técnica de *arrastrar y soltar* para interactuar con los *commits*.
- **Postman⁹:** herramienta que facilita el desarrollo, prueba y validación de elementos en una API REST. Nació como una extensión de Chrome que evolucionó en multiplataforma; hoy en día se ha convertido en un *must-have* del desarrollo web.
- **Overleaf¹⁰:** servicio de LaTeX¹¹ colaborativo en línea con el que redactar y publicar documentos académicos de una manera más rápida, visualizando el resultado tras cada modificación automáticamente.

⁷<https://www.gitkraken.com/>

⁸<https://git-scm.com/>

⁹<https://www.getpostman.com/>

¹⁰<https://www.overleaf.com/>

¹¹<https://www.latex-project.org/>

6. Análisis y especificación

6.1. Especificación requisitos funcionales

En el Anexo B se detallan los requisitos funcionales de la aplicación.

6.2. Casos de uso

La figura 6.1 describe los casos de uso diseñados a partir de los requisitos funcionales descritos en el Anexo B.

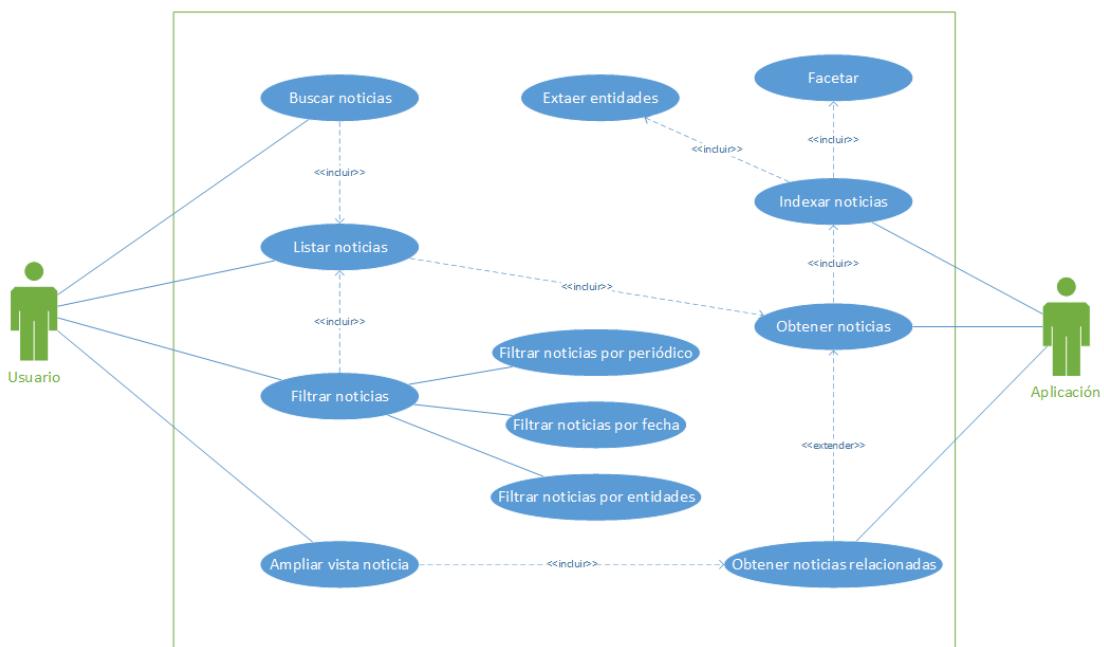


Figura 6.1.: Casos de uso.

7. Diseño

7.1. Arquitectura seleccionada

En la figura 7.1 se detalla el conjunto de tecnologías seleccionadas para el desarrollo del proyecto. Cabe destacar que algunas se han omitido, ya que sirven de soporte a otras que sí que se han detallado.

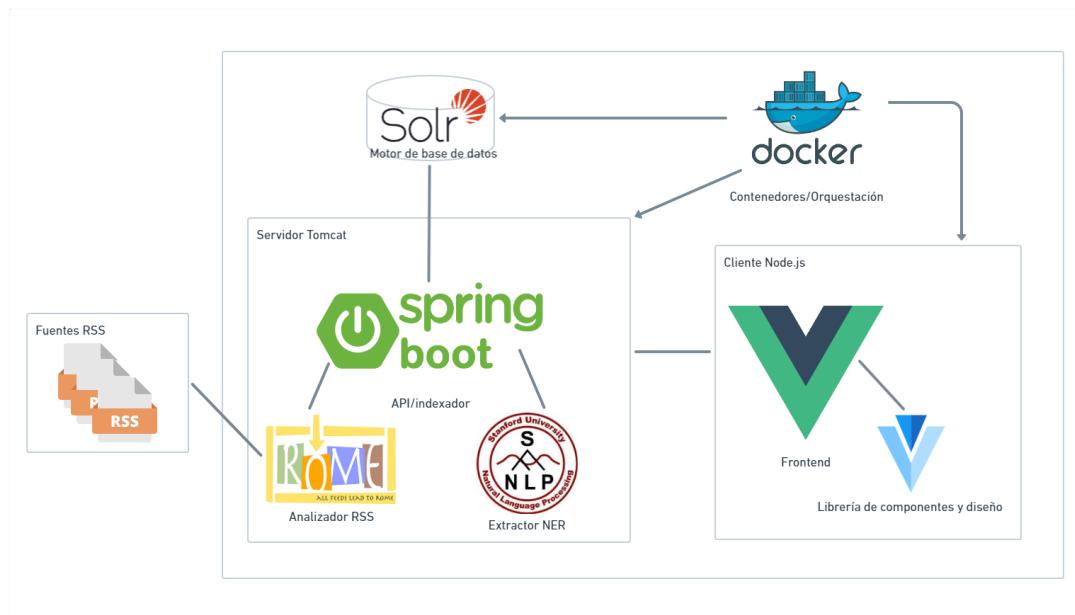


Figura 7.1.: Stack tecnológico utilizado por Contrast.

7.2. Tecnologías

El siguiente apartado justifica las tecnologías escogidas entre las analizadas en la sección 4, Marco Teórico. Por una parte, se ha optado por utilizar Java y por lo tanto Spring Boot, en el backend por la gran compatibilidad que ofrece con el resto de tecnologías que se tenían que integrar con él, por su madurez y por su extensa documentación. Esta decisión también se ha visto condicionada por:

- Tener una capa de abstracción para tratar directamente con el motor de elección de base de datos.

- Tener la posibilidad de utilizar CoreNLP mediante una dependencia del proyecto, simplificando su uso e integración.

Dado que se ha escogido Java, la elección de Rome como analizador era muy clara. Esto es debido a que se integra del mismo modo como una dependencia más del proyecto, con el añadido de ser además la herramienta más potente de las tres analizadas.

Entre los motores de búsqueda la elección ha sido más difusa, de hecho se ha realizado el desarrollo prácticamente paralelamente sobre ambas opciones. Esto ha sido posible gracias a que la capa de abstracción de Spring Boot sobre Apache Solr y Elasticsearch, permitía con modificaciones mínimas cambiar entre un motor u otro. Finalmente se ha escogido Solr dado que posibilitaba crear previamente el esquema de datos mediante una sencilla declaración de parámetros en un XML.

Como herramienta de procesamiento de lenguaje natural, la balanza se ha inclinado hacia CoreNLP por las razones que se detallan a continuación:

- CoreNLP se ha integrado con el backend escogido como una dependencia más, simplificando su uso.
- CoreNLP posee una mayor documentación que el resto analizado, facilitando el entendimiento del uso de los extractores.
- CoreNLP es la herramienta más utilizada de las tres analizadas en este ámbito.

Por último, el frontend se ha construido con Vue.js. Dado que la aplicación requiere de pocas interfaces y pocos elementos en esta, este framework posibilita un rápido desarrollo con componentes preconstruidos sobre los que tan solo había que enlazar los datos. Encima de este se ha utilizado Vuetify como librería de diseño y componentes, que al ser nativa de Vue, ha permitido embellecer el resultado sin realizar excesivas modificaciones.

