Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DE SOFTWARE

APLICACIÓN PARA EL ANÁLISIS VISUAL DE COLECCIONES DE REFERENCIA SOBRE EL PATRIMONIO CULTURAL EXTRAÍDAS DE REDES SOCIALES

Estudiante: Martín Fernández Fernández
Dirección: Patricia Martín Rodilla

A Coruña, Septiembre de 2021.

A mi familia y amigos

Agradecimientos

Quiero agradecer a todos los profesores que he tenido, tanto de la facultad como los que tuve antes, por educarme y ayudar a forjar mi camino para poder llegar hasta aquí. También a la directora de este proyecto, Patricia, por su atención y su gran ayuda para el desarrollo del mismo.

A todos los compañeros y amigos que hice durante esta etapa, que también me ayudaron a estar aquí, también quiero darles las gracias.

Y sobre todo, agradecer a mi familia, en especial a mis padres; y a mis mejores amigos, por estar siempre a mi lado, y ayudarme y apoyarme en todo lo que hago.

Resumen

En este Trabajo de Fin de Grado se ha desarrollado una aplicación web para el Análisis de Colecciones de Referencia sobre el Patrimonio Cultural extraídas de Redes Sociales.

El trabajo está motivado por la cantidad de profesionales de la investigación que utilizan las redes sociales para llevar a cabo su trabajo, encontrando en estas redes grandes fuentes de información de las que obtener colecciones de referencia sobre sus temas de estudio. El análisis de estas colecciones de referencia de manera manual presenta inconvenientes para estos usuarios investigadores, como el tratamiento de lenguaje formal y el alto volumen de datos que pueden representar.

Los profesionales en investigación necesitan tecnologías software de apoyo a su trabajo para solventar estos inconvenientes. Son necesarias herramientas con las que puedan tratar estas colecciones de referencia de manera ágil y que asistan al análisis y visualización de información sobre las mismas.

Recientemente, investigadores de la UDC se han centrado en optimizar los mecanismos de creación de estas colecciones de referencia, extrayendo, por ejemplo, colecciones en temáticas sociales afines a patrimonio cultural, como el movimiento social *Black Lives Matter*. El sistema software que se pretende construir con este proyecto trata de procesar este tipo de colecciones de referencia que están en un formato no legible para un usuario no especialista en sistemas de información, sobre la temática mencionada. El sistema permite la detección, extracción y visualización de entidades nombradas en interfaces de mapas, así como datos y estadísticas a modo de *dashboard* web.

Para el desarrollo del proyecto se ha seguido una adaptación de la metodología ágil Scrum, junto con otras prácticas en desarrollo ágil como Test Driven Development (TDD). Se han utilizado también tecnologías como Java, Spring, Hibernate, JavaScript, React, Redux, etc.; y herrameintas de apoyo al desarrollo como IntellijIDEA, Visual Studio Code, Gradle, Git y Taiga.

Abstract

In this Final Degree Project we have developed a Web Application for the Analysis of Cultural Heritage Reference Collections extracted from Social Media.

The project is motivated by the number of research professionals who use social media as a datasource to conduct research on social and humanities fields, especially in cultural heritage sub-areas, finding in these networks great sources of information from which to obtain reference collections on their study topics. The manual analysis of these reference collections

presents drawbacks for these research users, such as the treatment of formal language and the high volume of data available.

Research professionals need software technologies to support their work to solve these drawbacks. Tools are needed with which they can deal with these reference collections in an agile way and assist in the analysis and visualization of information about them.

Recently, UDC researchers have focused on optimizing the mechanisms for creating these reference collections, extracting, for example, collections on social issues related to cultural heritage, such as the *Black Lives Matter* social movement. The software system that is tried to be built with this project tries to process this type of reference collections that are in a format that is not readable for a non system information speciallist user, on the mentioned subject. The system allows the detection, extraction and visualization of named entities on map interfaces, as well as data and statistics as a web *dashboard*.

For the project development, an adaptation of the Scrum Agile Methodology has been followed, along with other agile development practices such as Test Driven Development (TDD). Technologies such as Java, Spring, Hibernate, JavaScript, React, Redux, etc. have also been used; and development tools such as IntellijIDEA, Visual Studio Code, Gradle, Git and Taiga.

Palabras clave:

- Colecciones de Referencia
- Git
- Hibernate
- Java
- JavaScript
- Patrimonio Cultural
- Procesado de Lenguaje Natural
- React
- Redes Sociales
- Scrum
- Spring
- Taiga

Keywords:

- Cultural Heritage
- Git
- Hibernate
- Java
- JavaScript
- Natural Language Processing
- React
- Reference Collections
- Scrum
- Social Media
- Spring
- Taiga

Índice general

1 Introducción **1**

1.1 Motivación . . . 1

1.1.1 Estado del Arte . . . 2

1.2 Objetivos . . . 3

2 Fundamentos tecnológicos y herramientas utilizadas **5**

2.1 Tecnologías . . . 5

2.1.1 Lado Servidor (*Back-End*) . . . 5

2.1.2 Lado Cliente (*Front-End*) . . . 13

2.1.3 Bases de Datos . . . 18

2.1.4 Herramientas de Desarrollo . . . 19

2.2 Hardware . . . 24

3 Metodología **25**

3.1 Scrum . . . 25

3.1.1 Roles . . . 26

3.1.2 Artefactos . . . 26

3.1.3 Eventos . . . 28

3.2 Desarrollo Guiado por Pruebas (*Test Driven Development*) . . . 29

3.3 *Git-Flow* . . . 30

3.4 Adaptación . . . 31

4 Análisis **33**

4.1 Ámbito de la aplicación y descripción general . . . 33

4.2 Requisitos . . . 34

4.2.1 Actores . . . 34

4.2.2 Historias de Usuario . . . 35

4.3 Modelo de Datos . . . 42

5 Diseño **47**
5.1 Diseño de la Interfaz . . . 47
5.2 Arquitectura . . . 50
5.2.1 Arquitectura del Lado del Servidor . . . 51
5.2.2 Arquitectura del Lado del Cliente . . . 54

6 Planificación **55**
6.1 Planificación inicial . . . 55
6.2 Presupuesto . . . 55
6.3 Seguimiento . . . 56

7 Implementación **57**
7.1 Sprint 1 . . . 57
7.2 Sprint 2 . . . 58
7.3 Sprint 3 . . . 60
7.4 Sprint 4 . . . 62
7.5 Sprint 5 . . . 62

8 Pruebas **65**
8.1 Pruebas de Unidad . . . 65
8.2 Pruebas de Integración . . . 66
8.3 Pruebas de Aceptación . . . 66
8.4 Pruebas de Calidad . . . 66

9 Producto Final **67**
9.1 Inicio de Sesión y Registro . . . 67
9.2 Configuración de la Cuenta . . . 69
9.3 Gestión de Usuarios . . . 69
9.4 Manejo de Colecciones . . . 71
9.5 Análisis de Colecciones . . . 72

10 Conclusiones **77**
10.1 Desarrollo del proyecto . . . 77
10.2 Lecciones aprendidas . . . 78
10.3 Líneas de trabajo futuras . . . 78

Lista de Acrónimos **83**

Glosario **85**

Bibliografía **87**

Índice de figuras

1.1 Memoriales considerados racistas eliminados en el Sur de Estados Unidos desde el asesinato de George Floyd . . . 2

2.1 Arquitectura Modular de *Spring* . . . 8
2.3 Arquitectura en *pipeline* de *CoreNLP* . . . 12

3.1 Flujo de eventos de Scrum . . . 28
3.2 Ciclo de *Test Driven Development* . . . 29
3.3 Diagrama de *Git-Flow* . . . 31

4.1 Estructura de un XML reconocido por la aplicación . . . 34
4.2 Jerarquía de actores del sistema . . . 35
4.3 Modelo de Datos conceptual (Entidad-Relación) . . . 42
4.4 Modelo de Datos Técnico (Tablas de Base de Datos Relacional y Colecciones de Base de Datos No Relacional) . . . 46

5.1 Maqueta de la Lista de Usuarios para el Usuario Administrador . . . 48
5.2 Maqueta de las Listas de Colecciones Propias y Colecciones Compartidas para el Usuario Investigador . . . 48
5.3 Maqueta de la Listas de Colecciones Compartidas para el Usuario Colaborador 49
5.4 Maqueta del Mapa de Entidades de una colección . . . 49
5.5 Maqueta del Dashboard de una colección . . . 50
5.6 Arquitectura Global (Cliente-Servidor) de la aplicación . . . 51
5.7 Arquitectura del Lado del Servidor . . . 52

7.1 Método para generar códigos JWT . . . 58
7.2 *Dispatcher* de acciones de inicio de sesión en Redux . . . 58
7.3 Autoridades del Rol Administrador en la base de datos de MongoDB . . . 59
7.4 Métdo utilizando la anotación *PreAuthorize* para filtrar por rol de usuario . . . 59

7.5 Modelo de Navegación en el *Front-End* . . . 60
7.6 Procesado con CoreNLP . . . 61
7.7 Ejemplo de método que se ejecuta en segundo plano con anotación *Async* . . . 61
7.8 Ejemplo de método con SSE . . . 61
7.9 Renderizado de las entidades en MapBox . . . 62
7.10 Renderizado de gráfico con D3 . . . 63
7.11 Palabras semilla del tesauro original para la extracción de colecciones de referencia sobre *Black Lives Matter* . . . 64
7.12 Búsqueda de términos del tesauro . . . 64

8.1 Ejemplo de error identificado por la extensión *SonarLint* . . . 66

9.1 Página de Inicio . . . 67
9.2 Formularios de Inicio de Sesión y Registro . . . 68
9.3 Página de Inicio de Usuario Identificado . . . 68
9.4 Aviso para que el usuario verifique su cuenta de Email . . . 69
9.5 Menú de Configuración y Cierre de Sesión . . . 69
9.6 Configuración de cuenta . . . 70
9.7 Lista de Usuarios . . . 70
9.8 Formulario para añadir usuario nuevo desde cuenta de administrador . . . 71
9.9 Listas de Colecciones . . . 72
9.10 Formulario para subir colección . . . 73
9.11 Aviso de colección cargada . . . 73
9.12 Página para compartir o dejar de compartir una colección . . . 74
9.13 Vista global del Mapa de Entidades . . . 74
9.14 Vista de detalle de una de las entidades del mapa . . . 75
9.15 Vista del Dahsboard con el Gráfico de Burbujas de Hilos . . . 75
9.16 Vista del Dahsboard con el Gráfico de Tipos de Entidad y Ocurrencias de Términos . . . 76

Capítulo 1

Introducción

En este primer capítulo se comentan las principales líneas maestras de este trabajo, así como la motivación, estado del arte y los objetivos que se persiguen.

1.1 Motivación

Las redes sociales constituyen una reflexión de la realidad política, económica y social. Son muchos los investigadores en humanidades que usan las redes sociales como fuente primaria de sus investigaciones, en campos tan dispares como la sociología, la política o el patrimonio cultural [1] [2]. La extracción, recuperación de información y construcción de colecciones de referencia para estos investigadores desde redes sociales presenta problemas como el tratamiento de lenguaje informal, la recuperación consistente de datos o la selección de metodologías para tal fin. Algunos de estos fueron tratados recientemente, con la producción de colecciones de referencia en diversos ámbitos desde redes sociales que permiten el estudio sistemático de fenómenos sociales [3] [4].

Disponemos de colecciones de referencia extraídas mediante métodos de *pooling* [5] de la red social Reddit, de relevancia con términos relacionados con las revueltas y ataques a entidades patrimoniales derivadas del asesinato de George Floyd y del movimiento *Black Lives Matter*. Estas colecciones presentan formatos *machine-readable* (XML, por ejemplo), que pueden resultar poco amigables para el análisis por parte de investigadores de campos que no pertenecen a las Tecnologías de la Información (TI).

Mediante este proyecto se pretende asistir automáticamente a la detección, extracción y análisis de este tipo de colecciones de referencia, asi como su presentación de manera legible y cómoda para el usuario final investigador, consiguiendo entre otras cosas la visualización en mapas de entidades geográficas mencionadas en los textos; así como también gráficos obtenidos de datos y estadísticas sobre la colección a tratar (empleando técnicas de InfoVis [6]).

1.1.1 Estado del Arte

Derivado del asesinato de George Floyd [7], y por consiguiente, del movimiento social *Black Lives Matter*, se llevaron a cabo oleadas de protestas y de indignación a nivel mundial. Desde entonces se han producido multitud de manifestaciones, revueltas y actos vandálicos, como respuesta al suceso en sí y al racismo subyacente que se le atribuye.

Debido a estas protestas, se han producido ataques y deterioro de entidades patrimoniales relacionadas principalmente con símbolos considerados colonialistas y esclavistas [8], al estar éstos relacionados con un legado de racismo (eg. estatuas de Cristóbal Colón en algunas ciudades han sido objetivo de actos vandálicos [9], o el derribo de la estatua de Edward Colston en Bristol [10]). En la figura 1.1 se puede ver el emplazamiento de algunos memoriales y símbolos racistas que han sido derribados debido a las manifestaciones (esta infografía cuenta con fuentes de información limitadas y no provenientes de redes sociales, en cualquier caso no abordan una visualización general del fenómeno, como sí intentamos en nuestro proyecto).

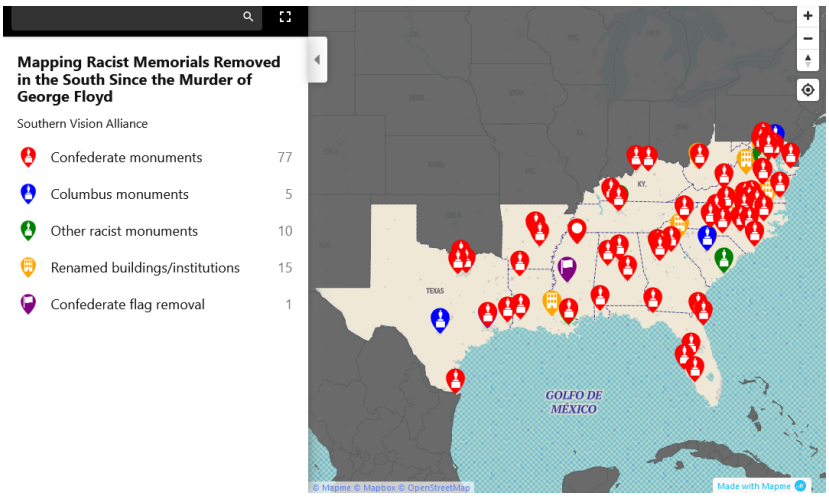


Figura 1.1: **Memoriales considerados racistas eliminados en el Sur de Estados Unidos desde el asesinato de George Floyd**
Fuente: **Southern Vision Alliance** (*https://southernvision.org/topplingracism/*)

Como cualquier otro gran acontecimiento o movimiento de gran importancia social, las redes sociales juegan un papel muy importante en la divulgación de información u opiniones de relevancia con la temática. A medida que se suceden las diferentes revueltas y ataques, se comparte mucha información al respecto en distintas redes sociales [11].

Son muchos los usuarios de habla inglesa de estas redes que escriben comentarios sobre estas revueltas, dónde suceden, el impacto que tienen sobre el patrimonio, etc.; consituyendo estas manifestaciones en redes sociales un verdadero archivo social [4]. El análisis de esta información obtenida y conocer más acerca de las protestas y de los ataques a entidades juega un papel fundamental en la investigación en el campo del patrimonio cultural. Numerosas

investigaciones tratan de obtener colecciones de referencia sobre estas redes sociales, bien de forma manual, o bien utilizando técnicas para una construcción automática sobre estas colecciones como *pooling* [5], cuyo resultado utilizaremos en este proyecto como colecciones de prueba. Se han estado haciendo estudios en la misma temática y con tecnologías similares en otros idiomas, como el español [12].

Por consiguiente, este TFG aborda la problemática de asistir al análisis de estas colecciones, para una detección, extracción y presentación de datos legible para usuarios con un perfil sin experiencia en tecnología [13]. Para realizar estos objetivos es necesario explotar colecciones de prueba con documentos extraídos previamente, que trataremos mediante técnicas de análisis sintáctico (*parsing*) y Procesamiento de Lenguaje Natural (NLP) [14] con el objetivo de extraer entidades patrimoniales y otros datos sobre la colección de referencia, para así mostrar de una manera presentable para el usuario investigador final sobre una interfaz Web.

1.2 Objetivos

El proyecto persigue principalmente **el desarrollo de una aplicación Web para el análisis de colecciones de referencia en el ámbito del patrimonio cultural extraídas de redes sociales**.

Para poder realizar su trabajo, la aplicación necesitará como entrada una de estas colecciones de referencia, que estará en un formato que podrá "leer" (en nuestro caso, los documentos representarán hilos de la red social Reddit, que estarán en formato XML siguiendo una estructura fija para poder identificar todos los elementos necesarios: Hilos, Autores y Posts).

La aplicación recibirá entonces una de estas colecciones como entrada. El usuario dispondrá para ello de un formulario que le permitirá subir colecciones en formato ZIP, conteniendo todos los archivos. Al venir en formato XML representando la estructura de los hilos, lo primero que tendrá que hacer será procesar (mediante algoritmos de "parsing" que adaptaremos a la fuente) cada uno de los archivos XML para obtener la estructura de los hilos. Con los contenidos de los post en texto plano, lo siguiente será utilizar una librería de NLP para extraer entidades nombradas (Reconocimiento de Entidades Nombradas (NER) [15]). El sistema, una vez realizados los pasos previos (carga de la colección), persistirá toda la información obtenida en bases de datos. Las entidades de tipo Localización será persistidas junto con información geográfica (coordenadas) obtenida previamente gracias a una API externa para poder mostrar más adelante.

Disponemos también de un tesauro de palabras semilla sobre la temática que vamos a tratar (*Black Lives Matter*) [5]. El proyecto también incluye la expansión semántica de este tesauro a partir de redes semánticas tipo WordNet [16]. El tesauro expandido final nos ha permitido asociar las entidades obtenidas con los términos relacionados semánticamente con

las primeras palabras semilla, a medida que las fuimos extrayendo.

Se persistirá toda la información obtenida siguiendo un modelo de datos que permitirá identificar todos estos elementos, sus relaciones y la colección con la que fueron subidas. El análisis de la colección una vez subida se compondrá de dos elementos de interfaz principales en la web:

1. Una **interfaz de mapa**, donde el usuario podrá consultar todas las entidades geográficas (tipo Localización), junto a cada una de estas entidades se mostrarán las referencias a los post en los que son citadas, así como hilo y autor.

2. Una **interfaz de *dashboard***, donde el usuario podrá consultar diferentes datos sobre la colección: número de posts totales, número de autores, número de posts eliminados, peso de cada tipo de entidad sobre el total de entidades extraídas, etc.

La aplicación cuenta con 3 roles principales: **Administrador**, **Investigador** y **Colaborador**.

Un usuario **Administrador** tendrá como funcionalidades disponibles las relativas a gestión de usuarios: Podrá dar de alta usuarios nuevos, darlos de baja y consultar una lista de todos los usuarios pertenecientes a los otros dos roles. También podrá cambiar el rol de estos usuarios entre Investigador y Colaborador. No podrá gestionar colecciones.

El usuario **Investigador** tendrá permisos para cargar colecciones como describimos previamente. Una vez cargadas esas colecciones podrá consultar para ellas cualquiera de las dos interfaces habilitadas para el análisis. También podrá eliminar o compartir colecciones que haya subido con otros usuarios Investigadores o Colaboradores para que puedan consultarlas también.

El usuario **Colaborador** sólo podrá consultar las colecciones que le hayan sido compartidas por un rol Investigador. No podrá subir colecciones, ni borarrlas ni modificarlas.

Cualquier usuario dispondrá de un formulario básico para cambiar la contraseña si así lo desea. Los usuarios Investigadores y Colaboradores dispondrán además de una función para darse de baja.

Para realizar cualquiera de todas estas funcionalidades es necesario estar identificado previamente en el sistema. El inicio de sesión consiste en un formulario básico basado en nombre de usuario y contraseña. Para el registro se ha habilitado también un formulario para usuarios no identificados, que permite registrarse como usuario Colaborador. Un usuario administrador podrá registrar usuarios con los roles Investigador o Colaborador, de esta manera un usuario nuevo no podrá ser Investigador por su propia voluntad.

Capítulo 2

Fundamentos tecnológicos y herramientas utilizadas

PARA el desarrollo del proyecto se han utilizado una serie de tecnologías y herramientas que han facilitado la labor de desarrollo. En este capítulo se explican las tecnologías y herramientas utilizadas, los motivos por los que se han escogido y posibles alternativas que hemos tenido en cuenta.

2.1 Tecnologías

Al tratarse de una aplicación cliente-servidor, hemos decidido dividir esta sección en 4 partes: *Back-End* o lado servidor, *Front-End* o lado cliente, bases de datos y herramientas de desarrollo.

2.1.1 Lado Servidor (*Back-End*)

El lado de Servidor se ha construido principalmente utilizando Java y frameworks para facilitar un desarrollo ágil como Spring e Hibernate.

Java

Java [17] es una de las plataformas de desarrollo líderes hoy en día. Fue comercializada por primera vez en 1995 por Sun Microsystems. Utiliza un lenguaje de programación (lenguaje Java) de tipado estático, orientado a objetos, de propósito general y que soporta concurrencia. La sintaxis de este lenguaje deriva principalmente de C y C++, aunque estos dos últimos a bajo nivel tienen más utilidades.

Una de las características que hace único a Java es que su código no es compilado directamente a lenguaje máquina si no que se compila a un lenguaje intermedio conocido como *Java Bytecode*. Este lenguaje puede ser interpretado por la máquina virtual de Java, permitiendo así que esta plataforma sea perfecta para ejecutar aplicaciones en diversas arquitecturas.

Los principales motivos por los que hemos decidido elegir Java como plataforma de desarrollo para el lado del servidor son:

- Nuestro alto conocimiento en el lenguaje de programación, puesto que lo hemos utilizado en muchas asignaturas del grado y actualmente es en el que mayor dominio tenemos.
- Al ser una de las plataformas líderes actuales, y estar respaldada por una enorme comunidad, resulta muy fácil encontrar soporte y soluciones a muchos problemas que se presentan en desarrollo.
- Sus características permiten cubrir sin problemas las necesidades que demanda nuestro sistema a desarrollar.
- Posibilidad de integrarse con multitud de herramientas Open-Source también muy utilizadas y que utilizaremos en el proyecto como Gradle, y frameworks como Spring e Hibernate muy utilizados junto con la plataforma.

Hemos elegido para el desarrollo la **versión del JDK 8**, que introduce respecto a versiones anteriores el uso de *lambda expressions* y *streams*, las cuales permiten utilizar programación funcional para resolver algunos pequeños problemas como la iteración de listas o conjuntos de una manera más legible y eficiente.

La principal alternativa a Java (y que también hemos barajado) es .NET de Microsoft, otra de las plataformas líderes en desarrollo Web y Multiplataforma. La principal diferencia es que .NET es multilenguaje y permite escribir código en diversos lenguajes como C# o VBasic. Aún así hemos decidido descartarla principalmente porque Java es mejor en los aspectos que hemos comentado.

Spring

Spring [18] es un framework de desarrollo de aplicaciones de código abierto para la plataforma Java. Spring proporciona una infraestructura y un contenedor de objetos (basado en el

principio de Inversion of Control, IoC) destinados a abstraer al programador de tareas repetitivas o tediosas como la gestión de dependencias, o facilitando la aplicación de principios como la Programación Orientada a Aspectos (AOP): gracias al uso de anotaciones se puede evitar escribir directamente código relativo a gestión de transacciones de las bases de datos o gestionar el control de acceso de los usuarios. Mientras Spring reduce el código y el trabajo relativo a estas tareas, el desarrollador se puede centrar en el desarrollo propio de la aplicación, lo que se traduce en un desarrollo mucho más ágil y un código menos repetitivo.

Entre otras cosas, el framework proporciona:

1. Un modelo de gestión de dependencias (basado en IoC), mediante anotaciones o configuración con archivos XML. Esto permite una inyección de las dependencias en el proyecto de manera flexible y cómoda, reduciendo el acoplamiento entre objetos, ya que evitamos, por ejemplo, llamar directamente a las implementaciones de los componentes.

2. Soporte para Programación Orientada a Aspectos (AOP) mediante anotaciones, que permiten aislar el código referente a aspectos (i.e. gestión de transacciones, seguridad y control de acceso, configuración, etc.) de la lógica propia del sistema en desarrollo.

3. Implementaciones para especificaciones importantes, por ejemplo mediante Spring Data Access/Integration podemos implementar Java Database Conectivity (JDBC) o Java Persistence API (JPA).

4. Soporte o integración de alto nivel para otros frameworks como Hibernate.

5. Soporte y funcionalidades para facilitar la construcción de interfaces Web o APIs REST siguiendo el patrón Modelo Vista Controlador (MVC).

6. Módulo de Testing para construir pruebas de integración de manera cómoda simulando el entorno de la aplicación real.

Spring Framework sigue una arquitectura modular (véase figura 2.1), organizado aproximadamente en 20 módulos, agrupados en:

- Core Container
- Data Access/Integration

- Web
- AOP
- Instrumentation
- Test

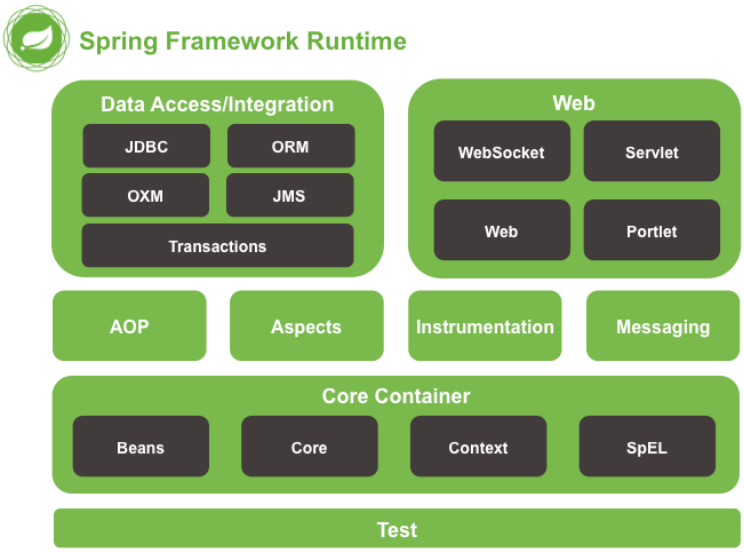


Figura 2.1: **Arquitectura Modular de *Spring***
Fuente: **Spring Documentation** (*https://docs.spring.io/spring-framework/docs/4.0.x/spring-framework-reference/html/overview.html*)

Hemos decidido utilizar Spring principalmente por ser el Framework más popular para el desarrollo web ágil en Java, y contar (al igual que la propia plataforma Java) con una gran comunidad de desarrolladores dando soporte. Utilizaremos las siguientes características de Spring:

- Contenedor IoC para inyección de dependencias.
- Módulo AOP para gestión de transacciones de bases de datos.
- *Spring Data* junto con Hibernate para el acceso a base de datos y mapeado objeto-relacional.
- *Spring Web* para el desarrollo del API Rest que permita comunicar con el cliente.
- *Spring Security* para abstracción de los procesos relacionados con la seguridad: Autenticación, Gestión de Roles y Control de Acceso.
- *Spring Test* para simular un entorno de producción en las pruebas de integración.

Spring Boot

Spring Boot [19] es una tecnología dentro del ecosistema de Spring que simplifica aún más el desarrollo con este framework. Permite la cómoda integración con herramientas de gestión del proyecto y dependencias (como Maven o Gradle), ya que agrupa los módulos de Spring y sus dependencias con terceros en unos módulos propios de Spring Boot conocidos como *starters*. De esta manera, importando sólo los *starters* necesarios, Spring Boot se encargará de configurar las dependencias (tanto de Spring como de terceros) necesarias para el desarrollo. Por ejemplo, especificando únicamente en Maven (o Gradle) el starter de Spring Web, se importarán todos los módulos necesarios para trabajar con el grupo de módulos de Web de Spring, incluido MVC y también los parsers de JSON necesarios (Jackson) para el mapeado de peticiones y respuestas en este formato. Esto reduce el trabajo y el código de configuración necesario en los scripts de la herramienta de automatización de construcción utilizada.

Además de esto, Spring Boot facilita la ejecución de la aplicación Web. Permite empaquetar la aplicación de tipo servicio en un archivo JAR en vez de un WAR o un EAR, de esta manera se puede ejecutar el servidor como si fuera un programa Java convencional, ahorrando tener que configurar un servidor Web (como Jetty o TomCat) para realizar pruebas. Además se puede ejecutar el JAR en entornos de producción preparados, por ejemplo, con Docker.

También permite, mediante un archivo *application.properties* o *application.yml*, una configuración de Spring más sencilla mediante variables ya definidas, o la posibilidad de definir nuevas variables para ultilizar en la aplicación.

Por último, se integra muy fácil con los módulos de Testing de Spring, de esta manera podemos anotar una clase de tests de integración con la anotación *SpringBootTest*, y cada uno de esos test se ejecutará como si fuera una aplicación Spring Boot independiente, facilitando la ejecución de pruebas de integración y la independencia entre éstas; manteniendo el entorno de producción o desarrollo, o permitiendo configurar un entorno específico para estas pruebas.

Aprovechando que utilizamos Spring, hemos decidido también utilizar Spring Boot debido a todas estas facilidades que ofrece. En concreto hemos optado por la versión *2.3.9.RELEASE* de los *starters*.

Hibernate

Hibernate [20] es una herramienta de código abierto que permite el mapeado de entidades de bases de datos a objetos: Mapeado Objeto-Relacional (ORM). Está disponible para la plataforma Java y también para .NET (para éste último bajo el nombre de NHibernate). Mediante anotaciones o configuración XML (en nuestro caso utilizaremos la primera) es posible configurar el mapeado y así abstraer al desarrollador del proceso de adaptar las entidades de bases de datos a objetos manejables en Java.

Hibernate implementa el estándar JPA, una especificación de Java para persistir objetos como entidades de bases de datos relacionales.

A parte del ORM, Hibernate también dispone de otras características, como un validador que implementa el estándar de Java (*javax.validation:validation-api*) o un Mapeado Object/-Grid (OGM) para bases de datos no relacionales. Hibernate OGM proporciona soporte de JPA para bases de datos NoSQL (como MongoDB).

Utilizaremos Hibernate ORM e Hibernate Validation por su facilidad de uso y nos ahorrará tareas recurrentes de mapeado de entidades de bases de datos relacionales. Además, junto con Spring Data, facilitará y agilizará mucho el desarrollo de toda la parte relativa a acceso a datos. La versión escogida será la *5.4.28.Final*, al ser la que ya viene integrada en el *starter* de Spring Data.

Lombok

Lombok [21] es una librería que actuará de precompilador para nuestro código Java. Permite, mediante una mínima configuración y unas cuantas anotaciones, reducir código repetitivo considerado *boilerplate*.

Algunos ejemplos de lo que permite esta librería y que hemos utilizado son:

- Con la anotación *@Data* sobre una clase y con los correspondientes atributos privados, automáticamente se generan los accesores (*getter* y *setter*) correspondientes, así como

un constructor para inicializar todos los atributos, y se sobreescribirán automáticamente los métodos *equals* y *hashcode*.

- Mediante las anotaciones *@Getter* y *@Setter* sobre la clase o sobre un atributo en particular es posible definir esos accesores para todos los atributos o para el atributo en concreto, respectivamente.

- Mediante la anotación *@EqualsAndHashcode* es posible sobreescribir los métodos *equals* y *hashcode* sin necesidad de hacerlo explícitamente. Lombok generará una implementación efectiva y eficiente para estos métodos, basada en los atributos de la clase.

- La anotación *@SneakyThrows* evita tener que declarar explícitamente que excepciones puede lanzar un método.

Hemos decidido utilizar Lombok debido a estas características, que facilitarán el proceso, sobre todo, de escribir código de DTOs. La versión escogida ha sido la *1.18.16*.

El único inconveniente, es que es necesario instalar un plugin de esta librería en el IDE a utilizar para que éste reconozca la configuración y anotaciones de Lombok, y no detecte errores de compilación al no declarar los accesores. Los principales IDEs, por ejemplo, *Eclipse* e *IntellijIDEA* disponen de su correspondiente plugin para la librería.

Algoritmos de NLP: Stanford CoreNLP y Apache OpenNLP

Para procesar los textos en inglés de la colección, dividir en oraciones y extraer las entidades nombradas (NER), es necesario utilizar algoritmos de Procesamiento de Lenguaje Natural (NLP). Para este caso hemos decidido probar dos alternativas diferentes: la solución de la Universidad de Stanford, **CoreNLP**; y la de Apache, **OpenNLP**.