7.3. Diagrama de clases

A continuación se describen el conjunto de clases y componentes creados durante el desarrollo del proyecto. Se ha dividido en dos imágenes distintas: la figura 7.2 detalla la relación entre las clases implementadas en la API o backend; por otro lado, la figura 7.3 muestra la conexión entre los componentes del frontend o capa de visualización.

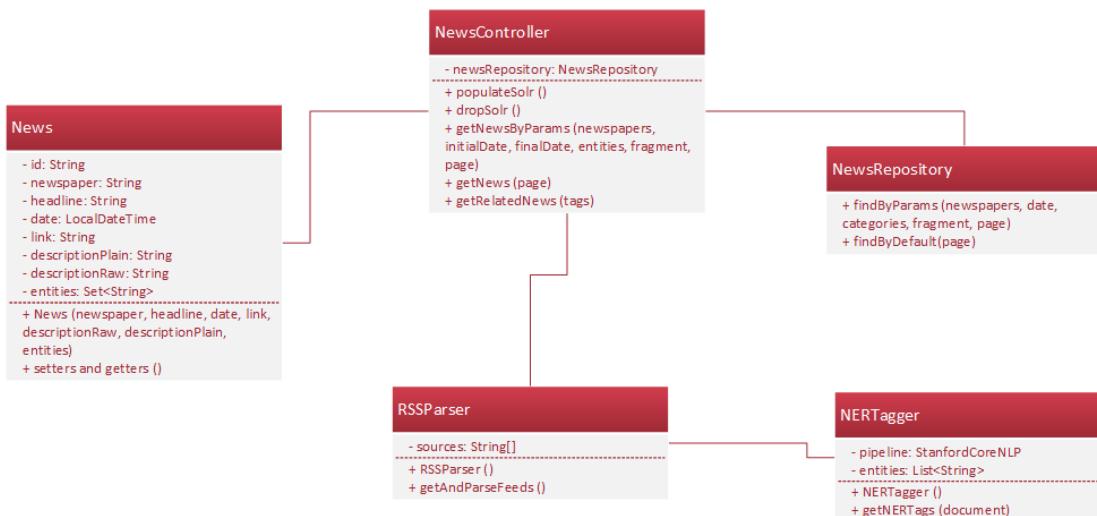


Figura 7.2.: Diagrama de clases del backend.

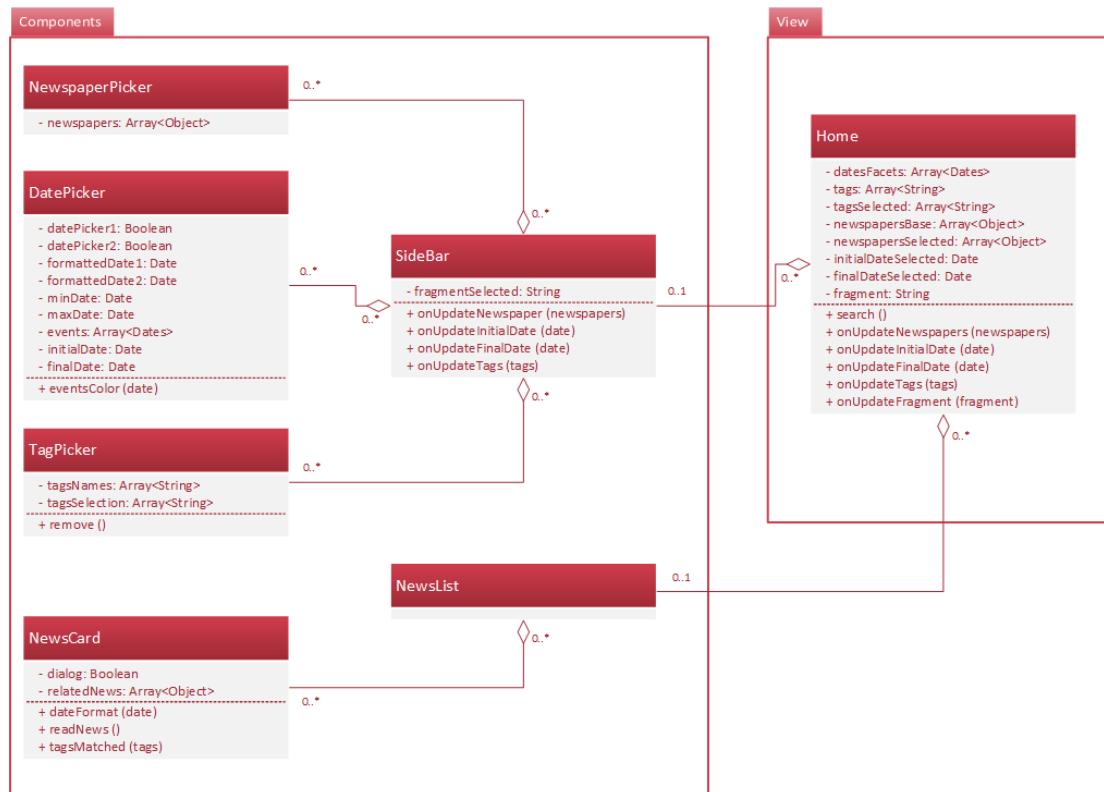


Figura 7.3.: Diagrama de clases del frontend.

7.4. Mockups

En la siguiente sección se detallan los mockups que se realizaron para ambos Hitos. El primero plasma un esbozo de la idea original, sin entrar en detalles de su posterior implementación y el segundo, en cambio, muestra la idea refinada con un aspecto similar al de su implementación final.

7.4.1. Hito 1

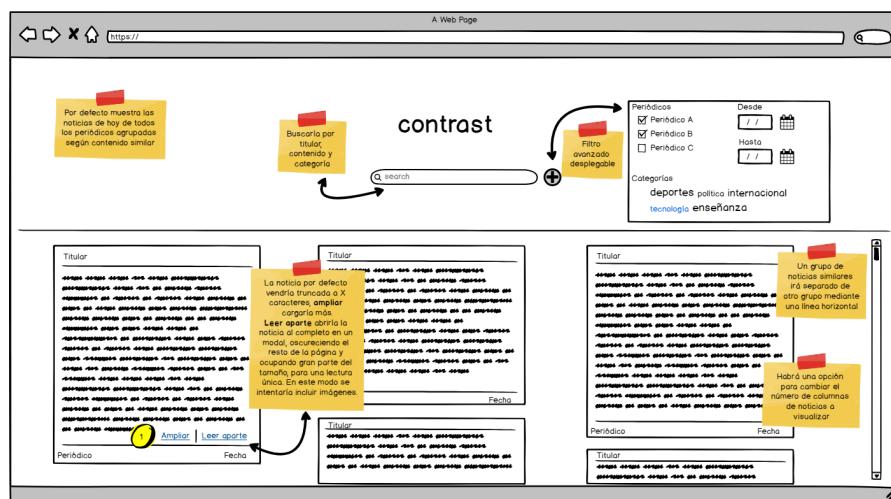


Figura 7.4.: Prototipo de interfaz para página de inicio y resultado búsqueda.

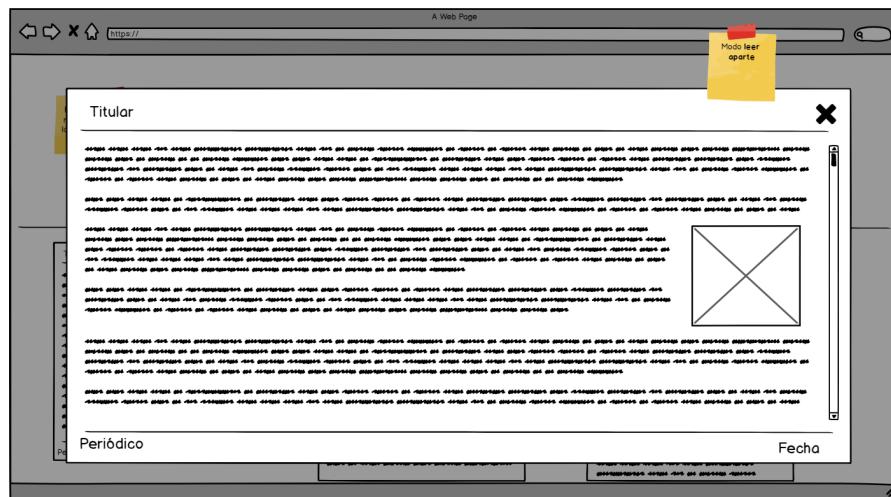


Figura 7.5.: Prototipo de interfaz para noticia leída en modo leer aparte.