Stanford **CoreNLP** [22] es una librería escrita en y para la plataforma Java, y mantenida por la propia Universidad de Stanford. Soporta multitud de funcionalidades propias de su campo como: división en palabras y oraciones, análisis sintáctico, Reconocimiento de Entidades Nombradas (NER), identificación de dependencias sintácticas, coreferencia, etc. Para ello utiliza una arquitectura basada en filtros o *pipeline*, por la que se divide el procesado en varias fases según el nivel que se quiera alcanzar de éste, y la salida de cada fase servirá de entrada

para la siguiente (véase figura 2.3). Al final del procesado devuelve un objeto (documento anotado) en el que se puede consultar todos los elementos obtenidos a través de llamadas simples. CoreNLP soporta actualmente 6 idiomas: Árabe, Chino, Inglés, Francés, Alemán y Español.

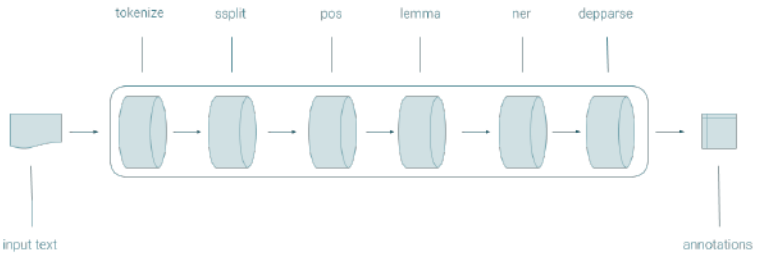


Figura 2.3: **Arquitectura en *pipeline* de *CoreNLP***
Fuente: **CoreNLP Site** (*https://stanfordnlp.github.io/CoreNLP/*)

Apache **OpenNLP** [23] es una librería basada en aprendizaje máquina para realizar las tareas más comunes de NLP. También está escrita para la plataforma Java y es de código abierto. Utiliza una serie de modelos para llevar a cabo cada una de las tareas de NLP que soporta: división de palabras, división de oraciones, NER, etc. No todos los modelos están disponibles para todos los idiomas, pero para inglés sí. Hay que ejecutar cada uno de los modelos de manera separada, a diferencia de CoreNLP, el cuál puede lanzar el *pipeline* de manera que haga todas las operaciones.

Hemos probado las dos alternativas y hemos construido dos módulos en el proyecto para poder procesar textos en inglés con cualquiera de las dos opciones, aunque la versión del producto final utiliza CoreNLP porque en las pruebas que hemos realizado ha sido capaz de extraer más entidades y obtener unos resultados más fiables que la alternativa de Apache.

JUnit

JUnit [24] es un popular framework para escribir pruebas de unidad para aplicaciones Java. Permite especificar en código pruebas llamadas *de caja negra*, esto es, se prepara un caso de prueba para el componente, módulo o método a probar, éste recibe una(s) entrada(s) y se evalúa la correspondiente salida, sin evaluar para nada internamente el código a probar.

Un test de JUnit se compone de una o varias sentencias a evaluar una vez ejecutado el código a probar. Si todas las condiciones son evaluadas con éxito y el resultado obtenido es el esperado, el test será satisfactorio (*passed*), en caso contrario se considerará fallido (*failed*). Cualquier error previo o durante la ejecución del código a probar también producirá un error en el test.

Utilizaremos JUnit debido a su popularidad como framework para pruebas de Java, además de ser el más utilizado durante el grado y del que más dominio disponemos. En concreto usaremos la versión Junit5 (*JUnit Platform* + *JUnit Jupiter* + *JUnit Vintage*). Una alternativa a JUnit puede ser TestNG, que está basado en JUnit y que añade nuevas funcionalidades, pero JUnit es suficiente para el trabajo a desarrollar.

Mockito

Mockito [25] es otro popular framework utilizado para pruebas en Java. Permite realizar **pruebas de unidad** mediante *mocking*, esto es, permite aislar los módulos de sus dependencias sustituyendo éstas por versiones simuladas de las mismas que podemos hacer que devuelvan el resultado esperado de estas llamadas. De esta manera el resultado del test no depende de estas llamadas y se centrará en validar el código del módulo a probar. Esto es lo que diferencia una prueba de unidad de otras pruebas como las de integración o de sistema.

Junto con otros frameworks como JUnit se pueden escribir pruebas de unidad efectivas y legibles. A parte de las verificaciones de JUnit, Mockito dispone de métodos de verificación para saber si el componente a probar ha llamado un número concreto de veces a las funciones de los objetos *mock*. Utilizaremos Mockito junto con JUnit por las facilidades que nos brinda para poder realizar las pruebas de unidad de manera efectiva.

2.1.2 Lado Cliente (*Front-End*)

El lado de Cliente se ha construido principalmente utilizando tecnologías basadas en JavaScript: se ha utilizado el estándar ECMAScript6 de este lenguaje, junto con la librería React (utilizando Hooks) y Redux para el manejo del estado de la aplicación.

JavaScript (ECMAScript 6)

JavaScript [26] es un lenguaje dinámico, interpretado (o compilado *just-in-time*) y ligero. Soporta los principales paradigmas de programación (imperativo, funcional y orientado a objetos) y emplea funciones de primera clase (las funciones en JavaScript son tratadas como variables), lo que facilita el uso de callbacks, o funciones llamadas a posteriori.

JavaScript puede ser interpretado por un navegador Web, como Firefox o Chrome, o mediante un motor de ejecución para el lenguaje, como Node.js. Este útimo puede ser útil en caso de aplicaciones JavaScript que renderizen en el lado del servidor, o en el caso de que se quiera utilizar el lenguaje también para desarrollar el *Back-End* (utilizando Express, por ejemplo).

ECMAScript 6 es el estándar de JavaScript que se lleva utilizando desde 2015 (por esta razón también se conoce como ECMAScript 2015). Introdujo novedades en la sintaxis orientadas a una mayor limpieza y claridad del código [27].

Una alternativa a JavaScript puede ser el lenguaje TypeScript, un lenguaje creado por Microsoft en 2012 que deriva de JavaScript y que introduce un tipado estático, a diferencia de JavaScript que es dinámico. Este lenguaje se compila a JavaScript y es considerado como un superset de éste, añadiendo nuevas características a JavaScript. A pesar de esta clara ventaja, hemos optado por JS por nuestro mayor conocimiento sobre él.

React

React [28] es una popular librería de código abierto de JavaScript para desarrollar interfaces de usuario. Está mantenida por Facebook y por una enorme comunidad de desarrolladores. Se centra en facilitar el desarrollo de Single-Page Applications o Aplicaciones de una Sóla Página (SPA). Una SPA es una aplicación en la que inicialmente se renderiza una única página en HTML y según se interactúa con ésta se va actualizando, utilizando técnicas como Asynchronous JavaScript and XML (AJAX). Esto mejora considerablemente el rendimiento de la aplicación, ya que sólo se cargan aquellas partes que son necesarias en el momento adecuado.

React introduce mecánicas que evitan que tengamos que manipular HTML, CSS o JavaScript de bajo nivel, como la sintaxis JSX, que nos permite especificar el renderizado de un componente de forma declarativa. La librería está basada en componentes encapsulados y reutilizables. Como la lógica de los componentes está escrita en JavaScript, resulta sencillo pasar datos a través de la aplicación y mantener el estado fuera del DOM.

A partir de la versión *16.8*, React incorpora *Hooks*, una nueva mecánica que nos permite escribir componentes y manejar su estado y otras características de éstos sin necesidad de escribir clases, a través de funciones. Hemos optado por utilizar la versión *17.0.2* de React para poder utilizar React Hooks. Alternativas a React pueden ser por ejemplo AngularJS, otro framework para crear SPAs mantenido por Google. Las dos librerías son potentes y se ofrece bastante soporte por parte de las comunidades de desarrolladores, pero hemos optado por utilizar React por tener cierta familiaridad.

Redux

Redux [29] es un contenedor predecible y ligero del estado de aplicaciones de JavaScript. Ayuda a escribir aplicaciones que se comportan de manera consistente, corren en distintos entornos, y son fáciles de probar. Proporciona una gran experiencia de desarrollo. Se puede utilizar Redux combinado con React, o con cualquier otra librería de JavaScript para *Front-End*.

Su arquitectura se basa en funciones llamadas reductores y acciones. Los reductores se combinan en un reductor raíz que se asigna a un *store*, que es el componente que mantiene el estado de la aplicación. Las acciones son bloques de información que se envían desde la aplicación al *store*; dependiendo de la acción que se haya enviado el *store* actualizará su estado convenientemente.

Hemos decidido utilizar Redux debido a que nos proporciona una manera flexible de gestionar el estado de nuestra aplicación, por ejemplo, manteniendo la sesión del usuario o la información de la página en la que se encuentra actualmente el usuario (para mostrar el nombre de la página en otros componentes).

Materialize CSS

Materialize [30] es un moderno y popular framework de CSS y JavaScript, que se utiliza para crear interfaces Web *responsive* de manera sencilla. Permite ahorrar trabajo predefiniendo unos estilos y componentes visuales (modales, barras de navegación, etc.) y reducir así las

hojas de estilo que tenemos que escribir por nuestra cuenta para nutrir a nuestra apliación de un estilo apropiado para la interfaz.

Está basado en Material Design [31], un sistema de diseño creado por Google para ayudar a construir interfaces con una gran experiencia de usuario.

La alternativa más conocida a Materialize es Bootstrap, framework mantenido por Twitter, que es el framework de CSS para diseño de interfaces más popular. A pesar de que hemos utilizado Bootstrap en más ocasiones y Materialize es nuevo para nosotros, hemos optado por esta segunda opción para el proyecto con el objetivo de probar una tecnología nueva.

SASS

Syntactically Awesome Style Sheets (SASS) [32] es un sistema que permite extender la sintaxis de CSS a un nuevo nivel. Permite utilizar variables, reglas anidadas, *mixins*, funciones, etc., evitando código duplicado en las hojas de estilo. Se compila a CSS nativo y muchos frameworks de CSS (por ejemplo Bootstrap y Materialize) ya incluyen soporte para SASS y sus propias hojas de estilos en este formato.

Debido a que simplifica la definición de estilos propios para la interfaz y ya se integra bien con Materialize CSS, hemos optado por utilizar SASS.

MapBox

MapBox [33] es un cojunto de servicios de información geográfica para dar soporte a aplicaciones Web. Contiene, entre otros, renderizado de mapas, vectores, API de geocodificación, navegación, atlas, etc.

Utilizaremos MapBox debido a sus capacidades para abordar nuestro problema. A pesar de que el plan gratuito es limitado, para nuestras pruebas es suficiente. En concreto usaremos su librería de JavaScript (MapBoxGL) para renderizar los mapas de nuestra aplicación, y su API REST de geocodificación para resolver las coordenadas de las entidades de tipo localización. Alternativas conocidas pueden ser el Servicio de Google Maps y los servicios de OpenStreetMaps.

D3

D3 (Data-Driven Documents) [34] es una librería de JavaScript para la visualización de documentos gráficos interactivos y dinámicos en aplicaciones web, a partir de grandes cantidades de datos.

Utilizaremos D3 porque nos permitirá de manera sencilla mostrar gráficos para poder visualizar información en el *dashboard* de las colecciones. Una alternativa conocida a D3 puede ser Chart.js.

Herramientas de Testing de JavaScript: Jest y Testing Library

Para las pruebas en el lado del cliente haremos pruebas de unidad de cada componente haciendo uso de **Jest** junto con **Testing Library**.

Jest [35] es un framework de JavaScript para pruebas, que se enfoca en la simplicidad de éstas y permite escribir pruebas de unidad rápidamente gracias a su API sencilla, y es capaz de

construir objetos simulados (*mocks*) de manera fácil en JavaScript. Alternativas a Jest pueden ser Mocha o Jasmine.

Testing Library [36] es una librería de JavaScript que permite evaluar componentes del árbol DOM, de manera que se pueden hacer sentencias de evaluación sobre objetos de la interfaz Web. En conjunción con otras herramientas de pruebas como Jest permite construir pruebas efectivas sobre la interfaz. Una alternativa popular a Testing Library es Enzyme.

2.1.3 Bases de Datos

Hemos optado por utilizar dos Bases de Datos para el dominio del proyecto: una Relacional (MySQL) y otra No Relacional (MongoDB), debido principalmente a temas de simplicidad y eficiencia para poder implementar correctamente el modelo de datos (véase sección 4.3).

MySQL

MySQL [37] es un Sistema de Gestión de Bases de Datos (SGBD) relacional. Está desarrollado y mantenido bajo una licencia dual (Licencia Pública General/Licencia Comercial) por Oracle Corporation. Es considerada la base de datos de código abierto más popular del mundo, junto con Oracle y MySQL Server de Microsoft. Otra posible alternativa, muy popular también, podría ser PostgreSQL.

El software de MySQL proporciona un rápido, robusto y concurrente servidor de bases de datos basado en Structured Query Language (SQL), con soporte para multitud de usuarios.

Las principales razón de haber optado por MySQL, aparte de las ya mencionadas, son su facilidad para integrarse con el resto de las herramientas que utilizaremos y su facilidad de instalación y mantenimiento en nuestro sistema, pudiendo crear bases de datos de manera sencilla tanto para entorno de pruebas, para desarrollo o para producción. Hemos optado por la versión 5.7, una versión ya algo antigua pero muy estable.

MySQL permite utilizar diversos motores de almacenamiento, siendo los más populares MyISAM (muy veloz, ideal para operaciones de sólo lectura) e InnoDB (soporte para atomicidad, ACID). Como realizamos todo tipo de operaciones contra la base de datos, no sólo operaciones de sólo lectura, utilizaremos el segundo ya que soporta transacciones ACID, las cuáles serán de vital importancia en nuestro proyecto.

MongoDB

MongoDB [38] es el SGBD NoSQL más conocido y utilizado. Está diseñado para facilitar el desarrollo y la escalabilidad de aplicaciones. Es una base de datos distribuida y de uso general para desarrolladores de aplicaciones modernas.

MongoDB está basado en documentos, estructuras de datos en formato BSON, que siguen un esquema dinámico, de esta manera es posible persistir entidades (en este caso documentos) sin tantas restricciones como en una base de datos relacional, lo que posibilita una mejor integración de algunos datos. A diferencia de las bases de datos relacionales, que persisten las entidades como filas de tablas y con relaciones entre éstas, los documentos en formato BSON en esta base de datos se persisten en colecciones.

Utilizaremos MongoDB debido a que para persistir algunos tipos concretos de datos creemos que es más cómodo que utilizar una base de datos relacional como MySQL. Hemos utilizado la versión *4.2* del MongoDB Server, ya que respecto a versiones anteriores trae como novedad el soporte para transacciones ACID, las cuáles son importantes para lograr la consistencia de los datos junto con los que se persistirán en la base de datos relacional (MySQL).

2.1.4 Herramientas de Desarrollo

Hemos utilizado diversas herramientas para poder facilitar el desarrollo del proyecto: IDEs para *Back-End* como IntellijIDEA, editores de texto para JavaScript como Visual Studio, herramientas de automatización de construcción como Gradle, etc.

IntellijIDEA

IntellijIDEA es, quizá junto con Eclipse y Netbeans, uno de los Entornos de Desarrollo Integrado (IDE) más populares para la plataforma Java. Es desarrollado y mantenido por la empresa JetBrains y, a pesar de no ser código abierto, dispone de una buena cantidad de plugins destinados a mejorar la productividad con esta herramienta [39]. De esta manera podemos tener

soporte para integración del sistema de SCM, autocompletado, autogeneración y organización de código, etc. A parte trae de forma nativa un potente sistema de depuración (*debug*) que facilitará la resolución de errores durante el desarrollo.

Hemos decidido utilizar este IDE para la parte de *Back-End* en lugar de otros como Eclipse, por tener una mejor familiaridad con este IDE y por considerarlo más productivo según nuestro punto de vista personal. El IDE dispone de una versión gratuita para la comunidad pero con capacidades más limitadas y de una versión más comercial; de las cuáles utilizaremos la primera al ser suficiente para el desarrollo del problema.

Visual Studio Code

Visual Studio Code es un editor de código gratuito desarrollado y mantenido por Microsoft. Soporta una gran cantidad de lenguajes por lo que es ideal para editar aquel código de nuestro proyecto que no sea Java. Tiene además un enorme *marketplace* con extensiones que pueden facilitar el integrar nuevas herramientas y lenguajes, así como añadir soporte a determinadas sintaxis [40].

Hemos optado por este editor de código porque es perfecto para el desarrollo del *Front-End* gracias a su soporte nativo de JavaScript y JSON y la cantidad de extensiones que tiene para añadir soporte para React, hojas de estilos, etc. Alternativas populares a esta herramienta pueden ser Atom o SublimeTxt. Hemos optado por VSCode debido a nuestra experiencia previa con él.

Git y GitHub

Git [41] es el Sistema de Control de Versiones (VCS) más utilizado. Ha sido diseñado por Linus Torvalds (desarrollador conocido por iniciar y mantener el Kernel Linux) para ayudar a manejar el código fuente desde pequeños hasta grandes proyectos, así como ayudar y fomentar el

trabajo en equipo. Es un VCS distribuido, a diferencia de otras alternativas como SubVersion, esto quiere decir que el usuario sincroniza su trabajo con un repositorio local en lugar de con un repositorio remoto.

GitHub [42] es una plataforma de desarrollo colaborativo que fue creada para alojar proyectos de software utilizando Git. Actualmente es la plataforma más importante de colaboración para proyectos Open-Source. Está pensado para tener repositorios de software públicos que cualquiera pueda clonar, aunque recientemente han incorporado la posibilidad de tener repositorios privados gratuitos. Varias alternativas a GitHub que utilizan también Git como VCS pueden ser GitLab o Bitbucket (Atlassian).

Hemos decidido utilizar Git frente a otras alternativas por las ventajas de ser distribuido, lo que nos permite trabajar en local cómodamente sin tener conexión todo el tiempo y poder gestionar mejor nuestro trabajo, y también por nuestro gran conocimiento con esta herramienta, aunque hemos utilizado SubVersion más veces durante el Grado. También hemos optado por un repositorio privado de GitHub para alojar nuestro código.

Gradle

Gradle [43] es una herramienta de automatización de construcción (*build automation*) del software, con soporte para distintas plataformas y lenguajes de programación. Permite un ciclo de vida completamente personalizable basado en tareas o *tasks*, incluyendo entre ellas la gestión de artefactos o dependencias, compilación, ejecución de pruebas automáticas, empaquetado y despliegue.

Es posible definir la configuración y el ciclo de vida que usará Gradle para el proyecto utilizando para ello scripts escritos en lenguaje Groovy o Kotlin DSL. Además es posible incluir la distribución de Gradle utilizada con ayuda de *Gradle Wrapper*, una herramienta que nos permite empaquetarla junto con el proyecto y hace que este pueda ejecutar Gradle en cualquier entorno sin necesidad de tener una distribución de Gradle preinstalada.

La principal alternativa de Gradle es Apache Maven, la principal y más utilizada herramienta de *build automation* para Java, que utiliza scripts XML para definir la configuración. A pesar de tener Maven una mayor documentación y soporte, y nosotros tener mejor dominio con ella que con Gradle, nos hemos decantado por Gradle por probar una tecnología nueva. Además, la posibilidad de utilizar scripts con un lenguaje de programación como Groovy o

Kotlin DSL en lugar de un lenguaje de marcado como XML, permite definir la configuración de la herramienta de una manera más compacta y legible.

Utilizaremos un *Gradle Wrapper* con una distribución de la versión *5.6.4*. Nuestros scripts de configuración estarán escritos utilizando Kotlin DSL en lugar de Groovy. Groovy tiene más documentación y soporte pero Kotlin permite tipado, lo que lo hace menos propenso a errores de compilación.

NPM

Node Package Manager (NPM) [44] es el sistema de gestión de paquetes por defecto de Node.js, entorno de ejecución de JavaScript. Permite gestionar las dependencias que utilizaremos en el *Front-End* con JavaScript. Los proyectos que utilizan NPM disponen de un archivo *package.json* donde aparte de otra información del proyecto también están declaradas las dependencias junto con sus versiones.

Junto a Webpack, será la herramienta de *build automation* que utilizaremos para el cliente. NPM permite también ejecutar algunos paquetes sin descargarlos utilizando la herramienta embebida *npx*.

Una alternativa a NPM es Yarn, aunque hemos decidido decantarnos por NPM por mayor familiaridad y conocimiento de ésta. Hemos utilizado la versión *6.14.4*.

Webpack

Webpack [45] es un empaquetador de módulos para aplicaciones JavaScript modernas. Permite el empaquetado (*bundle*) del código JavaScript, de manera que puede agrupar todas las dependencias y el código de nuestra aplicación en un "producto final".

A partir de la versión 4.0, Webpack no necesita un archivo de configuración, sin embargo, es sorprendentemente fácil y recomendable realizar ésta. La configuración permite definir varios entornos como producción, para empaquetar la aplicación para ejecutarla en un contenedor posteriormente de una manera eficiente (despliegue), o un entorno de desarrollo, que

permite *source maps* o mapeadores de código fuente para permitir el debug. También posibilita incorporar diversos plugins como Babel, que permite el compilado de JavaScript en E6 a JavaScript nativo para poder ser interpretado por el navegador.

Hemos utilizado la versión *5.35.0*.

SonarQube

SonarQube [46] es una herramienta de inspección continua de código abierto. Consiste en una plataforma que permite evaluar el código fuente de un proyecto, detectando bugs, vulnerabilidades y código duplicado, entre otros. También permite configurar diversas métricas para la calidad de los proyectos, como por ejemplo la cobertura de tests. Dispone de gran cantidad de extensiones para incrementar sus funcionalidades y tiene soporte para multitud de lenguajes y frameworks.

Es una herramienta muy utilizada que los equipos de desarrollo suelen integrar con otras herramientas de integración continua para un *workflow* automatizado.

Hemos utilizado SonarQube para medir la calidad de nuestro código y detectar errores, así como medir la cobertura de las pruebas. Como es una plataforma la hemos desplegado en un contenedor docker a modo de servicio. Para facilitar el uso de SonarQube, los principales IDEs disponen de la extensión *SonarLint*, que permite desde el IDE sincronizar con el servidor de SonarQube y realizar análisis en local.

Hemos utilizado la versión de SonarQube*8.5* junto con la extensión de SonarLint para IntellijIDEA.

Herramientas de Diseño: draw.io y Pencil

Como herramientas para facilitar la fase de diseño de la aplicación Web hemos decidido utilizar draw.io y Pencil.

draw.io [47] es una herramienta para la creación de diagramas basados en UML. Dispone de una versión Web así como versiones para diversos sistemas operativos incluido Windows. Nosotros utilizaremos esta última.

Pencil [48] es una herramienta para prototipado y creación de maquetas de interfaces de usuario. Es muy simple e intuitiva de utilizar.

Taiga

Taiga [49] es una herramienta de gestión de proyectos software online. Su división natural en *sprints* hace que esté pensada para trabajar con la metodología Scrum. Permite registrar hasta un proyecto privado gratuito por usuario. Tiene soporte para gestión de historias de usuario, tareas y *sprints*, y permite visualizar información sobre el estado de los mismos.

Al ser una herramienta sencilla de utilizar y estar pensada para trabajar con Scrum, se considera suficiente para nuestro proyecto, así que nos hemos decantado por ella. Posibles alternativas libres podrían ser Redmine, o también algunas más comerciales como JIRA de Atlassian.

2.2 Hardware

Los recursos hardware y servicios utilizados en este proyecto han sido los siguientes:

- Ordenador de Sobremesa **HP EliteDesk 800 G4 TWR**. Caracteríticas:
  - Procesador **Intel(R) Core(TM) i5-8500 CPU @ 3.00GHz**
  - Memoria RAM **32GB** DDR4
  - Sistema operativo **Windows 10 Pro** con arquitectura de **64 bits**
- Acceso a **Internet de banda ancha**

Capítulo 3
Metodología

En este capítulo se introduce la metodología de trabajo seguida durante el desarrollo del proyecto: Una adaptación de la metodología ágil *Scrum*, cogiendo aspectos de *eXtreme Programming* como Test Driven Development (TDD) o *Desarrollo Guiado por Pruebas*. También se ha empleado la mecánica de *Git-Flow* para trabajar con el VCS Git.

3.1 Scrum

Scrum [50] es un marco de trabajo o conjunto de procesos que permite abordar problemas complejos adaptativos, a la vez que permite entregar productos del máximo valor posible tanto productiva como creativamente. Es una metodología ágil, y cómo tal y al igual que otras metodologías de esta característica se apoya en el *Manifesto for Agile Software Development* [51]; que establece que una metodología de esta característica debe seguir estos 4 principios: (1) **Individuos e iteraciones** *sobre* procesos y herramientas; (2) **Software funcionando** *sobre* documentación extensiva; (3) **Colaboración con el cliente** *sobre* negociación contractual; y (4) **Respuesta ante el cambio** *sobre* seguir un plan.

Con Scrum se pretende entregar valor a un proyecto de manera iterativa e incremental siguiendo estos principios, para ello, la metodología se basa en pequeñas iteracciones de duración constante llamadas *sprints*, de una duración no menor a 1 semana y no mayor a 4. El cliente o *stakeholders*, a través de la figura del propietario del producto o *Product Owner* definirán y priorizarán los diferentes problemas a abordar y el trabajo a realizar. Será el *Scrum Master* junto con el equipo de desarrollo el encargado de definir cada *sprint* con las tareas a abordar.

Algunas de las ventajas de aplicar Scrum a nuestro proyecto podrían ser las siguientes:

- Al ser una metodología enfocada en objetivos a corto plazo, el equipo de desarrollo puede centrarse en llevar las tareas a cabo para la iteracción o *sprint* actual, aumentando la productividad de éste.

- Al entregar valor al cliente con cada iteracción, éste puede hacer una evaluación constante durante todo el ciclo del proyecto. Se flexibiliza así la introducción de posibles cambios de requisitos y facilita la anticipación a éstos.
- La división en iteracciones bien definidas permite conocer la velocidad media del equipo de desarrollo, por consiguiente resulta más sencillo estimar las tareas a abordar en cada uno de los *sprints*.
- Al no estar tan orientada a procesos, y con ello eliminar la documentación excesiva propia de otras metodologías para centrarse en el valor del producto, permite ahorrar tiempo y esfuerzo y centrarse propiamente en lo que es el desarrollo del producto en sí.

3.1.1 Roles

En la metodología Scrum participan los siguientes roles o actores:

- **Propietario del producto (*Product Owner*)**: Se encarga de representar al cliente o interesados (*stakeholders*) de cara al equipo de desarrollo, y viceversa. Junto con los *stakeholders* su objetivo es negociar y definir los requisitos del producto (historias de usuario).
- **Scrum Master**: Se responsabiliza de introducir la metodología de Scrum en el equipo de desarrollo, asegurándose de que se cumplen sus pautas. Para ello trata de introducir valores sobre la metodología tanto teórica como prácticamente a los miembros del equipo, y ayudar a éstos a que desarrollen su capacidad de autogestión.
- **Equipo de desarrollo (Development Team)**: El equipo de desarrollo es el encargado de trabajar en el producto y entregar valor con cada *sprint*. El equipo de desarrollo ideal consta de no muchos miembros (4-10 personas).

Los siguientes roles no participan directamente en el proceso de Scrum: (1) clientes o interesados (*stakeholders*); y (2) usuarios finales.

3.1.2 Artefactos

Los artefactos de Scrum son los elementos que representan trabajo o valor. Están diseñados para maximizar la transparencia de la información clave. Cada artefacto garantiza que se proporciona información para mejorar la transparencia y el enfoque, con el que se puede medir el progreso.

- **Product Backlog**: La pila del producto, o *Product Backlog* es una lista con las historias de usuario sin realizar, a abordar en *sprints* posteriores al actual, para lograr el objetivo

del producto (*Product Goal*). Estas historias de usuario han sido previamente priorizadas y ordenadas por el *Product Owner* en negociación con los *stakeholders*.

  - **Objetivo del producto (*Product Goal*)**: Estado futuro del proyecto que se pretende alcanzar, y que debería ser el resultado del valor entregado de todas las historias de usuario, o la suma de todos los incrementos de todos los *sprints*. Es un objetivo a largo plazo a conseguir por el equipo de desarrollo
  - **Historia de Usuario**: Son los requisitos del cliente que forman parte del objetivo del producto y que es necesario trabajar y completar para otorgar valor a éste. Son redactadas y especificadas en el *Product Backlog* por el *Product Owner*, en un lenguaje sencillo desde la perspectiva del usuario.
  - **Épica**: Son historias de usuario de gran tamaño, que para ser más manejables deben subdividirse en historias más pequeñas, antes de abordarse en un *sprint*.

- **Sprint Backlog**: Es el subconjunto de tareas historias de usuario que se extrae del *Product Backlog* al principio de cada *sprint* para definir el objetivo de éste (*Sprint Goal*), en función de las capacidades del equipo de desarrollo y de la prioridad de los elementos que van a conformar el *Sprint Backlog*.

  - **Compromiso (*Sprint Goal*)**: El compromiso es un objetivo único del *sprint*, que los desarrolladores definen y se comprometen a alcanzar al final de éste durante el evento de *Sprint Planning*. Cada *sprint* tiene un único compromiso y los desarrolladores deben tenerlo en cuenta durante todo el progreso de éste.
  - **Tarea**: Son unidades de trabajo en los que se suele dividir una historia de usuario antes de entrar en el *Sprint Backlog*. Se utiliza para su documentación un lenguaje más técnico propio de los miembros del equipo de desarrollo.

- **Incremento**: Un Incremento es un paso firme hacia el Objetivo del Producto. Cada Incremento debe ser aditivo a todos los Incrementos anteriores y estar probado y verificado a fondo, asegurando que todos los Incrementos funcionan en consonancia. Para proporcionar el valor, el incremento debe ser utilizable.

  - **Definición de Hecho (*Definition of Done*)**: La Definición de Hecho es una descripción formal del estado del incremento cuando cumple con las medidas de calidad requeridas para el producto. En el momento en que un elemento de trabajo pendiente de producto cumple con la definición de hecho, se crea un incremento.

3.1.3 Eventos

Los eventos de Scrum son oportunidades formales para inspeccionar y adaptar los artefactos de la metodología. Estos eventos están diseñados específicamente para permitir la transparencia necesaria. Se utilizan para crear regularidad y minimizar la necesidad de reuniones no definidas en Scrum (véase figura 3.1).

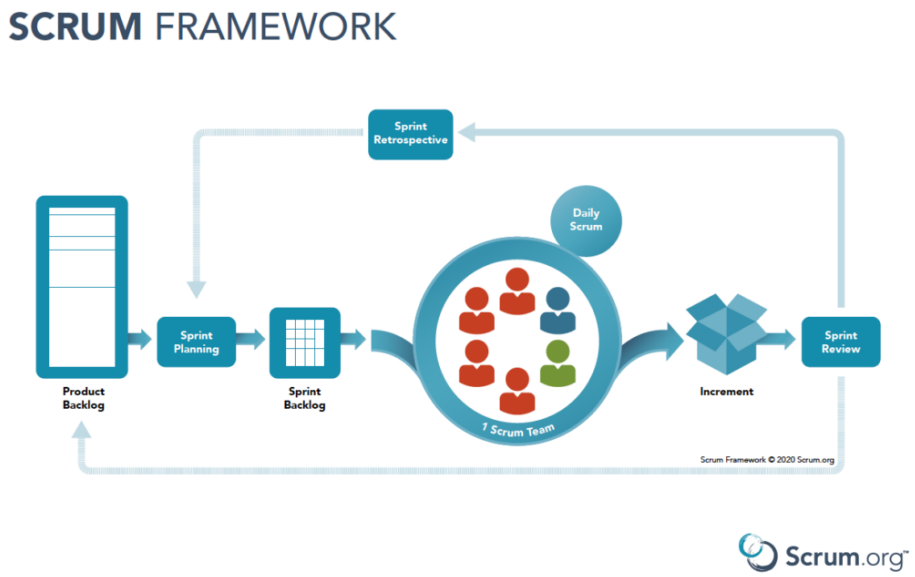


Figura 3.1: **Flujo de eventos de Scrum**
Fuente: **scrum.org** (*https://www.scrum.org/resources/blog/que-es-scrum*)

- ***Sprint***: Es el evento fundamental de Scrum. Todo el ciclo de desarrollo del producto se divide en estos eventos, que son de longitud fija (2-4 semanas) y consecutivos, de manera que la finalización de uno da lugar al comienzo del siguiente. Funcionan también como contenedor del resto de eventos.
- **Planificación del *sprint* (*Sprint Planning*)**: Es una reunión que tiene lugar al principio de un nuevo *sprint* para definir el objetivo de éste y, por lo tanto, el *Sprint Backlog*. Participan directamente tanto equipo de desarrollo como *Product Owner*.
- ***Daily Scrum***: Es un evento entre los miembros del equipo cuya principal meta es conocer el estado del objetivo del *sprint* actual. Los desarrolladores exponenen el trabajo realizado el día anterior de manera organizada. Es una reunión corta (10-15 minutos) y por comodidad se lleva a cabo en el mismo lugar todos los días laborables definidos para el *sprint*.
- **Revisión del *sprint* (*Sprint Review*)**: El objetivo de este evento es inspeccionar el resultado del *sprint* una vez terminado. El equipo presenta el resultado de su trabajo

a los interesados (*Product Owner*, *stakeholders*) y con ellos se discute el progreso hacia el objetivo del producto. La reunión suele tener un plazo máximo de 4 horas para un *sprint* de 1 mes.