7.4.2. Hito 2

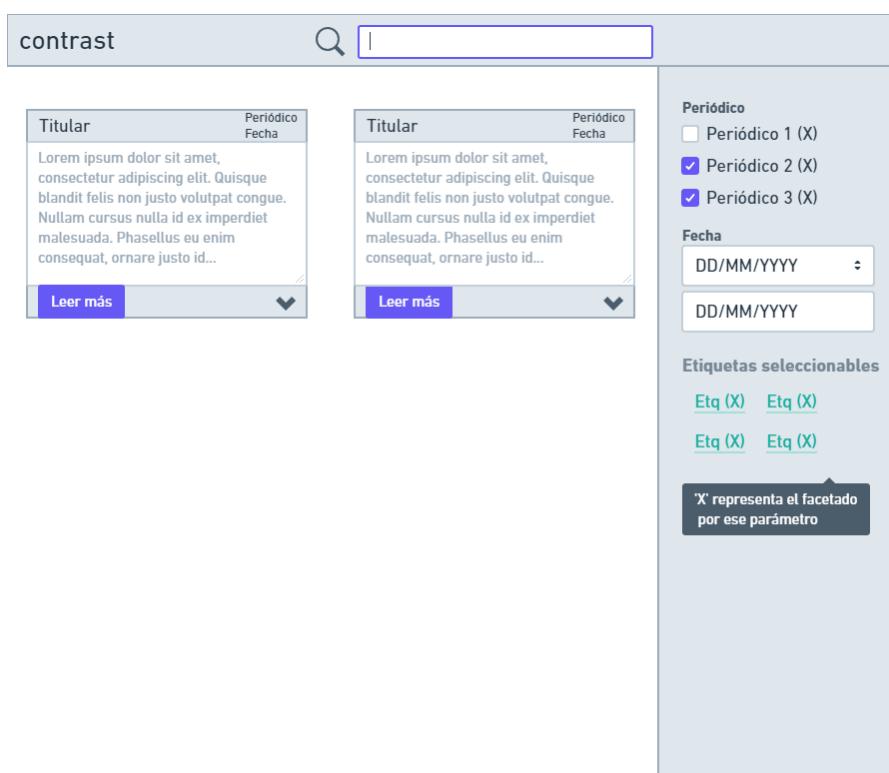


Figura 7.6.: Diseño final de interfaz para página de inicio y resultado búsqueda.

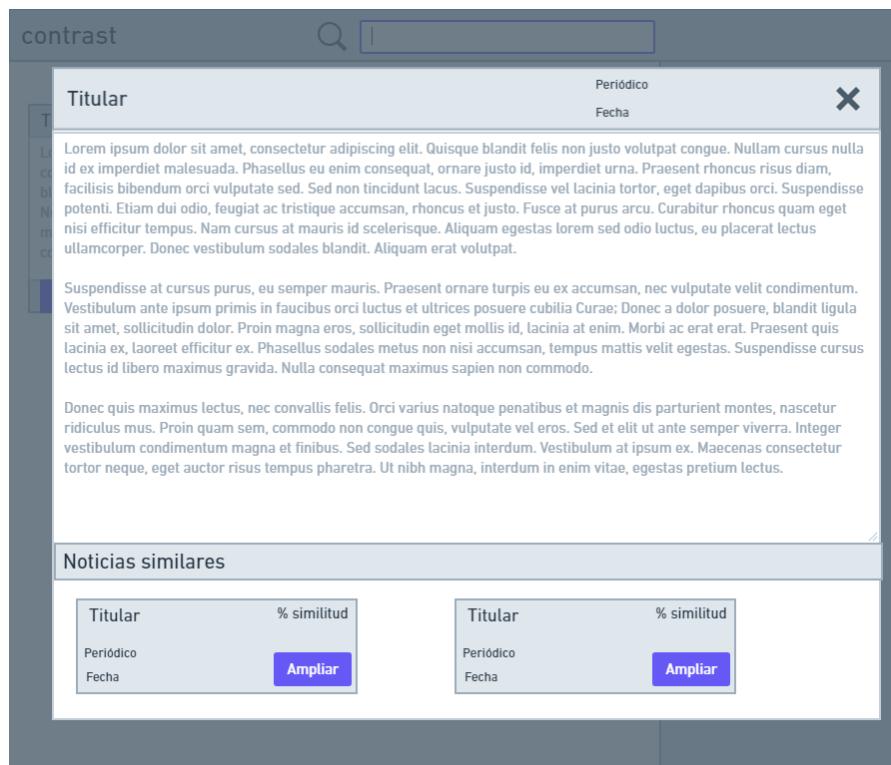


Figura 7.7.: Diseño final de interfaz para noticia leída en modo leer aparte.

8. Implementación

El siguiente capítulo detalla el avance conseguido en cada iteración, la problemática encontrada y qué pasos se han seguido para resolverla.

8.1. Hito 1

8.1.1. Iteración I

En primer lugar se elaboraron unos objetivos tangibles y alcanzables que cumplieran las necesidades del proyecto y los intereses personales. Acto seguido se estableció una metodología de trabajo basada en la recopilación de documentación detallada en el capítulo 5.

El estudio del marco teórico en el que se engloba el proyecto ha sido realizado en distintas etapas. La primera de ellas ha consistido en la recolección de información acerca de qué productos se encuentran en el mismo nicho de mercado y cumplen funciones similares. Para ello se ha realizado el registro en cada una de las plataformas y se ha utilizado con el mismo conjunto de fuentes durante un par de días. Este proceso ha resultado muy interesante ya que no solo sirvió para determinar qué se quería mostrar, sino que también se utilizó para observar cómo mostrar esa información. Algunos servicios no han podido utilizarse debido a que eran de pago, en este caso se observaron notas de prensa y contenido multimedia relacionado con el servicio.

El siguiente paso ha sido la investigación del marco legal en el que se engloba el proyecto. Este es un aspecto importante dado que Contrast se vale de contenido protegido de terceros. Se realizó una búsqueda de licencias y avisos legales para la reproducción del contenido en sitios de terceros y se además se documentaron sucesos relevantes al respecto, como el conflicto AEDE.

El paso posterior ha consistido en el estudio de las tecnologías con las que se ha desarrollado el proyecto. Para ello se plantearon diversas propuestas en cada uno de los apartados, según la investigación en plataformas técnicas especializadas.

Tras esto, se desgranaron los objetivos en diferentes requisitos funcionales que la aplicación debería satisfacer y se completaron con observaciones extraídas de las plataformas analizadas.

8.1.2. Iteración II

La segunda iteración comenzó describiendo los casos de uso.

Tras finalizar el paso anterior, se procedió a prototipar la primera versión de las interfaces a un bajo nivel, que incluiría todos los requisitos funcionales que se pudieran

mostrar visualmente junto a elementos de diseño observados en los servicios analizados en el 4. Para ello se realizaron dos prototipos sencillos con las dos vistas principales que albergaría la aplicación. Por un lado se creó la visualización principal o por defecto, con dos áreas significativas claramente diferenciadas, una extensión para el listado de noticias y otra para la búsqueda y filtrado de estas. En la segunda visualización se mostró el resultado de ampliar una noticia para su lectura extendida. Cabe destacar que este primer prototipado se utilizó para volcar las ideas que se tenían al respecto en la implementación del proyecto, sin tener en cuenta elementos de diseño específicos. Es por ello que se utilizaron múltiples anotaciones textuales de varios elementos para clarificar su utilidad final.

El paso siguiente consistió en probar las diferentes tecnologías propuestas para aumentar la documentación en la memoria y así poder tomar una decisión respecto a ellas en su uso para la implementación del proyecto. Dado que los lenguajes de programación ya eran conocidos y habían sido utilizados en otros proyectos, no se destinó a tiempo a tratar con ellos específicamente ya que en el uso de las tecnologías seleccionadas basadas en ellos habría tiempo suficiente para decantarse por uno o por otro. Es por ello que se abordó en primer lugar las tecnologías que darían soporte al backend y por ende se empezó a testear las diferentes librerías de recuperación y parseo de contenido RSS, con ejemplos mínimos proporcionados por la propia documentación de la tecnología pero que satisfacen los intereses del proyecto, pues la funcionalidad que se requiere de ellos para Contrast no es compleja.

En el extracto de código 8.1 se detalla el proceso de obtención de contenido de un feed desde una URL en la librería Universal Feed Parser de Python.

Listado 8.1: Universal Feed Parser

```

1 import feedparser
2 d = feedparser.parse('URL-DEL-FEED')

```

El código incluido en 8.2 contiene el proceso que utiliza la librería Rome Tools de Java para obtener contenido de un feed en una URL.

Listado 8.2: Rome Tools

```

1 import java.net.URL;
2 import java.io.InputStreamReader;
3 import com.rometools.rome.feed.synd.SyndFeed;
4 import com.rometools.rome.io.SyndFeedInput;
5 import com.rometools.rome.io.XmlReader;
6
7 public class FeedReader {
8
9     public static void main(String[] args) {
10         if (args.length==1) {
11             try {
12                 URL feedUrl = new URL("URL-DEL-FEED");
13

```

```
14     SyndFeedInput input = new SyndFeedInput();
15     SyndFeed feed = input.build(new XmlReader(feedUrl)
16         );
17 }
18 catch (Exception ex) {
19     ex.printStackTrace();
20     System.out.println("ERROR: "+ex.getMessage());
21 }
22 }
23 }
```

La pieza de código en 8.3 describe los pasos necesarios para recuperar contenido de un feed en una URL con la librería rss-parser de Javascript.

Listado 8.3: rss-parser

```
1 let Parser = require('rss-parser');
2 let parser = new Parser();
3
4 (async () => {
5     let feed = await parser.parseURL('URL-DEL-FEED');
6 })();
```

Tal y como se puede observar, las propuestas basadas en Python y Javascript son más sencillas y legibles dado a la falta de tipado ya que son lenguajes orientados a una mayor velocidad y flujo. Por otro lado la propuesta con Java es autoexplicativa debido a justo lo contrario que las anteriores; a pesar de ser más extensa, se observa con mejor definición el proceso y su control.

Tras esta prueba de concepto, se pasó a investigar las propuestas de los motores de búsqueda. Splunk fue descartado automáticamente debido a su alta orientación a Big Data, así que la dualidad estaba entre Solr y Elasticsearch. Dado que ambos están construidos sobre Apache Lucene comparten muchos aspectos en común, especialmente en cuanto a funcionalidad, pero teniendo en cuenta que la experiencia en estos motores es inexistente, se optó por empezar el desarrollo con aquel que fuese más intuitivo de utilizar. En este aspecto Solr sufrió una discriminación positiva ya que cuenta con una GUI que permite realizar muchas operaciones básicas. En el caso de Elasticsearch, se puede obtener una GUI a través de frontends desarrollados por terceros, sin soporte oficial.

Elasticsearch también ha sido instalado para que, una vez definido el esquema de datos, realizar pruebas de indexación y recuperación sobre ambos motores y así poder obtener una experiencia real del proceso que se requiere para el proyecto.

Por otra parte, se probaron las herramientas de procesamiento de lenguaje natural seleccionadas. Tanto CoreNLP como Freeling, proporcionan una demo web en la que puedes insertar texto y extraer características de él, aunque las opciones disponibles no permiten personalización. En ambas se introdujo la misma frase: ^{Esto}es una prueba



Figura 8.1.: Interfaz de usuario gráfica de Apache Solr

de extracción de entidades para el TFG de Ingeniería Multimedia de la Universidad de Alicante de Adrián Tomás.”



Figura 8.2.: Demostración de extracción de entidades en CoreNLP



Figura 8.3.: Demostración de extracción de entidades en Freeling

En la extracción de entidades se puede constatar que Freeling considera todo el contenido a partir de la palabra ”TFG” como una sola entidad y que las obtenidas por CoreNLP se acercan bastante a las esperadas. En los casos de Freeling y OpenNLP se halla además que la documentación tiende a ser escueta y que su presencia en los foros especializados es menor. Con todo ello se escoge CoreNLP sobre las demás propuestas.

Por último para terminar con la prueba de las tecnologías seleccionadas para el backend, se repasan los frameworks sobre los que se desarrollaría la API REST. Inicialmente en este apartado no se le da excesiva importancia a la elección del framework en sí, dado que la implementación planteada es sencilla. Es por ello que se decide priorizar la integración del framework con las tecnologías descritas anteriormente para facilitar el desarrollo del proyecto. En este sentido Spring Boot contiene una capa de abstracción para ambos motores de búsqueda propuestos y además tanto la herramienta de parseo de RSS (Rome Tools), como la de extracción de entidades (CoreNLP) se integran en el proyecto como una dependencia más en el archivo pom.xml.

Si se quisiera utilizar Express.js, habría que realizar las llamadas a la API del motor de búsqueda manualmente, el analizador de RSS se integraría fácilmente y habría que

realizar llamadas a la API proporcionada por CoreNLP, la cual además habría que ejecutarla en un servidor.

Django en este sentido pecaría de la misma situación que Express.js, pues son frameworks ligeros pensados para escalarlos a medida y que por ello, las integraciones son algo más costosas aunque más personalizables.

Una vez expuestos los razonamientos de cada una de las tecnologías probadas anteriormente, se decide utilizar Spring Boot, Rome y CoreNLP en la parte del backend.

La etapa siguiente era la prueba y decisión de tecnologías para el desarrollo del frontend. Las propuestas están construidas en Javascript y con unos principios muy similares, ya que comparten una gran cantidad de características y en muchos aspectos la transición de uno a otro tiene una curva de aprendizaje ligera. Los tres son frameworks jóvenes pero altamente utilizados por la comunidad, poseen extensa documentación y están en constante evolución, es por ello que la decisión en este campo ha consistido más en una preferencia personal.

Dado que Vue.js es el más nuevo entre ellos, fue construido basándose en las otras dos opciones y es la más sencilla de las propuestas en cuanto a implementación, se ha decidido utilizarlo como framework para el frontend.

En cuanto a las librerías de diseño y componentes, tanto Bootstrap como Vuetify ofrecen integraciones sencillas en la arquitectura del proyecto, teniendo la segunda opción una mayor definición semántica ya que parte de los componentes naturales de Vue.js. En el caso de Foundation, su integración es más compleja ya que requiere componentes de terceros para apoyar su funcionalidad al completo. Bootstrap es más extenso que Vuetify pero las necesidades del proyecto en cuanto a componentes no son tan grandes.

Listado 8.4: Declaración de un botón en Vuetify

```
1 | <v-btn color="success">Success</v-btn>
```

Listado 8.5: Declaración de un botón en Bootstrap

```
1 | <button type="button" class="btn btn-primary">Primary</button>
```

Es por ello que se decide utilizar Vuetify como librería de componentes y diseño ya que facilita la nomenclatura y coherencia semántica del frontend.

Una vez escogido el stack de tecnologías con los que inicialmente se va a desarrollar el proyecto, se procede a construir el analizador RSS con Rome partiendo del ejemplo anterior.

Por un lado se crea un archivo que almacene por línea los enlaces de las fuentes de las cuales se va a recuperar contenido RSS. Posteriormente, utilizando el ejemplo proporcionado por la documentación de Rome y mostrado anteriormente, se construye el analizador imprimiendo en pantalla los resultados de la recuperación y parseo de los RSS. Este es un paso importante, pues ha servido para observar qué campos van a conformar el esquema de datos.