- **Retrospectiva del *sprint* (*Sprint Retrospective*)**: El principal objetivo de este evento es buscar entre los miembros del equipo formas de aumentar la calidad y la eficacia. Se analiza básicamente que ha ido bien en el *sprint* y que problemas fueron encontrados. Suele tener un plazo máximo de 3 horas para un *sprint* de 1 mes.

3.2 Desarrollo Guiado por Pruebas (*Test Driven Development*)

El desarrollo guiado por pruebas (TDD) es una práctica de Ingeniería de Software para ayudar a crear un producto robusto y eficaz de manera ágil. Consiste en abordar el desarrollo de un requisito mediante la escritura de pruebas primero, e ir escribiendo el código de la solución de manera que esas pruebas funcionen [52]. Es una práctica propia de la metodología *eXtreme Programming* [53] y es muy utilizada en desarrollo ágil. TDD consiste básicamente en escribir nuestro código en pequeñas y seguras iteraciones, siguiendo el siguiente ciclo (véase figura 3.2):

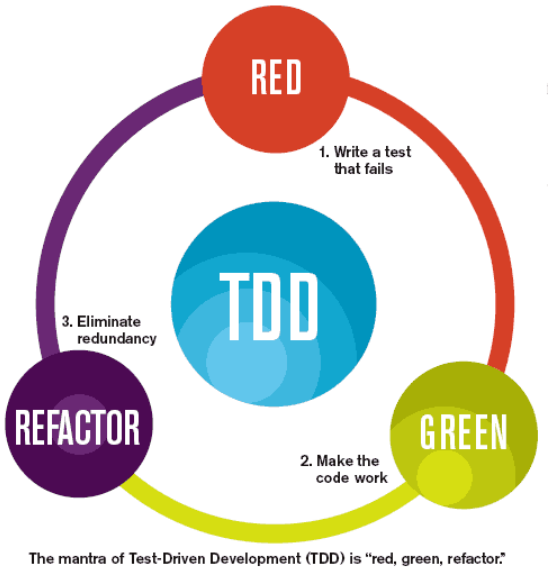


Figura 3.2: **Ciclo de *Test Driven Development***
Fuente: **spec-india.com** (*https://www.spec-india.com/blog/spec-india-implemented-test-driven-developmenttdd*)

- En primer lugar se representa el caso de prueba a abordar mediante un test pero sin que haya código que lo satisfaga. Como tal, el test deberá fallar lo que dará lugar a "luz roja".

- El siguiente paso es codificar lo mínimo posible para que ese test funcione y satisfacer el caso de prueba, de esta manera obtenemos "luz verde".
- Una vez que el test pase y tengamos "luz verde", se intentará refactorizar el código nuevo para evitar duplicidad y mejorarlo, pero intentando siempre que los test existentes no fallen y como tal manteniendo la "luz verde".

Las pruebas utilizadas son principalmente tests de unidad, las cuáles prueban los componentes del software de manera aislada unos de otros. Esto es importante ya que no podemos tener en cuenta las dependencias entre las unidades del sistema al escribirse las pruebas durante todo el ciclo de desarrollo, y el uso de componentes simulados (*mocks*) es fundamental.

La principales ventajas de utilizar TDD en nuestro proyecto podrían ser las siguientes:

- Aumenta la confianza en el código escrito porque ya se va probando a medida que avanzamos en su desarrollo.
- Mejora el diseño y la calidad, minimizando la cantidad de código escrito, ya que al ir realizando pruebas se evita escribir código que no se vaya a utilizar.
- Permite escribir software verificado de manera más ágil porque evitamos tener que realizar más pruebas al final al tener código ya probado. Ademas se optimiza el futuro mantenimiento de éste.

3.3 *Git-Flow*

Git-Flow es un método o flujo de trabajo con el SCM Git que permite un cómodo trabajo en equipo gracias a la división en ramas por funcionalidades [54], y favorece el desarrollo continuo y la integración con prácticas de *DevOps*. Hemos decidido emplear *Git-Flow* porque nos proporciona una nomenclatura de ramas y un método para trabajar con Git cómodo de seguir. Define una nomenclatura y unas funciones específicas para las distintas ramas del control de versiones (véase figura 3.3). Dichas ramas son las siguientes:

- *main* o *master*: Contiene el historial de publicación del proyecto, con commits con cada una de las versiones liberadas a producción.
- *develop*: Sirve de integración para las diferentes funcionalidades (*features*) y correcciones de errores que no estén en producción).
- Ramas de funcionalidad (*feature/*nombre_descriptivo): Parten de *develop* y se integran en la misma cuando el desarrollo sobre estas ramas está finalizado. En estas ramas se aborda desarrollo evolutivo (nuevas funcionalidades) y corrección de errores no urgentes.

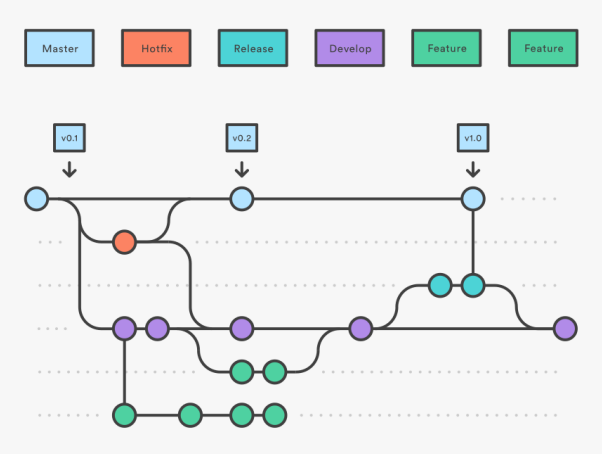


Figura 3.3: **Diagrama de *Git-Flow***
Fuente: **atlassian.com** (*https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow*)

- Rama de liberación (*release/*nombre_version): Esta rama se crea a partir de *develop* cuándo ésta tiene las suficientes funcionalidades para dar lugar a una versión nueva o se acerca una fecha de entrega programada.
- Rama de correcciones (*hotfix/*nombre_descriptivo): Esta rama se crea a partir de *main* para abordar la corrección de una incidencia que tiene lugar en producción.

Las ramas *main* y *develop* son ramas protegidas, esto quiere decir que ningún usuario puede subir directamente cambios a las mismas, únicamente permiten la integración de otras ramas a partir de *pull-requests*, previa revisión y aprobación por otros miembros del equipo. Es importante que antes de una revisión y aprobación de estas *pull-requests* el contenido de la rama a integrar sea correctamente verificado.

3.4 Adaptación

La metodología utilizada para este proyecto ha sido una conjunción de las tres descritas anteriormente, adaptadas a un equipo de trabajo de una sola persona.

Con respecto a **Scrum** (3.1), estas son las modificaciones que hemos tenido que realizar:

- Debido a que se trata de un proyecto llevado a cabo por una sola persona, el peso de los roles que participan directamente en Scrum recae sobre el autor del proyecto, asumiendo principalmente el rol del equipo de desarrollo. El director del proyecto asumirá también roles de cliente o parte interesada (*stakeholders*) en los eventos de *Sprint Review* y ofrecerá soporte sobre los requisitos que se han ido definiendo para transformar en historias de usuario, así como el rol de *Scrum Manager*. Las historias de usuario han sido redactadas por el autor del proyecto en consenso con el *Scrum Manager* (director).

- Los *sprints* según Scrum deben ser de duración constante y tener una carga de trabajo equitativa entre ellos; además de que deben ser sucesivos, esto es, la conclusión de uno debe dar lugar a la apertura del siguiente. En el caso de este proyecto, no hemos seguido un ritmo de trabajo constante y los *sprints* no siempre han empezado según terminaba el anterior, habiendo periodos de tiempo "muerto" entre ellos. Sí hemos procurado balancear la carga de trabajo, abordando más o menos dos historias de usuario con una carga de trabajo algo considerable en cada uno, y todos han durado entre 2 y 3 semanas.
- Los eventos:
  - ***Sprint Planning***, ***Sprint Review*** y ***Sprint Retrospective*** se han simulado mediante reuniones de seguimiento entre el autor y el director del proyecto. Han sido reuniones de alrededor de una hora de duración en las que el autor principalmente toma el rol de equipo de desarrollo, y el director de cliente o *stakeholder*. Se le presenta el trabajo realizado en el *sprint* anterior y problemas encontrados, y entre los dos acuerdan el objetivo del siguiente *sprint*. Posteriormente a la reunión hemos estimado las historias de usuario a formar parte del nuevo *Sprint Backlog* y subdividido en tareas.
  - ***Daily Scrum***: Autoanálisis por parte del autor del proyecto del trabajo realizado el día anterior. Se ha tenido que reestimar alguna historia de usuario en base a este evento.

Respecto a **TDD** (3.2), hemos intentado seguir la práctica lo más fiel posible siguiendo el flujo (figura 3.2) para la parte del lado servidor. Para ello hemos definido casos de prueba para cada una de las nuevas tareas a abordar y hemos creado un test de unidad para cada uno, haciendo primero que fallaran y luego escribiendo el código para que funcionasen, con su posterior refactorización. El uso de *Mockito* (2.1.1) ha sido fundamental para esta parte, pues al no tener todos los módulos codificados teníamos que probarlos de manera aislada simulándo las dependencias con *mocks*. Para la parte de cliente hemos realizado pruebas de unidad pero no hemos aplicado TDD, debido a que son más complejas porque son sobre el árbol DOM de HTML, y por consiguiente eran más difíciles de escribir antes del código a probar.

Respecto a ***Git-Flow*** (3.3), hemos adoptado su nomenclatura y reglas para las distintas ramas. Hemos creado ramas *feature* a partir de *develop* para cada una de las historias de usuario. También hemos simulado trabajo en equipo al abordar algunas historias de usuario en distintas ramas de forma paralela, realizando *pull-requests* y resoluciones de conflictos. En total hemos liberado una versión después del *sprint* final con el producto acabado.

Capítulo 4

Análisis

En este capítulo se describen los requisitos y necesidades que dan lugar al desarrollo del proyecto, así como su ámbito y modelo conceptual. El análisis de requisitos se lleva a cabo en los instantes iniciales de un proyecto y es una de las partes fundamentales del desarrollo de software. En esa fase se determinarán las necesidades o condiciones clave que debe cumplir nuestra aplicación. Antes de iniciar la fase de diseño es muy importante que se analice con profundidad el alcance del sistema y sus funcionalidades.

4.1 Ámbito de la aplicación y descripción general

El sistema desarrollado con el proyecto permitirá al usuario investigador subir un archivo ZIP representando una colección de referencia. Dicha colección de referencia estará compuesta por archivos XML representando cada uno la estructura de un hilo de Reddit. En al figura 4.1 puede verse la estructura que deberán seguir estos archivos XML. La colección deberá llevar un nombre descriptivo introducido en el formulario, que ayude a identificarla después.

Cuando la colección es subida, el sistema analizará sintácticamente (*parsing*) los archivos XML para obtener una estructura del Hilo con cada uno de los posts y junto a los autores que los han escrito. Luego se procesará cada uno de los contenidos de los posts con algoritmos de NLP para obtener las entidades nombradas. El sistema también obtendrá las coordenadas de las entidades de tipo "localización". También se reconocerán las entidades en relación a un conjunto de términos especificado (tesauro) relevantes sobre la temática que estamos tratando, permitiendo asociar cada una de ellas con estos términos si aparecen en conjunción con ellos, pudiendo ser visualizados más adelante. Toda esta información se persistirá en las bases de datos.

Una colección subida podrá ser analizada tanto por el usuario investigador que la subió como por los investigadores y colaboradores a los que se la haya compartido. Estos usuarios podrán ver la lista de colecciones que tienen disponibles y junto a cada una tendrán un enlace

```
            <thread>
                <id>6234</id>
                <relevant-posts>
                    <id>6236</id>
                    <id>279976</id>
                </relevant-posts>
                <posts>
                    <post>
                        <id>279976</id>
                        <author>279969</author>
                        <body>
                        Changing building and/or institutional names and
    removing statues is a positive but empty gesture if it isn´t accompanied
    by impactful, functional, sustainable change.
                        </body>
                    </post>
                    <post>
                        <id>279977</id>
                        <author>279970</author>
                        <body>
                        This times a million! Don´t ever change it.
                        </body>
                    </post>
                </posts>
            </thread>

```

Figura 4.1: Estructura de un XML reconocido por la aplicación

por cada una de las vistas para realizar el análisis: Vista de **Mapa de Entidades** y vista de ***Dashboard***. La vista del mapa permitirá contemplar en un mapa todas las entidades de tipo localización obtenidas junto a información de cada una como las referencias a posts y hilos en los que son nombradas. La vista de *dashboard* posibilitará visualizar estadísticas sobre la colección como: Número de hilos, autores, posts y entidades obtenidas, así como un gráfico con la distribución del número de autores, posts y entidades por hilo, entre otros.

4.2 Requisitos

La tarea principal del análisis de este proyecto es la recogida de requisitos, parte fundamental pues será la forma de comprender la aplicación que se intenta construir, poniendo en contexto tanto a desarrolladores como a clientes y/o usuarios.

4.2.1 Actores

Para la recogida de los diferentes requisitos hemos identificado los siguientes actores que podrían hacer uso del sistema (puede verse la jerarquía de los mismos en la figura 4.2):

- **Usuario no identificado**: Usuario que accede al sistema pero no ha iniciado sesión.

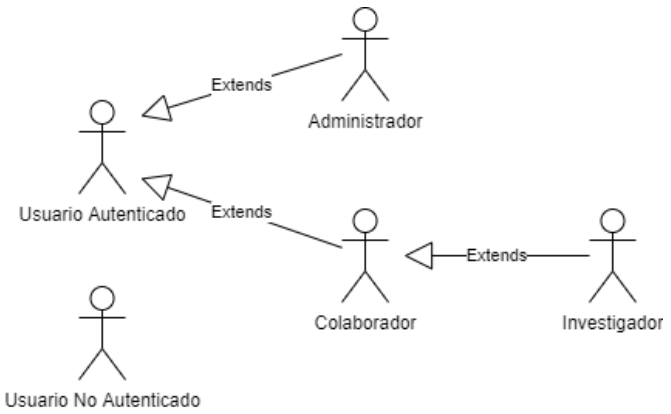


Figura 4.2: Jerarquía de actores del sistema

- **Usuario identificado**: Usuario que ha inciado sesión en el sistema. Hay varios roles para éste:
  - **Administrador**: Usuario encargado de la gestión de otros usuarios investigadores o colaboradores.
  - **Investigador**: Usuario que técnicamente posee las colecciones y tiene permiso para subirlas y compartirlas, así como también analizarlas.
  - **Colaborador**: Usuario que sólo tiene permisos para analizar colecciones que le hayan compartido.

4.2.2 Historias de Usuario

Tras un estudio de las necesidades que debería cubrir el futuro sistema, se llevó a cabo la definición de las funcionalidades, la cuál hemos estado realizando no sólo al inicio del proyecto si no a medida que avanzaban los diferentes *sprints* y hemos ido elaborando el *Product Backlog*.

Se han agrupado los diferentes requisitos en épicas e historias de usuario. Junto con cada historia de usuario se han definido una serie de **Criterios de Aceptación** los cuáles deben verificarse con pruebas de aceptación una vez finalizada para comprobar que la historia de usuario ha sido cubierta.

- **Épica 1 — Identificación y Acceso**

  En esta épica hemos introducido historias de usuario relacionadas con usuarios no identificados en el sistema, así como algunas para usuarios identificados independientemente de su rol, que deseen modificar su cuenta o cerrar sesión.

  - **Historia Usuario 1.1** — *Como* **usuario no identificado** *quiero* iniciar sesión en el sistema *para* tener acceso a las demás funcionalidades.