8.1.3. Iteración III

La tercera iteración ha comenzado analizando los resultados del analizador construido en la iteración anterior. El objeto de análisis ha sido determinar qué campos de datos van a ser útiles para el desarrollo de la aplicación. Tal y como estaba especificado en los requisitos funcionales, cada noticia debe poseer al menos el titular, la fecha de publicación y el periódico al que pertenece para poder mostrarse. Adicionalmente se define un campo de datos más, las entidades que se reconocen en la noticia. Los campos descritos anteriormente definen el esquema mínimo de datos para la aceptación de una noticia. Rome proporciona estos datos a través de las siguientes llamadas:

Listado 8.6: Funciones de Rome para la consecución del esquema mínimo de datos

```

1 SyndFeed feed = new SyndFeedInput().build(new XmlReader(stream));
2 String newspaper = feed.getTitle();
3 for (SyndEntry entry : feed.getEntries()) {
4     String headline = entry.getTitle()
5     Set<String> entities = new HashSet<String>();
6     for (SyndCategory cat : entry.getCategories()) {
7         entities.add(cat.getName());
8     }
9     LocalDate publishedDate = entry.getPublishedDate().toInstant()
10        .atZone(ZoneId.systemDefault()).toLocalDate();
11 }
```

Para la conformación del esquema de datos óptimo, es decir, aquel en el que el funcionamiento del objeto en la aplicación sea el idóneo, se necesitaría almacenar dos campos adicionales. Estos son el enlace a la noticia y el cuerpo de la noticia. Se añaden como datos adicionales dado que no todas las fuentes que se probaron proporcionan estos datos y porque son prescindibles para el funcionamiento general pensado de la aplicación.

Listado 8.7: Funciones de Rome para la extracción de los campos adicionales para el esquema de datos óptimo

```

1 entry.getLink();
2 entry.getDescription().getValue();
```

La función para la obtención del cuerpo de las noticias devuelve además todas las etiquetas HTML que contiene el cuerpo original de la noticia, esto es el formato de visualización con el que fue creado, además de los elementos multimedia. Es por ello que debido al RF09 de mantenimiento de formato en el cuerpo de las noticias, se decide utilizar dos campos para el almacenamiento de la descripción de las noticias. El primero almacena el valor obtenido por la función; el segundo almacena el valor obtenido tras eliminar todas las etiquetas HTML. Esto es realizado a través de una expresión regular:

Listado 8.8: Eliminación de etiquetas HTML del cuerpo de la noticia

```

1 entry.getDescription().getValue().replaceAll("<[^>]*>", "")
```

De esta manera, se utiliza el campo sin etiquetas para la recuperación de información y el original para la visualización únicamente.

Una vez concretados los esquemas de datos, se procede a la construcción del modelo que los soporte y la API que interactúe con ellos. Para ello, se incluye la dependencia de Spring Boot Data sobre Solr y siguiendo las líneas propuestas en la documentación de Spring Boot, se empieza por crear el modelo con las variables descritas a continuación.

Listado 8.9: Definición del modelo de datos

```
1  @SolrDocument(collection = "news")
2  public class News {
3      @Id
4      @Field
5      private String id;
6
7      @Field
8      private String newspaper;
9
10     @Field
11     private String headline;
12
13     @Field
14     private LocalDateTime date;
15
16     @Field
17     private String link;
18
19     @Field
20     private String descriptionPlain;
21
22     @Field
23     private String descriptionRaw;
24
25     @Field
26     private Set<String> entities;
27 }
```

Cabe destacar que mediante:

```
1  @SolrDocument(collection = "news")
```

Se especifica que el modelo de datos asociado va a conformar los campos de datos que se van a encontrar en el motor de búsqueda Apache Solr. De este modo, Solr va a crear sus índices a partir de esta definición, con lo que se abstrae la configuración del esquema de datos del motor de búsqueda a la declaración simple de variables en el modelo. Este

modo de definición es llamado Schemaless Mode¹.

Posteriormente se construye el repositorio y la configuración que darán soporte a las operaciones transaccionales de inserción y recuperación de información.

Listado 8.10: Interfaz del repositorio Solr

```
1 public interface NewsRepository extends SolrCrudRepository<News ,  
2   String > {}
```

Al ser una extensión del repositorio proporcionado por Solr, permite derivar las queries a través del nombre utilizado en las funciones siguiendo las estrategias definidas en la documentación².

Listado 8.11: Configuración de conexión a cliente Solr

```
1 @Configuration  
2 @EnableSolrRepositories(basePackages = "contrast.contrast")  
3  
4 @ComponentScan  
5 public class SolrConfig {  
6  
7     @Value("${spring.data.solr.host}")  
8     String solrURL;  
9  
10    @Bean  
11    public SolrClient solrClient() throws MalformedURLException ,  
12        IllegalStateException {  
13        return new HttpSolrClient.Builder("http://" + solrURL) .  
14            build();  
15    }  
16  
17    @Bean  
18    public SolrTemplate solrTemplate(SolrClient client) throws  
19        Exception {  
20        return new SolrTemplate(client);  
21    }  
22}
```

Para completar la API REST mínima faltaría crear un controlador sobre el que realizar las llamadas. Para una primera implementación que permita probar los resultados, se decide crear las operaciones de guardado, borrado y recuperación total.

Listado 8.12: Controlador de la API REST de Contrast

```
1 @RestController
```

¹https://lucene.apache.org/solr/guide/7_7/schemaless-mode.html

²<https://docs.spring.io/spring-data/solr/docs/4.0.8.RELEASE/reference/html/repositories.query-methods.details>

```
2 public class NewsController {  
3  
4     @Autowired  
5     private NewsRepository newsRepository;  
6  
7     @PostMapping("/populate")  
8     public void populateSolr() throws IllegalArgumentException {  
9         RSSParser parser = new RSSParser();  
10        newsRepository.saveAll(parser.parseFeeds());  
11    }  
12  
13    @DeleteMapping("/dropSolr")  
14    public void dropSolr() {  
15        newsRepository.deleteAll();  
16    }  
17  
18    @GetMapping("/getNews")  
19    public Iterable<News> getAll() {  
20        return newsRepository.findAll();  
21    }  
22 }
```

Esta API es ejecutada con el servidor Tomcat integrado en la propia instalación de Spring Boot, ya que este framework provee muchas facilidades para un desarrollo ágil y personalizable.

Por último para satisfacer el último objetivo de la iteración, se procede a la definición del modelo de negocio mediante la creación de un lienzo de modelo para negocios o Lean Canvas. Este es utilizado para realizar una declaración de intenciones de los objetivos que persigue el proyecto; del mismo modo refuerza la importancia de algunas ideas en su implementación. Gran parte de los aspectos que se subrayan son tenidos en consideración únicamente en caso de que se publicase una versión comercial de la aplicación, ya que el cometido de este trabajo de final de grado es lograr una versión académica del estudio de la implementación.

8.1.4. Iteración IV

Para la implementación inicial del frontend, se desestima integrar directamente la librería de componentes, pues esta es una extensión final del framework base y se prima tener el control sobre la visualización de los resultados por encima de la estética en esta fase de la implementación. En este sentido se crea una única vista para mostrar el resultado de recuperar todas las noticias indexadas.

A la hora de realizar la llamada se percibe un problema de intercambio de recursos de origen cruzado pues ambos están siendo ejecutados en servidores y puertos distintos. Es por ello que se decide implementar un proxy inverso sobre el frontend para que lance todas las llamadas desde la propia dirección sobre la que se ejecuta la API.

Listado 8.13: Proxy inverso del frontend al backend

```

1 module.exports = {
2     devServer: {
3         proxy: {
4             '/api': {
5                 target: "http://localhost:8080",
6                 ws: true,
7                 changeOrigin: true
8             }
9         }
10    }
11}

```

La decisión de ejecutar ambas partes sobre servidores distintos es debido a motivos de desacople, mantenibilidad, escalabilidad y seguridad. Por una parte, en vistas a un entorno de producción se podrían desplegar diferentes réplicas de cada una de las partes sin que se vea comprometida la otra. Del mismo modo la caída de una de las partes no debería implicar la caída del servicio entero y uno de los principios que se persigue con esta decisión es la ortogonalidad de las partes implicadas, para que sean reemplazables sin afectar a la otra.

Llegada la hora de integrar la herramienta de procesamiento de lenguaje natural, en este caso Stanford CoreNLP, se añade como dependencia en el archivo pom.xml tanto la herramienta como el modelo entrenado para el español y se busca su integración en la fase de indexado de noticias. El objetivo que se pretende es enriquecer el ya definido dato de entidades de tal forma que sea la suma de las categorías proporcionadas por el analizador RSS y las entidades extraídas por la herramienta de procesamiento de lenguaje natural. Esto comporta cambios mínimos en la arquitectura actual, pues solo hay que añadir esta fase en la etapa de creación del objeto noticia tras el parseado.

Listado 8.14: Clase de reconocimiento de entidades

```

1 public class NERTagger {
2
3     private StanfordCoreNLP pipeline;
4
5     public NERTagger() {
6         Properties props = new Properties();
7         try {
8             props.load(IOUtils.readerFromString("StanfordCoreNLP-
9                 spanish.properties"));
10        } catch (IOException e) {
11            e.printStackTrace();
12        }
13        props.put("annotators", "tokenize,ssplit,pos,lemma,ner");
14        this.pipeline = new StanfordCoreNLP(props);
15    }

```

```

16     public Set<String> getNERTags(String doc) {
17         Set<String> nerTags = new HashSet<String>();
18         CoreDocument document = new CoreDocument(doc);
19         this.pipeline.annotate(document);
20         for (CoreEntityMention em : document.entityMentions()) {
21             nerTags.add(em.text());
22         }
23         return nerTags;
24     }
25 }
```

Al analizador de RSS se le añade las siguientes líneas para satisfacer la extracción de entidades. Cabe remarcar que se examina tanto el cuerpo de la noticia sin las etiquetas HTML, como el titular de la noticia. Esta decisión es debida a que como se ha mencionado anteriormente, hay noticias que no contienen el cuerpo de ella.

Listado 8.15: Uso del extractor de entidades y convergencia de ambos datos

```

1 NERTagger tagger = new NERTagger();
2 Set<String> entities = new HashSet<String>();
3 for (SyndCategory cat : entry.getCategories()) {
4     entities.add(cat.getName());
5 }
6 entities.addAll(tagger.getNERTags(entry.getDescription().getValue()
7     .replaceAll("<[^>]*>", ""))
+ " " + entry.getTitle()));
```

8.2. Hito 2

El segundo Hito parte unas semanas más tarde de finalizar el primer prototipo logrado en el anterior para replantear algunas decisiones arquitecturales y de implementación realizadas, que pueden favorecer o mejorar el desarrollo y/o despliegue del proyecto. El enfoque que se brinda a esta segunda etapa de desarrollo es para conseguir una primera versión de la aplicación con orientación a producción.

En este sentido y teniendo en cuenta que el desarrollo se estaba realizando sobre máquinas distintas con sistemas operativos distintos, se decide utilizar un sistema de virtualización con contenedores para independizar las dependencias y configuraciones de las diferentes partes implicadas en el proyecto. Esta decisión además facilita el posible despliegue en producción pues se utilizaría el mismo modus operandi. La herramienta que se escoge para posibilitar este escenario es Docker³.

Se crea un archivo de especificación de los servicios que van a utilizarse de manera independiente, pero no aislada entre ellos. Se definen instrucciones para crear contenedores de Apache Solr, de la API Spring Boot en Java y del frontend Vue sobre Node.js. El contenido de estos archivos se detalla en el Anexo C.

³<https://www.docker.com/>

Se intenta acoplar en el proyecto la herramienta de integración continua Travis CI⁴ sin éxito, debido a no conseguir correr Apache Solr en el susodicho servicio. Dicha imposibilidad causaba que la API fallase al no poder enlazar los repositorios y que por lo tanto, las pruebas y tests definidos fallasen siempre. En el caso de haber escogido Elasticsearch, Travis CI proporciona soporte para el uso de algunos motores de bases de datos⁵. El objetivo que se perseguía al intentar integrar este servicio era el de poder realizar pruebas para el correcto funcionamiento del proyecto de manera automatizada.

Se rediseña la interfaz de usuario desde los cimientos, esta vez introduciendo elementos de las últimas especificaciones de diseño y teniendo en cuenta la usabilidad y experiencia de usuario. Para ello se hace un mayor hincapié en la experiencia observada en el uso de las aplicaciones similares detalladas en el Marco Teórico. También se realizan simplificaciones y eliminación de adornos y funciones secundarias que no estaban descritas en los requisitos funcionales y que su aportación es irrelevante para el desarrollo académico de la aplicación.

Por último se documenta y describe la arquitectura final seleccionada contando con las tecnologías escogidas en el Hito 1 y las herramientas de soporte escogidas en esta iteración.

8.2.1. Iteración VI

En los resultados mostrados por las llamadas a la API se observan distintos problemas:

- Los resultados son sensibles a mayúsculas/minúsculas y signos de puntuación.
- Las entidades compuestas son tomadas en cuenta como elementos independientes.

Es por ello que se decide declarar un esquema manual de datos del que partirá Solr, dejando de lado el modo Schemaless. Para ello se crean tipos de datos que integren los filtros⁶ y tokenizadores⁷ que sean de interés para el proyecto.

Listado 8.16: Campo de datos personalizado

```

1  <fieldType name="text_general_keyword_tokenizer" class="solr.
2   TextField" positionIncrementGap="100" multiValued="true">
3   <analyzer type="index">
4     <tokenizer class="solr.KeywordTokenizerFactory"/>
5     <filter class="solr.ASCIIFFoldingFilterFactory"/>
6     <filter class="solr.LowerCaseFilterFactory"/>
7   </analyzer>
8   <analyzer type="query">
9     <tokenizer class="solr.KeywordTokenizerFactory"/>
10    <filter class="solr.ASCIIFFoldingFilterFactory"/>
11    <filter class="solr.LowerCaseFilterFactory"/>
```

⁴<https://travis-ci.org/>

⁵<https://docs.travis-ci.com/user/database-setup/>

⁶https://lucene.apache.org/solr/guide/6_6/filter-descriptions.html

⁷https://lucene.apache.org/solr/guide/6_6/tokenizers.html

```
11  </analyzer>  
12  </fieldType>
```

En este caso es de interés utilizar el tokenizador Keyword, que trata el campo de texto entero como un solo token y como filtros el ASCIIFolding para las acentuaciones y el LowerCase para evitar la distinción entre mayúsculas y minúsculas.

El esquema de datos final se describe en el Anexo A.

El siguiente paso ha consistido en realizar facetados en las consultas para enriquecer las vistas mostradas en los filtros del frontend. Dichos facetados están descritos en los requisitos funcionales y deben aplicarse sobre las fechas de publicación, el periódico y las entidades.

En este proceso se descubrió una nueva situación a afrontar pues al realizar los facetados por días, Solr agrupa los resultados teniendo en cuenta horas y minutos por lo que era un resultado indeseado. Esto tenía tres posibles soluciones:

- Crear un nuevo campo de datos que sea una copia del campo fecha de publicación pero truncando el valor al día únicamente.
- Despreciar la hora de publicación forzando manualmente un valor fijo para todas las noticias.
- Agrupar los resultados desde el backend, editando manualmente la respuesta de Solr.

Si el valor de hora y minuto hubiera sido relevante, se hubiera escogido la primera opción, pero dado que para la funcionalidad propuesta es completamente prescindible, se escogió la segunda y se fijó manualmente en la creación de las noticias que todas estuvieran publicadas a medianoche.

Por lo tanto se crean dos llamadas en el controlador y se elimina la llamada de recuperación de todas las noticias.

Listado 8.17: Endpoints finales de obtención de noticias

```
1  @GetMapping(value = "/content/{page}")  
2  @GetMapping(value = "/content/{newspapers}/{initialDate}/{finalDate}/{entities}/{fragment}/{page}")
```

La primera de ellas se propone para la visualización por defecto, sin aplicar ningún filtrado más que realizar la paginación. En la segunda llamada se incluyen todas las posibilidades de filtrado y es la que se propone para utilizar cuando el usuario interactúa con la aplicación introduciendo sus datos o preferencias de búsqueda de información.

8.2.2. Iteración VII

Esta iteración es planteada para incluir en el frontend la librería escogida de diseño de componentes y para optimizar el uso de CoreNLP pues el reconocimiento de entidades necesita varios segundos para mostrar los resultados.