  - *Dada* una petición para iniciar sesión, *entonces* se abre un formulario para cubrir con "Nombre de usuario", "Contraseña" y "Recuérdame".
  - *Dado* el formulario, *si* dejo un campo sin cubrir, *entonces* el sistema me avisa de que el campo es necesario.
  - *Dado* el formulario, *si* introduzco un usuario que no existe o un usuario con contraseña incorrecta, *entonces* el sistema me avisa de que las credenciales son inválidas.
  - *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
  - *Dado* el formulario, *en cualquier otro caso* me identifico en el sistema y me redirige a la página de inicio.

- **Historia Usuario 1.2** — *Como* **usuario no identificado** *quiero* registrarme en el sistema como colaborador *para* poder iniciar sesión.
  - *Dada* una petición para registrarse, *entonces* se abre un formulario para cubrir con "Nombre", "Apellidos", "Nombre de usuario", "E-Mail", "Contraseña" y "Confirmar Contraseña".
  - *Dado* el formulario, *si* dejo un campo sin cubrir, *entonces* el sistema me avisa de que el campo es necesario.
  - *Dado* el formulario, *si* introduzco un E-Mail con formato incorrecto, *entonces* el sistema me avisa de que es incorrecto.
  - *Dado* el formulario, *si* introduzco un Nombre de Usuario o un E-Mail que ya existe, *entonces* el sistema me avisa de que ya está en uso.
  - *Dado* el formulario, *si* introduzco una contraseña de menos de 8 caracteres, *entonces* el sistema avisa de que tiene que tener más de 8 caracteres.
  - *Dado* el formulario, *si* la contraseña y el campo "Confirmar Contraseña" no coinciden, *entonces* el sistema avisa que el campo debe coincidir.
  - *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
  - *Dado* el formulario, *en cualquier otro caso* se crea una cuenta en el sistema con los datos especificados y rol colaborador, me identifico y me redirige a la página de inicio.

- **Historia Usuario 1.3** — *Como* **usuario identificado** *quiero* cerrar sesión del sistema *para* dejar de tener acceso temporalmente al mismo.
  - *Dada* una petición para cerrar sesión, *entonces* el sistema borra toda la información en la web del usuario y redige a la página de inicio de usuario no identificado.

- **Historia Usuario 1.4** — *Como* **usuario identificado no administrador** *quiero* darme de baja del sistema *para* dejar de tener acceso permanentemente.
  - *Dada* una petición para darse de baja, *entonces* el sistema muestra un mensaje con las opciones de "cancelar" o "confirmar la baja".
  - *Dada* la cancelación, *entonces* el mensaje se cierra y se continúa como hasta ahora.
  - *Dada* la confirmación de la baja, *entonces* el sistema borra la sesión del usuario y la información persistida de su cuenta, impidiendo que se pueda volver a iniciar sesión con el usuario.
- **Historia Usuario 1.5** — *Como* **usuario identificado** *quiero* cambiar mi contraseña *para* poder disfrutar de un extra de seguridad.
  - *Dada* una petición de cambio de contraseña, *entonces* se abre un formulario para cubrir con "Contraseña Antigua", "Contraseña Nueva" y "Confirmar Contraseña Nueva".
  - *Dado* el formulario, *si* dejo un campo sin cubrir, *entonces* el sistema me avisa de que el campo es necesario.
  - *Dado* el formulario, *si* introduzco una contraseña nueva de menos de 8 caracteres, *entonces* el sistema avisa de que tiene que tener más de 8 caracteres.
  - *Dado* el formulario, *si* la contraseña antigua y la nueva coinciden, *entonces* el sistema me avisa de que no deben coincidir.
  - *Dado* el formulario, *si* la contraseña nueva y el campo "Confirmar Contraseña Nueva" no coinciden, *entonces* el sistema avisa que el campo debe coincidir.
  - *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
  - *Dado* el formulario, *en cualquier otro caso* la contraseña se cambia correctamente. La próxima vez que inicie sesión podré hacerlo con la nueva contraseña.
- **Historia Usuario 1.6** — *Como* **usuario recién registrado** *quiero* verificar mi dirección de correo electrónico enviando un E-Mail de verificación a ésta, *para* así poder utilizar una dirección de correo electrónico existente.
  - *Dado* un registro correcto de un nuevo usuario, *entonces* se redirige a éste a una nueva página que dice que tiene que confirmar su dirección de correo electrónico. La página le da la opción de volver a enviar el E-Mail de verificación o cerrar sesión. En la bandeja de entrada de mi E-Mail debería de aparecer el E-Mail de verificación.

  - *Dada* la página anterior, *si* hago click en "cerrar sesión", *entonces* ésta se cierra.
  - *Dada* la página anterior, *si* hago click en "volver a enviar mail", *entonces* el sistema me vuelve a enviar el E-Mail de verificación.
  - *Dado* el emial de verificación, *si* hago click en el enlace, *entonces* el sistema habilita a mi usuario y me abre el navegador con la sesión iniciada.

- **Épica 2 — Gestión de Usuarios**

  En esta épica hemos introducido historias de usuario relacionadas con la gestión de usuarios investigadores o colaboradores desde otra cuenta, que sólo podrá llevar a cabo un administrador.

  - **Historia Usuario 2.1** — *Como* **administrador** *quiero* registrar en el sistema a un nuevo usuario como investigador o colaborador *para* que éste pueda iniciar sesión y acceder al resto de funcionalidades del rol (categoría) especificado.
    - *Dada* una petición para registrar un usuario, *entonces* se abre un formulario para cubrir con "Nombre", "Apellidos", "Nombre de usuario", "Contraseña", "Confirmar Contraseña" y "Categoría" (rol).
    - *Dado* el formulario, *si* dejo un campo sin cubrir, *entonces* el sistema me avisa de que el campo es necesario.
    - *Dado* el formulario, *si* introduzco un Nombre de Usuario que ya existe, *entonces* el sistema me avisa de que ya está en uso.
    - *Dado* el formulario, *si* introduzco una contraseña de menos de 8 caracteres, *entonces* el sistema avisa de que tiene que tener más de 8 caracteres.
    - *Dado* el formulario, *si* la contraseña y el campo "Confirmar Contraseña" no coinciden, *entonces* el sistema avisa que el campo debe coincidir.
    - *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
    - *Dado* el formulario, *en cualquier otro caso* se crea una cuenta en el sistema con los datos especificados y el rol especificado en el combo de selección.
  - **Historia Usuario 2.2** — *Como* **administrador** *quiero* ver un listado con los usuarios investigadores y colaboradores *para* poder cambiarlos de categoría (rol) o darlos de baja.
    - *Dada* una petición para listar los usuarios, *entonces* obtengo un listado por páginas de usuarios colaboradores o investigadores, con la posibilidad de ordenarlos por "Nombre", "Apellidos" o "Nombre de Usuario"; ascendente o descendentemente.

  - **Historia Usuario 2.3** — *Como* **administrador** *quiero* dar de baja a un usuario investigador o colaborador *para* que éste deje de tener acceso al sistema.
    * *Dada* una petición para eliminar un usuario investigador o colaborador, *entonces* el sistema borra la sesión del usuario y la información persistida de su cuenta, impidiendo que se pueda volver a iniciar sesión con el usuario.
  - **Historia Usuario 2.4** — *Como* **administrador** *quiero* cambiar el rol de un usuario colaborador a investigador o viceversa *para* que éste pase a tener permisos específicos del nuevo rol.
    * *Dada* una petición de cambio de rol, *entonces* el sistema actualiza el rol del usuario y pasa a tener las funciones específicas.

- **Épica 3 — Manejo de Colecciones**

  En esta épica hemos introducido historias de usuario relacionadas con la subida de las colecciones al sistema y su posterior gestión como compartirlas a otros usuarios. Son todas funcionalidades disponibles sólo para un usuario investigador.

  - **Historia Usuario 3.1** — *Como* **investigador** *quiero* subir un archivo de colección de referencia *para* poder analizarla posteriormente.
    * *Dada* una petición para subir una colección, *entonces* se abre un formulario con los campos "Título" y "Archivo" (para introducir el archivo ZIP).
    * *Dado* el formulario, *si* dejo un campo sin cubrir, *entonces* el sistema me avisa de que el campo es necesario.
    * *Dado* el formulario, *si* introduzco un título para la colección muy largo (más de 30 caracteres), *entonces* el sistema me avisa de que utilice un título más corto.
    * *Dado* el formulario, *si* especifico un archivo que no está en formato ZIP, *entonces* el sistema me avisa de que el formato del archivo no es válido.
    * *Dado* el formulario, *si* especifico un archivo demasiado grande (más de 25 MB), *entonces* el sistema avisa de que el archivo debe ser más pequeño.
    * *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
    * *Dado* el formulario, *en cualquier otro caso* se crean metadatos para la colección en base de datos en base al título especificado, y se marca en estos metadatos la colección como "Cargando". Si hay un límite de colecciones a subir que permite el sistema y se excede, se marcará la colección como "En cola" y estará esperando a que haya un hueco disponible. Si lo hay, se lanza un proceso

en segundo plano asíncrono que intenta parsear la colección y obtener las entidades y la información de todos los autores, hilos y posts de los XML y persistirla en base de datos. Una vez finalizado el proceso, se avisará al usuario con un mensaje y se intentará parsear otra colección si estaba en cola.
    * *Dada* una colección subida, *si* se ha subido sin ningún error grave, *entonces* el sistema la marca como "Subida Correctamente".
    * *Dada* una colección subida, *si* se ha producido un error grave, *entonces* el sistema la marca como "Error al cargar" y no se permitirá el análisis sobre esta colección.
  - **Historia Usuario 3.2** — *Como* **investigador** *quiero* compartir o dejar de compartir una colección de mi propiedad con otros usuarios investigadores o colaboradores *para* que éstos puedan analizar la colección posteriormente o dejar de analizarla.
    * *Dada* una petición para ver las colecciones, *entonces* aparece una lista paginada con mis colecciones propias y la opción de "Ajustes" al lado de cada una.
    * *Dado* un click en la opción de ajustes de una colección, *entonces* el sistema me redirige a la correspondiente página para la colección especificada. Esta página tiene un formulario para introducir el nombre de un usuario al que compartirle la colección y una lista con los usuarios compartidos y la opción al lado de cada uno para dejarle de compartir.
    * *Dado* el formulario, *si* introduzco el nombre de un usuario que no existe o para el que ya está compartida la colección, entonces el sistema me muestra un error.
    * *Dado* el formulario, *si* se produce cualquier otro error, *entonces* el sistema me avisa de Error Interno.
    * *Dado* el formulario, *si* especifico un nombre de usuario correcto, *entonces* el sistema asigna la colección al usuario. Podrá consultarla a partir de ahora.
    * *Dado* un click en la opción de dejar de compartir de un usuario de la lista, *entonces* el sistema quita la colección de las colecciones disponibles de ese usuario. Ya no la podrá consultar.

- **Épica 4 — Análisis de Colecciones**

  En esta épica hemos introducido historias de usuario relacionadas con el análisis de colecciones: consulta del mapa de entidades y del *dashboard*. Son funcionalidades accesibles tanto para investigadores como colaboradores (siempre que se les hay compartido la colección a analizar).

- **Historia Usuario 4.1** — *Como* **investigador** o **colaborador** *quiero* ver las entidades de tipo localización de una colección en un mapa *para* así poder realizar el análisis del mismo.
  - *Dada* la lista de colecciones propias o compartidas conmigo, *entonces* aparece un enlace al lado de cada una que me permite consultar el mapa de entidades para dicha colección.
  - *Dada* una petición para ver el mapa de entidades de una colección, *entonces* aparece un mapa con cada una de las mencionadas entidades en la ubicación en la que están con una chincheta.
  - *Dado* un click en una de las chinchetas, *entonces* el sistema muestra un *pop-up* con información sobre la entidad señalada con la chincheta: Nombre, Descripción y Referencias (Hilo, Autor y Post) donde es mencionada.
- **Historia Usuario 4.2** — *Como* **investigador** o **colaborador** *quiero* ver un *dashboard* con determinadas estadísticas de una colección *para* así poder realizar el análisis del mismo.
  - *Dada* la lista de colecciones propias o compartidas conmigo, *entonces* aparece un enlace al lado de cada una que me permite consultar el *dashboard* para dicha colección.
  - *Dada* una petición para ver el *dashboard* de una colección, *entonces* aparece el *dashboard* con diferentes estadísticas sobre la colección: Número de Hilos, Autores, Posts y Entidades extraídas. También dispongo de un gráfico de burbujas que permite consultar la distribución del número de autores, posts, entidades y posts relevantes por cada hilo; así como un gráfico de sectores para consultar la proporción de cada tipo de entidad (Localización, Organización o Persona) sobre el totoal de entidades.
  - *Dado* el gráfico de burbujas, *si* paso el mouse sobre una de ellas, *entonces* el sistema muestra un *pop-up* con información del hilo que representa la burbuja (Número de Autores, Número de Posts, Número de Entidades y Número de Posts Relevantes).
- **Historia Usuario 4.3** — *Como* **investigador** o **colaborador** *quiero* ver que entidades extraídas hacen referencia a los términos de un tesauro especificado, *para* así poder saber si esas entidades son relevantes.
  - *Dado* el mapa de entidades de una colección, *entonces* en el *pop-up* de cada entidad, junto a cada referencia, puedo ver si esa referencia menciona algún término del tesauro y cuáles.

* *Dado* el *dashboard* de una colección, *entonces* aparece un nuevo gráfico de burbujas en el que puedo ver el número de ocurrencias de cada término del tesauro en toda la colección.

4.3 Modelo de Datos

En la figura 4.3 se muestra un modelo de datos conceptual (como diagrama de Entidad-Relación).

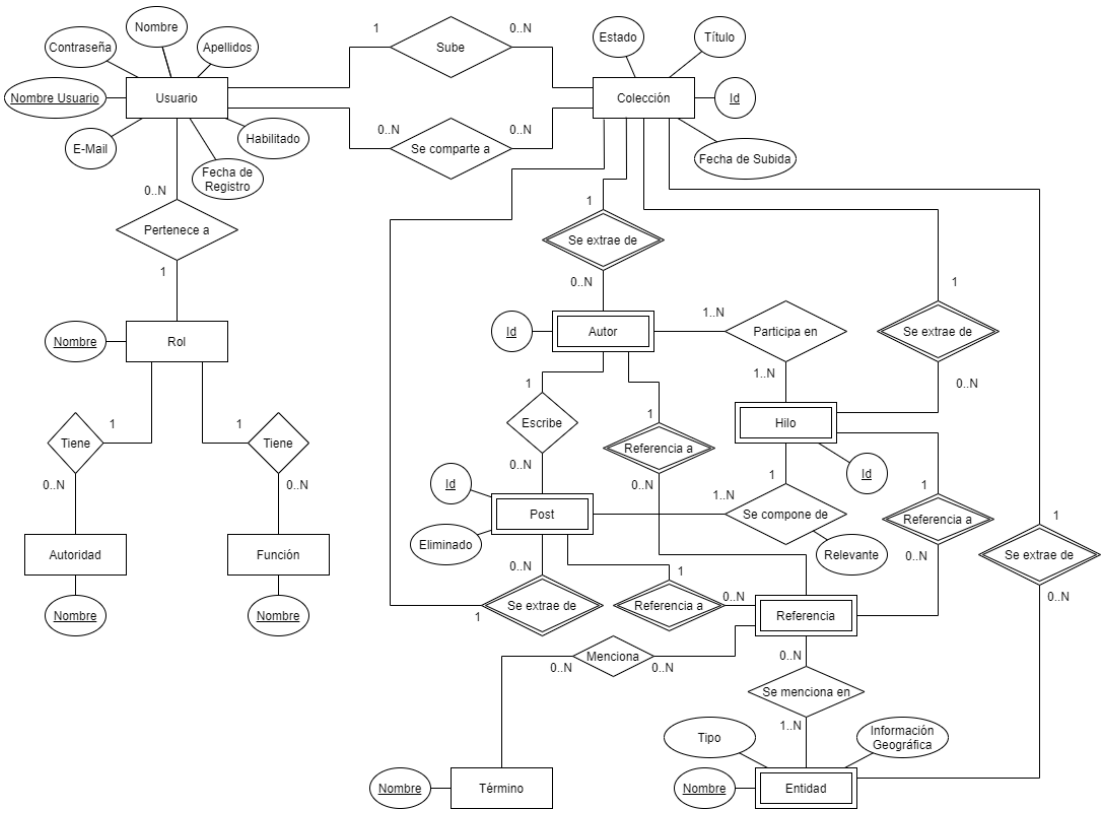


Figura 4.3: Modelo de Datos conceptual (Entidad-Relación)

A continuación describimos cada una de las entidades con sus atributos y las relaciones:

- **Usuario**: Representa a un usuario que inicia sesión e interactúa con el sistema.
  - **Nombre Usuario**: Nombre de Usuario único con el que el usuario se registra e identifica en el sistema.
  - **Contraseña**: Contraseña que el usuario utiliza para verificar su identidad al identificarse.
  - **Nombre**: Nombre de pila del usuario.

  - **Apellidos**: Apellidos del usuario.
  - **E-Mail**: E-Mail único y personal que el usuario especifica para registrarse.
  - **Fecha de Registro**: Fecha en la que se dió de alta al usuario.
  - **Habilitado**: Indica si el usuario está habilitado o no (los usuarios nuevos que se registran es necesario que verifiquen su correo electrónico).

- **Rol**: Roles a los que pueden pertenecer los usuarios y definen los permisos que éstos tienen.
  - **Nombre**: Nombre único del rol.

- Usuario **pertenece a** Rol: Todo usuario debe pertenecer siempre a un rol.

- **Autoridad**: Cada uno de los permisos que están definidos para cada una de las distintas funcionalidades del sistema.
  - **Nombre**: Nombre único y descriptivo de la autoridad.

- Rol **tiene** Autoridades: Un rol tiene varias autoridades que definen los permisos que puede tener el usuario sobre las funcionalidades del sistema.

- **Función**: Es una característica que puede tener un rol para determinadas funcionalidades. Por ejemplo la función, "Usuario Protegido" especifica que no se pueden borrar usuarios con un rol con esta función definida.
  - **Nombre**: Nombre único y descriptivo de la función.

- Rol **tiene** Funciones: Un rol puede tener varias funciones para hacer que algunas funcionalidades se comporten de manera especial con algunos usuarios.

- **Colección**: Representa los metadatos de una colección subida por un usuario investigador.
  - **Id**: Identificador único para la colección.
  - **Título**: Título descriptivo que el usuario especifica con la colección al subirla.
  - **Fecha de Subida**: Fecha (con hora incluida) en la que se subió la colección.
  - **Estado**: Estado de la colección con respecto a su subida: "En Cola", "Cargando", "Cargada Correctamente" o "Fallo al cargar".

- Usuario **sube** Colección: Un usuario (siempre que sea investigador) puede subir varias colecciones. Una colección siempre tiene que ser subida por un usuario.

- Colección **se comparte a** Usuario: Una colección puede ser compartida a varios usuarios. Un usuario puede tener varias colecciones compartidas con él.
- **Autor**: Cada uno de los autores que escriben posts en los hilos y que se extraen de la colección.
  - **Id**: Identificador único del autor pero sólo para la colección.
- Autor **se extrae de** Colección: Los autores se extraen siempre de una colección y están vinculados a la misma. Una colección puede extraer de 1 a muchos autores.
- **Hilo**: Cada uno de los hilos de posts extraídos de la colección.
  - **Id**: Identificador único del hilo pero sólo para la colección.
- Hilo **se extrae de** Colección: Los hilos se extraen siempre de una colección y están vinculados a la misma. Una colección puede extraer de 1 a muchos hilos.
- Autor **participa en** Hilo: Un autor puede escribir posts en 1 a muchos hilos. Un hilo puede tener de 1 a muchos autores escribiendo posts en él.
- **Post**: Cada uno de los posts de los hilos extraídos de la colección.
  - **Id**: Identificador único del post pero sólo para la colección.
  - **Eliminado**: Indica si el post ha sido eliminado en la red social.
- Post **se extrae de** Colección: Los posts se extraen siempre de una colección y están vinculados a la misma. Una colección puede extraer de 1 a muchos posts.
- Autor **escribe** Post: Un autor puede escribir de 1 a muchos posts en los hilos de la colección. Un post tiene que estar siempre escrito por un autor.
- Hilo **se compone de** Post: Un hilo tiene que tener al menos un post y puede tener hasta muchos. Un post tiene que estar siempre escrito en un hilo y pertenecer a él.
  - **Relevante**: Indica si el post es relevante para el hilo en el que está escrito.
- **Entidad**: Cada una de las entidades nombradas que se extraen de la colección.
  - **Nombre**: Nombre único de la entidad pero sólo para la colección.
  - **Tipo**: Tipo de la entidad: “Localización”, “Organización” o “Persona”.

  - **Información Geográfica**: Atributo compuesto que se especifica sólo si la entidad es de tipo "Localización". Representa las coordenadas geográficas y otra información necesaria para mostrarse en el mapa de entidades.

- Entidad **se extrae de** Colección: Las entidades se extraen siempre de una colección y están vinculados a la misma. Una colección puede extraer de 1 a muchas entidades.

- **Referencia**: Cada una de las referencias de una entidad por un autor, en un hilo y en un post.

- Referencia **referencia a** Autor: Una referencia tiene que especificar un autor. Un autor puede ser especificado en ninguna o en muchas referencias.

- Referencia **referencia a** Hilo: Una referencia tiene que especificar un hilo. Un hilo puede ser especificado en ninguna o en muchas referencias.

- Referencia **referencia a** Post: Una referencia tiene que especificar un post. Un post puede ser especificado en ninguna o en muchas referencias.

- Entidad **se menciona en** Referencia: Una entidad tiene que ser mencionada en al menos una referencia. Una referencia puede aplicarse a ninguna o a muchas entidades.

- **Término**: Cada uno de los términos del tesauro que aplicamos para buscar entidades relevantes.

  - **Nombre**: Nombre único del término.

- Referencia **menciona** Término: Un término puede ser mencionado de ninguna a muchas referencias. Una referencia puede mencionar de 0 a muchos términos del tesauro.

El modelo anterior dispone de una relativa gran cantidad de relaciones entre las distintas entidades. Algunas de ellas se pueden implementar como listas dentro de las entidades de base de datos (e.g. cada rol almacena una lista con sus autoridades y sus funciones, o cada entidad extraída de la colección almacena una lista de sus referencias). Para simplificar la persistencia y evitar tener que hacer demasiadas uniones (*joins*) en las consultas finales para recuperar los datos; hemos decidido utilizar, junto a una base de datos Relacional (MySQL) para entidades simples que no tengan de estos atributos tipo lista, una base de datos No Relacional para las entidades que si necesitan almacenar listas de datos como atributos (véase sección 2.1.3). MongoDB es perfecto para esto al estar orientada a documentos.

Por consiguiente, hemos transformado el modelo conceptual en un modelo técnico más ajustado a las entidades de BD y documentos que maneja nuestra aplicación; y transformado

algunas de las entidades en tablas para la BD relacional y otras entidades en colecciones de documentos para la BD no relacional.

Así, en la figura 4.4 puede verse un modelo de datos técnico, con las tablas y relaciones reales usadas en la base de datos relacional; y con las colecciones de documentos usadas en la no relacional, junto con las relaciones entre ambas bases de datos. Todas las relaciones en el modelo son de 1 a muchos. Puede observarse, como algunas entidades como Rol se han convertido en colecciones de una base de datos No Relacional (en verde) lo cuál nos permite tener atributos tipo lista (en este caso *autoridades* y *funciones*).

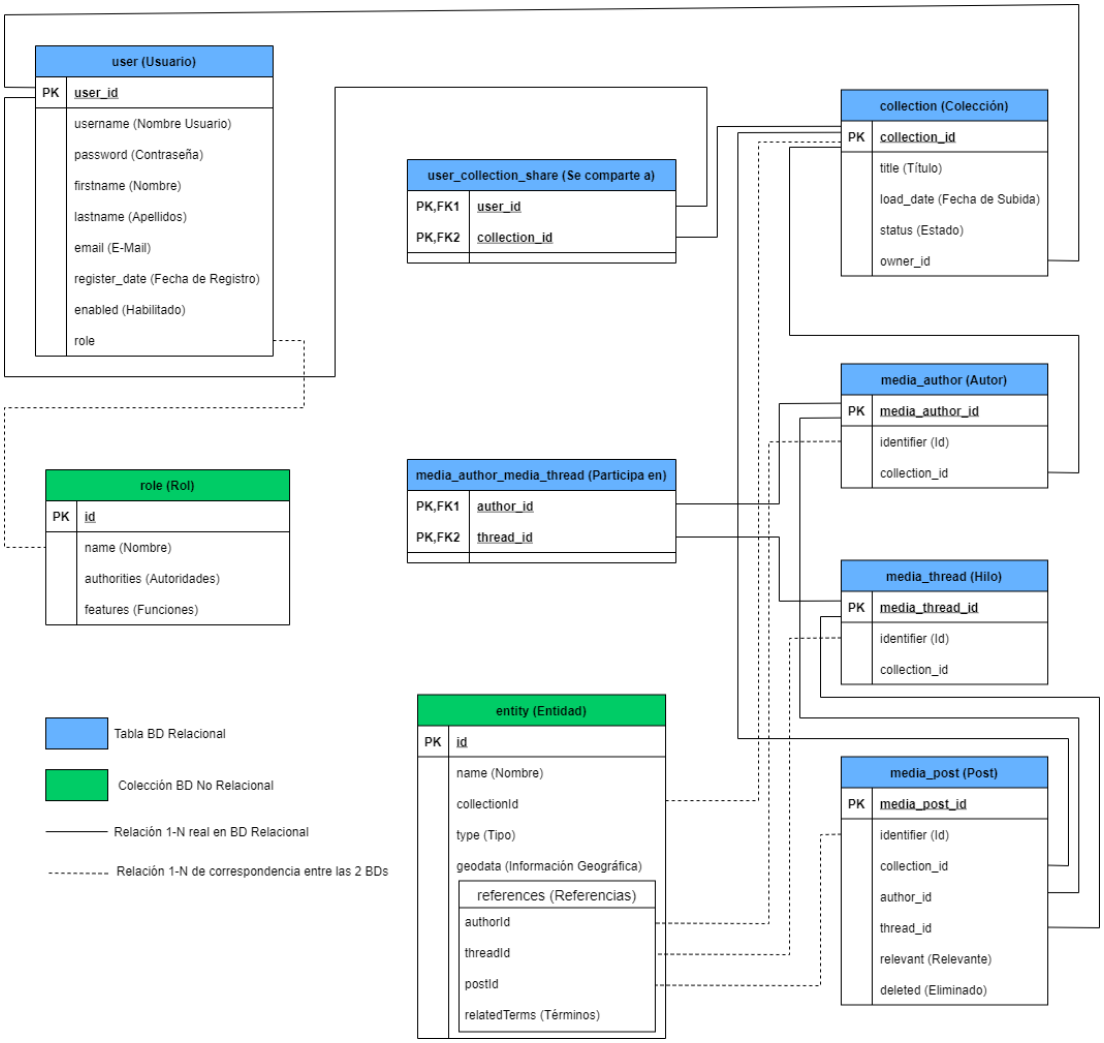


Figura 4.4: Modelo de Datos Técnico (Tablas de Base de Datos Relacional y Colecciones de Base de Datos No Relacional)

Capítulo 5

Diseño

En este capítulo se comentan las decisiones que se han tomado en cuanto al diseño de la aplicación y su arquitectura.

5.1 Diseño de la Interfaz

Para el diseño de la interfaz de la aplicación hemos ido creando diversas maquetas o bocetos (*mockups*) a medida que avanzábamos en el desarrollo del proyecto, utilizando la herramienta **Pencil** (véase sección 2.1.4). El objetivo es crear una interfaz para el usuario simple e intuitiva, para que éste pueda acceder a las distintas funcionalidades de manera sencilla y cómoda. Para poder conseguir esto se han intentado seguir los principios de User Interface (UI) y User Experience (UX) [55].

La solución de diseño planteada es tener una página principal a la que el usuario accede cuando no está identificado, que le de acceso de manera cómoda a los formularios de inicio de sesión y de registro. Una vez iniciada la sesión, el usuario dispone de una serie de elementos comunes de navegación, para todas las páginas y vistas:

- **Barra de navegación lateral (*sidebar*)**: Permite acceder a las distintas páginas a las que el usuario identificado tiene acceso.
- **Barra superior (*topbar*)**: Permite la navegación hacia atrás si está disponible (*breadcrumbs*) y dispone de un enlace para cerrar sesión y otro para acceder a la configuración de la cuenta.

Para el usuario administrador se ha optado por una vista de tabla sencilla con paginación para poder **visualizar la información de usuarios investigadores y colaboradores** de una manera cómoda, así como con botones al lado de cada uno para poder actualizar su rol o borrarlos de manera sencilla, y una opción al fondo de la página para añadir más usuarios (véase figura 5.1).

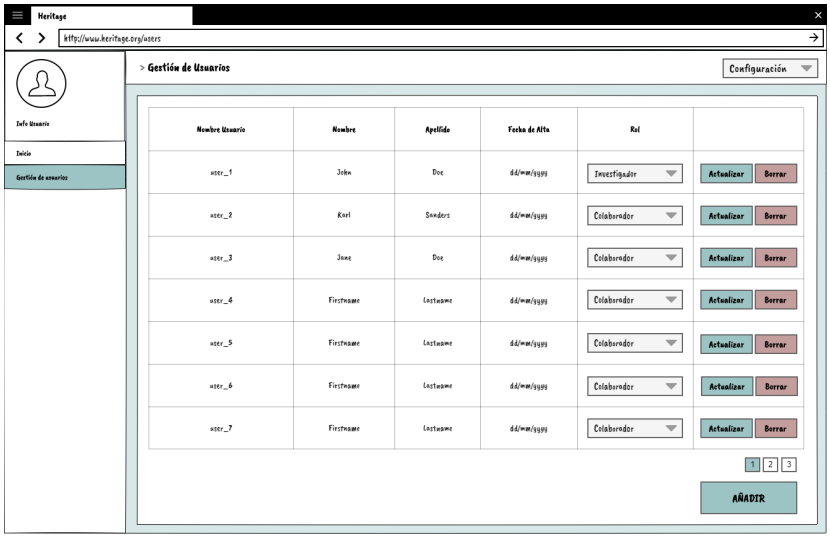


Figura 5.1: Maqueta de la Lista de Usuarios para el Usuario Administrador

Para el usuario investigador es necesario que **se muestren tanto las colecciones subidas por él mismo como las que le han compartido**, de manera que pueda diferenciarlas de manera simple. Esto lo hemos conseguido mediante dos listas en la misma página (también paginadas), de manera que pueda acceder a casi cualquier colección sin necesidad de cambiar de página, pero que tenga las colecciones propias separadas de las que son compartidas (véase figura 5.2).

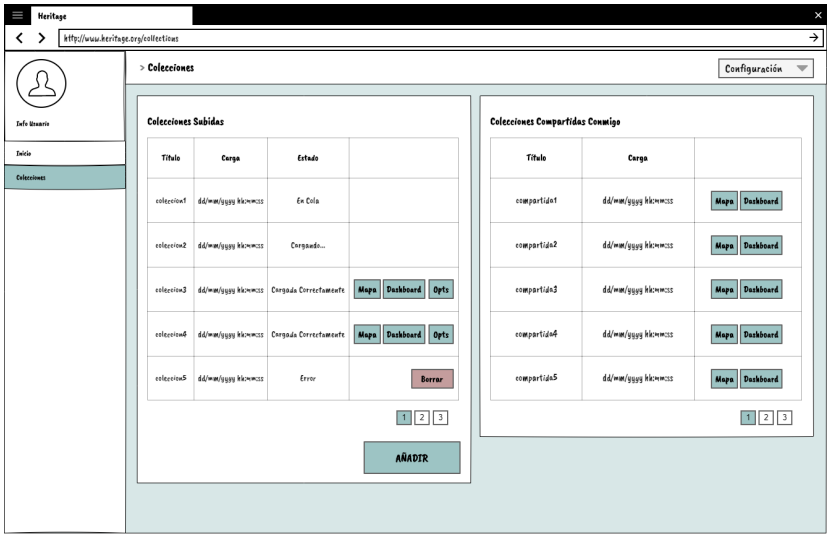


Figura 5.2: Maqueta de las Listas de Colecciones Propias y Colecciones Compartidas para el Usuario Investigador

Para el usuario colaborador se reutiliza la misma página pero ésta oculta la lista de colecciones propias puesto que un colaborador no puede tenerlas (véase figura 5.3).