Para satisfacer el primer cometido se empieza por aislar cada parte del frontend en componentes independientes que puedan ser utilizados en diversas vistas, a pesar de que solo se va a construir una única. De esta manera el código tiene una mejor mantenibilidad y es fácilmente escalable. Hecho esto y siguiendo la documentación de Vuetify, se añade la dependencia en el package.json y las correspondientes importaciones en los archivos de configuración del proyecto. Posteriormente se agregan y extienden componentes presentes en el propio framework de Vue.js, más otros implementados por Vuetify, que terminan aportando unas líneas estéticas con mucha riqueza sin tener que establecer manualmente código CSS o de estilos.

En este proceso se descubre que Vuetify es una librería aún más joven que Vue.js y que por lo tanto puede no satisfacer los intereses de muchos usuarios debido a sus limitaciones. Contrast en este caso, no necesita excesivos componentes para la visualización de los datos que pretende mostrar, por lo tanto el resultado previsto en el diseño no se ha visto alterado en gran medida. Aún así se pueden percibir ciertos cambios del diseño previsto al resultado final, que en absoluto empobrecen la calidad del producto final.

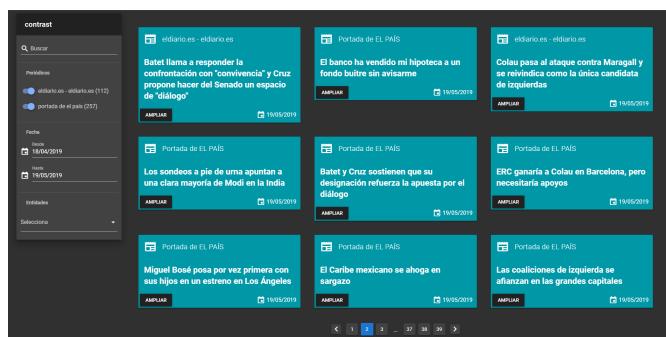


Figura 8.4.: Diseño final implementado de Contrast

Para reducir el coste computacional y los tiempos del uso de extractor de entidades, se ha tenido que analizar a documentación⁸ a fondo para considerar que aspectos se podían personalizar en la llamada. En primer lugar se han desactivado aquellas características que solo estaban implementadas para el modelo de reconocimiento en inglés. Estas son el reconocimiento de variables de tiempo con SUTime y la aplicación de clasificadores números para distinguir porcentajes, cifras, moneda, etc; que además no son de interés para el proyecto. Posteriormente se ha desactivado un componente que ralentizaba el tiempo de computación, que es el clasificador de grano fino. Por defecto coreNLP realiza clasificaciones por un total de 12 clases distintas que luego son recategorizadas en las 23 clases que da soporte el grano fino. En este caso además se han detallado que clasificadores son de interés, utilizando únicamente 3 de los 12 iniciales: organización, localización y persona. Esta ampliación de código se añade al constructor de la clase extractora de entidades, quedando finalmente así.

⁸<https://stanfordnlp.github.io/CoreNLP/ner.html>

Listado 8.18: Clase final extractora de entidades

```
1  public class NERTagger {  
2  
3      private StanfordCoreNLP pipeline;  
4      private List<String> selectedClassifiers;  
5  
6      public NERTagger() {  
7          Properties props = new Properties();  
8          try {  
9              props.load(IOUtils.readerFromString("StanfordCoreNLP-  
10                  spanish.properties"));  
11          } catch (IOException e) {  
12              e.printStackTrace();  
13          }  
14          props.put("annotators", "tokenize,ssplit,pos,lemma,ner");  
15          props.setProperty("ner.useSUTime", "false");  
16          props.setProperty("ner.applyNumericClassifiers", "false");  
17          props.setProperty("ner.applyFineGrained", "false");  
18          this.pipeline = new StanfordCoreNLP(props);  
19          this.selectedClassifiers = Arrays.asList("ORGANIZATION", "  
20              LOCATION", "PERSON");  
21      }  
22  
23      public Set<String> getNERTags(String doc) {  
24          Set<String> nerTags = new HashSet<String>();  
25          CoreDocument document = new CoreDocument(doc);  
26          this.pipeline.annotate(document);  
27          for (CoreEntityMention em : document.entityMentions())  
28              if (selectedClassifiers.contains(em.entityType()))  
29                  nerTags.add(em.text());  
30          return nerTags;  
31      }  
32  }
```

Una vez realizados los cambios se aprecia que el tiempo de computación pasa de segundos a milisegundos, convirtiendo su uso en una propuesta viable para un entorno de producción.

9. Pruebas y validación

10. Resultados

11. Conclusiones

11.1. Conclusiones

11.2. Líneas de trabajo futuras

Bibliografía

[AENOR, 1997] AENOR (1997). norma une 50136:1997.

A. Anexo A. Esquema de base de datos

A continuación se detallan los esquemas de datos implementados. En la versión final descrita en el presente documento se utiliza el esquema de datos óptimo.

Listado A.1: Esquema de datos mínimo utilizado

```
1 <field name="entities" type="text_general_keyword_tokenizer"
2   uninvertible="true" multiValued="true" indexed="true" stored
3   ="true"/>
4 <field name="date" type="pdate" uninvertible="true" multiValued
5   ="false" indexed="true" required="true" stored="true"/>
6 <field name="headline" type="text_es" uninvertible="true"
7   multiValued="false" indexed="true" required="true" stored="
8   true"/>
9 <field name="id" type="string" multiValued="false" indexed="true
9   " required="true" stored="true"/>
10 <field name="newspaper" type="text_general_keyword_tokenizer"
11   uninvertible="true" multiValued="false" indexed="true"
12   required="true" stored="true"/>
```

Listado A.2: Esquema de datos óptimo utilizado

```
1 <field name="entities" type="text_general_keyword_tokenizer"
2   uninvertible="true" multiValued="true" indexed="true" stored
3   ="true"/>
4 <field name="date" type="pdate" uninvertible="true" multiValued
5   ="false" indexed="true" required="true" stored="true"/>
6 <field name="descriptionPlain" type="text_es" uninvertible="
7   false" multiValued="false" indexed="true" required="false"
8   stored="true"/>
9 <field name="descriptionRaw" type="
9   text_general_keyword_tokenizer" uninvertible="false"
10  multiValued="false" indexed="false" stored="true"/>
11 <field name="headline" type="text_es" uninvertible="true"
12   multiValued="false" indexed="true" required="true" stored="
13   true"/>
14 <field name="id" type="string" multiValued="false" indexed="true
14   " required="true" stored="true"/>
15 <field name="link" type="string" uninvertible="false"
16   multiValued="false" indexed="false" required="true" stored="
17   true"/>
```

```
8 |     <field name="newspaper" type="text_general_keyword_tokenizer"  
|       uninvertible="true" multiValued="false" indexed="true"  
|       required="true" stored="true"/>
```

B. Anexo B. Requisitos funcionales

Identificador del requisito	RF01
Fuente del requisito	Aplicación web
Nombre	Listado de noticias
Descripción	La aplicación mostrará un listado de noticias recientes.
Características	<ul style="list-style-type: none">■ Las noticias estarán ordenadas según el día de publicación.■ Cada noticia de la lista contendrá, al menos, el titular, la fecha de publicación y el periódico al que pertenece.■ Este será el comportamiento por defecto de la carga inicial de la página.

Identificador del requisito	RF02
Fuente del requisito	Aplicación web
Nombre	Buscar noticias
Descripción	La aplicación obtendrá un listado de noticias según los parámetros introducidos por el usuario en el campo de texto.
Características	<ul style="list-style-type: none">■ El término de búsqueda se comparará con el titular, contenido y categorías de las noticias.■ La búsqueda podrá ser realizada con o sin acentos y obtendrá los mismos resultados.■ Cada noticia de la lista contendrá, al menos, el titular, la fecha de publicación y el periódico al que pertenece.

Identificador del requisito	RF03
Fuente del requisito	Aplicación web.
Nombre	Filtrar noticias por periódico.
Descripción	La aplicación obtendrá un listado de noticias con los periódicos seleccionados.
Características	<ul style="list-style-type: none">■ Todos los periódicos disponibles estarán seleccionados por defecto.■ Las noticias filtradas se ordenarán por fecha de publicación.■ Cada noticia de la lista contendrá, al menos, el titular, la fecha de publicación y el periódico al que pertenece.■ El periódico indicará en todo momento cuantas noticias de ese periódico está mostrando.■ Se podrán aplicar distintos filtros simultáneamente.

Identificador del requisito	RF04
Fuente del requisito	Aplicación web.
Nombre	Filtrar noticias por rango de fechas.
Descripción	La aplicación obtendrá un listado de noticias según el rango de fechas definido por el usuario.
Características	<ul style="list-style-type: none"> ■ Por defecto el rango de fechas estará situado entre la noticia más antigua y la más reciente. ■ Cada noticia de la lista contendrá, al menos, el titular, la fecha de publicación y el periódico al que pertenece. ■ Los días en que hay noticias tendrán un indicador que marque la cantidad. ■ Se podrán seleccionar únicamente fechas que contengan noticias. ■ Se podrán aplicar distintos filtros simultáneamente.

Identificador del requisito	RF05
Fuente del requisito	Aplicación web.
Nombre	Filtrar noticias por entidades.
Descripción	La aplicación obtendrá un listado de noticias según las entidades seleccionadas por el usuario.
Características	<ul style="list-style-type: none"> ■ Por defecto se mostrarán noticias con todas las combinaciones posibles de entidades. ■ Cada noticia de la lista contendrá, al menos, el titular, la fecha de publicación y el periódico al que pertenece. ■ Se podrán seleccionar cualquier cantidad de entidades, siempre y cuando hayan noticias que las contengan. ■ Se podrán aplicar distintos filtros simultáneamente.

Identificador del requisito	RF06
Fuente del requisito	Aplicación web.
Nombre	Ampliar noticia.
Descripción	La aplicación mostrará un botón en cada noticia para ampliar su vista y facilitar su lectura.
Características	<ul style="list-style-type: none"> ■ La aplicación ampliará la vista de la noticia ocupando más espacio en pantalla y bloqueando el resto del contenido. ■ La noticia mostrará, además de los ítems mencionados anteriormente, el cuerpo de la noticia. ■ Esta visualización tendrá elementos accionables que permitan volver a la vista anterior.

Identificador del requisito	RF07
Fuente del requisito	Aplicación web.
Nombre	Buscar noticias similares.
Descripción	La aplicación buscará noticias similares a la seleccionada.
Características	<ul style="list-style-type: none"> ■ Este comportamiento se dará únicamente cuando se amplíe una noticia. ■ Las noticias relacionadas se ordenarán según su similitud con la original. ■ De cada noticia relacionada se mostrará el titular, periódico, fecha de publicación y entidades en común con la seleccionada.

Identificador del requisito	RF08
Fuente del requisito	Aplicación web
Nombre	Indexación de noticias
Descripción	La aplicación indexará las noticias que obtenga del parseador para su posterior recuperación
Características	<ul style="list-style-type: none"> ■ Las noticias se indexarán si se ajustan al esquema de datos definido. ■ De cada noticia se almacenará el periódico, el titular, la fecha de publicación y las entidades como mínimo. Este será el esquema de datos mínimo de aceptación. ■ De cada noticia se almacenará adicionalmente el cuerpo de la noticia y el enlace. Estos datos junto a los especificados en el ítem anterior, conformarán el esquema de datos óptimo.

Identificador del requisito	RF09
Fuente del requisito	Aplicación web
Nombre	Mantenimiento de formato
Descripción	La aplicación mantendrá el formato original de las noticias en la visualización
Características	<ul style="list-style-type: none"> ■ Las noticias contendrán las etiquetas HTML del medio original que les dan formato. ■ Las noticias contendrán los elementos multimedia originales del medio digital.

Identificador del requisito	RF10
Fuente del requisito	Aplicación web
Nombre	Inserción de noticias
Descripción	La aplicación obtendrá noticias de distintos medios y las insertará en la BBDD
Características	<ul style="list-style-type: none"> ■ Las fuentes de las noticias quedarán definidas en un fichero externo. ■ Las noticias que se obtengan serán parseadas para ajustarse al esquema de datos.

Identificador del requisito	RF11
Fuente del requisito	Aplicación web
Nombre	Facetado por contenido
Descripción	La aplicación realizará distintos facetados que permitan complementar la información proporcionada por los filtrados
Características	<ul style="list-style-type: none"> ■ El facetado se realizará por fuentes, entidades y fechas de las noticias. ■ El resultado del facetado se mostrará junto a los filtros para una visualización óptima.

Identificador del requisito	RF12
Fuente del requisito	Aplicación web
Nombre	Paginación de noticias
Descripción	La aplicación mostrará un conjunto de noticias por página y permitirá avanzar y retroceder entre ellas.
Características	<ul style="list-style-type: none">■ El número de noticias a visualizar por página vendrá determinado por las dimensiones del contenido a mostrar, pero siempre se mantendrá una matriz cuadrada.■ Se proporcionarán elementos visuales sencillos para la navegación entre páginas, del mismo modo que para conocer en qué página se está.

Identificador del requisito	RF13
Fuente del requisito	Aplicación web
Nombre	Reconocimiento de entidades
Descripción	La aplicación utilizará extractores para reconocer entidades en las noticias.
Características	<ul style="list-style-type: none">■ Las entidades extraídas se indexarán junto a las categorías proporcionadas por el recuperador de medios RSS.■ Se hará una selección de las entidades relevantes a indexar en la base de datos.

C. Anexo C. Archivos Docker

Listado C.1: Archivo docker-compose.yml

```
1  version: "3.7"
2  services:
3    api:
4      image: spring-boot-custom
5      container_name: api
6      build: ./backend
7      ports:
8        - 8080:8080
9      depends_on:
10        - db
11      networks:
12        - contrast-net
13
14    db:
15      image: solr-from-schema
16      build: .
17      container_name: solr
18      entrypoint:
19        - docker-entrypoint.sh
20        - solr-precreate
21        - news
22        - /opt/solr/server/solr/configsets/newsConfig
23      ports:
24        - 8983:8983
25      networks:
26        - contrast-net
27      volumes:
28        - ./data:/opt/solr/server/solr/mycores
29
30    node:
31      image: node-vue
32      container_name: vue
33      build:
34        context: ./frontend
35        args:
36          - NODE_ENV=development
37      ports:
38        - 80:80
39      environment:
```

```
40      - NODE_ENV=development
41  depends_on:
42    - api
43  networks:
44    - contrast-net
45
46 networks:
47  contrast-net:
48    name: contrast-net
49
50 volumes:
51  data:
```

Listado C.2: Dockerfile de Solr

```
1 FROM solr:7.7.1-alpine
2 COPY coredir /opt/solr/server/solr/configsets/newsConfig
```

Listado C.3: Dockerfile de Vue.js

```
1 # build stage
2 FROM node:lts-alpine as build-stage
3 RUN npm i npm@latest -g
4 WORKDIR /app
5 COPY package*.json .
6 RUN npm install --no-optional && npm cache clean --force
7 COPY . .
8 RUN npm run build
9
10 # production stage
11 FROM nginx:stable-alpine as production-stage
12 COPY --from=build-stage /app/dist /usr/share/nginx/html
13 EXPOSE 80
14 CMD ["nginx", "-g", "daemon off;"]
```

Listado C.4: Dockerfile de Java

```
1 FROM openjdk:8-jdk-alpine as build
2 WORKDIR /workspace/app
3
4 COPY mvnw .
5 COPY .mvn .mvn
6 COPY pom.xml .
7 COPY src src
8
9 RUN ./mvnw install -DskipTests -Dmaven.test.skip=true
10 RUN mkdir -p target/dependency && (cd target/dependency; jar -xf
.../*.jar)
```

```
11
12 FROM openjdk:8-jre-alpine
13 VOLUME /tmp
14 ARG DEPENDENCY=/workspace/app/target/dependency
15 COPY --from=build ${DEPENDENCY}/BOOT-INF/lib /app/lib
16 COPY --from=build ${DEPENDENCY}/META-INF /app/META-INF
17 COPY --from=build ${DEPENDENCY}/BOOT-INF/classes /app
18
19 EXPOSE 8080
20 ENTRYPOINT ["java","-XX:+UnlockExperimentalVMOptions","-XX:+
    UseCGroupMemoryLimitForHeap","-XX:MaxRAMFraction=1","-
    XshowSettings:vm","-cp","app:app/lib/*","contrast.contrast.
    Application"]
```