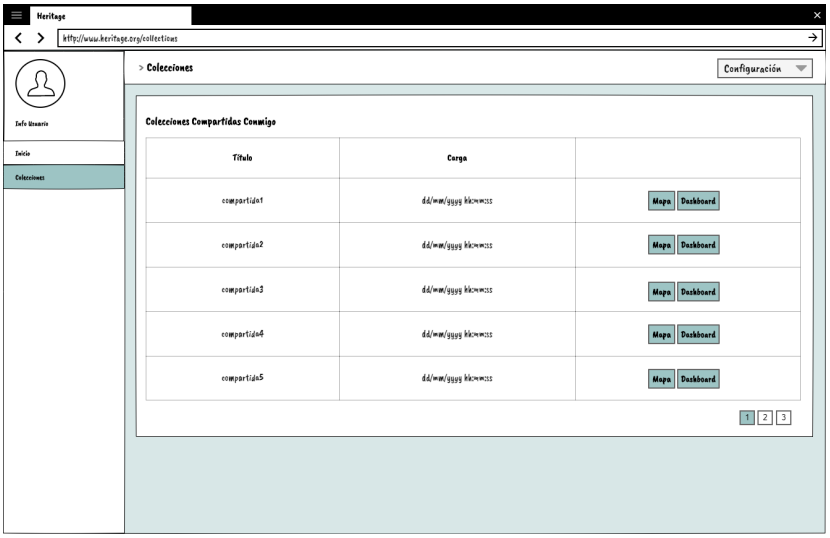


Figura 5.3: Maqueta de la Listas de Colecciones Compartidas para el Usuario Colaborador

La **vista del mapa** permite consultar la localización geográfica de cada una de las entidades de tipo localización extraídas de una colección. Deseamos que estas entidades se muestren en el mapa como chinchetas interactuables que al hacer click en una nos muestren la información sobre la entidad en la que tenemos interés, permitiendo seguir viendo el mapa y el resto de chinchetas a la vez (véase figura 5.4).

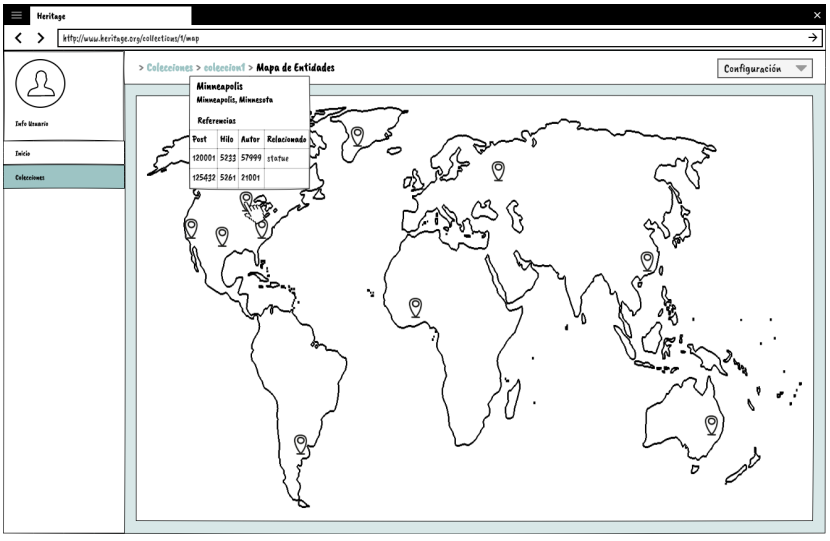


Figura 5.4: Maqueta del Mapa de Entidades de una colección

La **vista del dashboard** nos mostrará diferentes estadísticas de la colección seleccionada junto con varios gráficos. Se presentan estas visualizaciones de manera conjunta y a la vez diferenciada para poder contemplarlas a la vez sin necesidad de obligar al usuario a cambiar de

página o realizar acciones complejas para consultar cada uno de los datos (vista de *dashboard*, figura 5.5). Se muestran los siguientes gráficos:

- Gráfico de burbujas con ejes de **Distribución de Autores, Posts y Entidades por Hilo**: Permite para cada thread (representado por una burbuja) mostrar información de sus autores (eje Y), posts (eje X) y entidades (tamaño de la burbuja) extraídas. Si se pasa el cursor por encima de una burbuja se ve el número exacto para esas 3 variables.
- Gráfico de sectores de **Número de Referencias a Entidades por Tipo**: Permite para cada tipo de entidad (Localización, Organización, Persona) saber cuántas referencias a entidades hay de cada uno y su proporción o porcentaje sobre el total.
- Gráfico de burbujas de **Número de ocurrencias de Términos Relacionados del Tesauro**: Permite para cada término extraído del tesauro (representado por una burbuja) mostrar el número de total de apariciones del término en la colección determinado por el tamaño de la burbuja, siempre que el término esté relacionado con alguna entidad. Si se pasa el cursor por encima de una burbuja se ve el número exacto de ocurrencias.

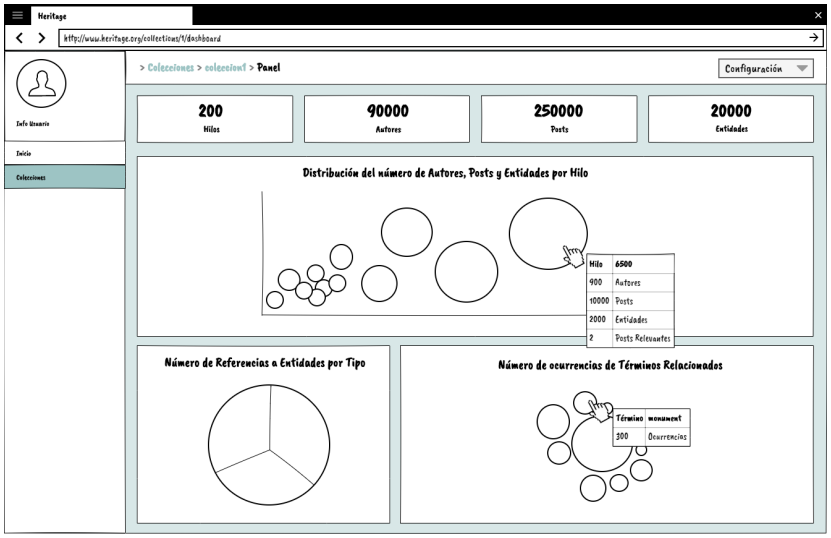


Figura 5.5: Maqueta del Dashboard de una colección

5.2 Arquitectura

La arquitectura elegida para la aplicación ha sido un cliente-servidor (figura 5.6). El lado cliente o *Front-End* proporciona la interfaz de usuario a los navegadores y se comunica con el *Back-End*. El lado servidor o *Back-End* sigue una arquitectura en capas (Capa Servicios o API

REST, Capa de Lógica de Negocio y Capa de Acceso a Datos), utilizando la API REST para la comunicación con el cliente, así como diversos servicios y otros componentes para la lógica de negocio y DAOs o repositorios para el acceso a las bases de datos. Para cada uno de los diagramas se ha utilizado la herramienta **draw.io** (véase sección 2.1.4).

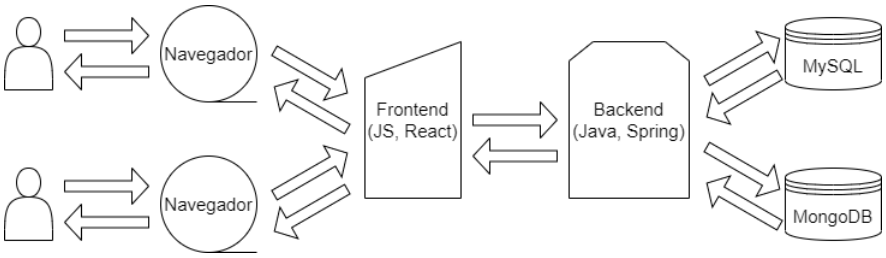


Figura 5.6: Arquitectura Global (Cliente-Servidor) de la aplicación

La aplicación se ha desarrollado utilizando un proyecto multimódulo de Gradle, que se compone de los siguientes subproyectos:

- *heritage-core-model*: Escrito en Java. Contiene el modelo del *Back-End*, que tiene las capas de lógica de negocio y de acceso a datos. Se subdivide en 2 módulos:
  - *heritage-core-model-api*: Expone las interfaces y DTOs que están accesibles a la capa de servicios (módulo *heritage-core-ws*).
  - *heritage-core-model-impl*: Implementación de las interfaces del modelo y dependencias.
- *heritage-core-ws*: Escrito en Java. Contiene la capa de servicios de la aplicación.
- *heritage-core-util*: Escrito en Java. Contiene clases de utilidad comunes a los otros módulos de Java.
- *heritage-web-client*: Escrito en JavaScript. Módulo que contiene todo el código del *Front-End*.

5.2.1 Arquitectura del Lado del Servidor

En el *Back-End* podemos distinguir una capa de servicios (o capa *Web*) y capa de modelo (subdividida a su vez en capa de lógica de negocio y capa de acceso a datos). En la figura 5.7 puede verse un diagrama de clases detallado sobre la arquitectura de esta parte.

La capa de servicio es la encargada de la comunicación con el cliente, construyendo las respuestas adecuadas a las peticiones de éste. Se desarrolló utilizando una API REST, una forma de comunicación utilizando el protocolo HTTP. Mediante *Spring Web* ha sido posible

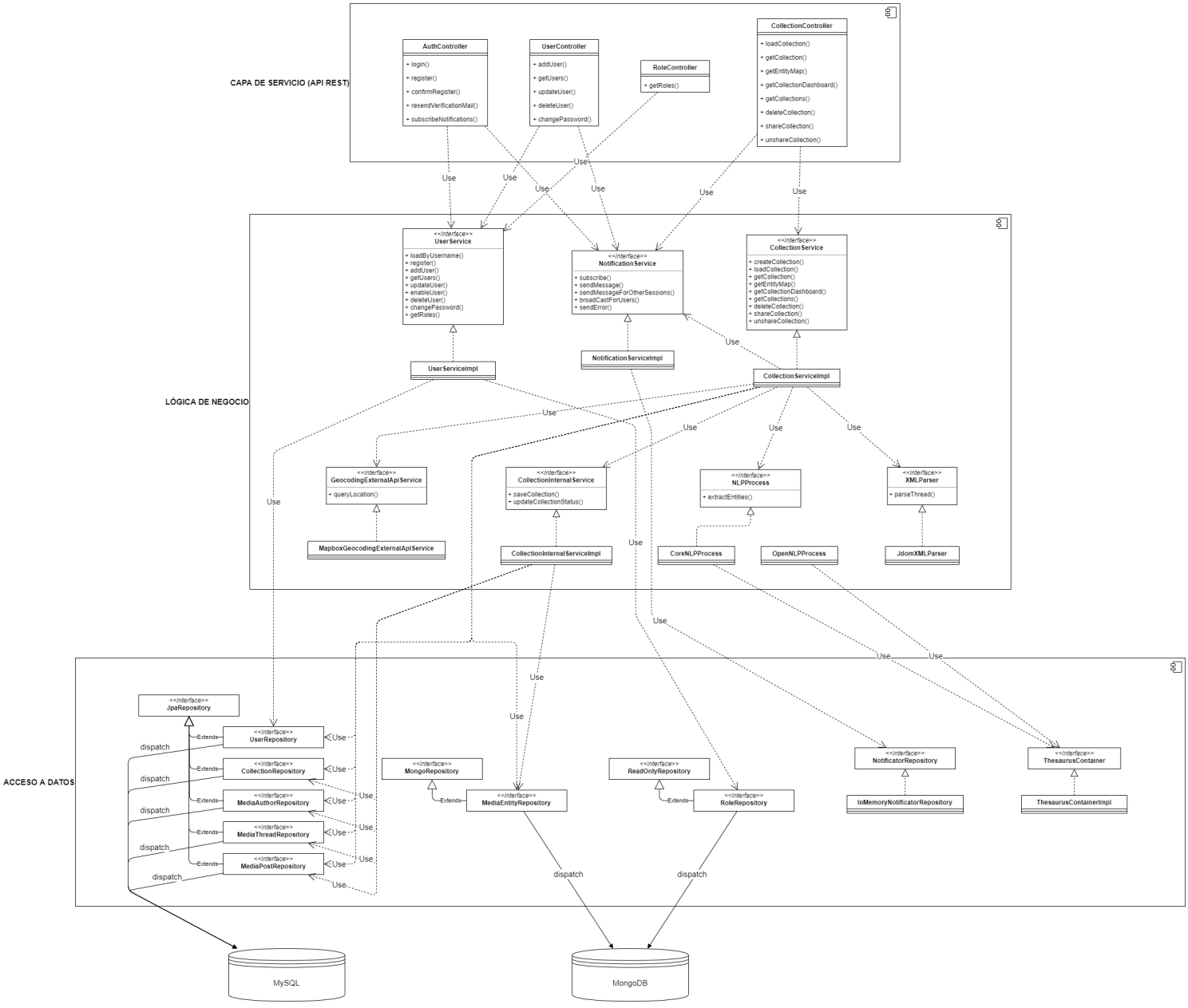


Figura 5.7: Arquitectura del Lado del Servidor

construir los diferentes controladores REST para el desarrollo de la API. El módulo *heritage-core-ws* que implementa esta parte contiene los siguientes paquetes relevantes:

- *com.mffernandez.heritage.core.ws.api.controller*: Contiene los controladores del API REST que realizan operaciones propias de la lógica de la aplicación para usuarios identificados en el sistema. Estos controladores ejecutan un filtro de identificación con *Spring Security* antes de atender la petición.
- *com.mffernandez.heritage.core.ws.auth.controller*: Contiene los controladores del API REST que realizan operaciones necesarias para ayudar a los usuarios a identificarse. Realizan operaciones de inicio de sesión o registro.
- *com.mffernandez.heritage.core.ws.handler*: Control de errores en los controladores.
- *com.mffernandez.heritage.core.ws.security*: Configuración de *Spring Security*.
- *com.mffernandez.heritage.core.ws.server*: Contiene el punto de entrada para la aplicación que arranca el servidor (*Spring Boot Application*).

La capa de modelo es la encargada de realizar las operaciones relacionadas con la lógica de negocio, así como el acceso a datos. El módulo *heritage-core-model* que implementa esta parte contiene los siguientes paquetes relevantes:

- *com.mffernandez.heritage.core.model.component*: Contiene diversas dependencias necesarias en otras clases del modelo.
- *com.mffernandez.heritage.core.ws.model.document*: Contiene objetos que utiliza *Spring Data* para transformar entidades de MongoDB a objetos Java.
- *com.mffernandez.heritage.core.ws.model.entity*: Contiene objetos que utiliza *Hibernate* para transformar entidades de bases de datos relacionales a objetos Java.
- *com.mffernandez.heritage.core.model.nlp*: Contiene los componentes que realizan la lógica de procesado NLP.
- *com.mffernandez.heritage.core.model.repository*: Contiene los DAO (o repositorios) que utiliza *Spring Data* para la capa de acceso a datos.
- *com.mffernandez.heritage.core.model.service*: Contiene los servicios que llevan a cabo cada una de las operaciones de la lógica de negocio.

5.2.2 Arquitectura del Lado del Cliente

El lado del cliente está alojado en el módulo de Gradle *heritage-we-client*. Se trata de un proyecto React escrito en JavaScript, utilizando React Hooks (nos permite escribir los componentes de React como si fueran funciones y no clases). React está basado en componentes y como tal ésta es la parte fundamental del *Front-End*. Dentro del cliente podemos dividir en los siguientes paquetes (o directorios):

- *components*: Contiene los distintos componentes utilizados por React. Se dividen en varios subdirectorios dependiendo de su propósito (e.g. *auth* para los componentes de identificación del usuario en el sistema), para poder localizarlos de manera sencilla.
- *lib*: Contiene módulos de utilidad (*helpers*) para los componentes, así como los reductores y los manejadores de acciones de Redux. También tiene los módulos *client*, que son los que utilizamos para construir las peticiones REST para enviar al servidor y recibir las respuestas. Al igual que los componentes, estos archivos se dividen en varios subdirectorios dependiendo de su función.
- *resources*: Archivos JSON de internacionalización, imágenes y hojas de estilo SASS.
- *setup*: Configuración del *store* o almacenamiento de Redux, de la internacionalización y del cliente de MapBox.

Internacionalización

Hemos diseñado el *Front-End* de la aplicación de manera que pueda adaptarse al idioma y región elegido por el usuario en su navegador, sin necesidad de realizar cambios de configuración ni de código en el proyecto.

Para ello se ha utilizado el framework *18next* [56]. Está escrito en JavaScript y permite la internacionalización de aplicaciones basadas en este lenguaje de manera simple con una pequeña configuración y mediante la utilización de archivo JSON.

Tenemos configurado una archivo JSON para cada idioma, con cada una de las traducciones para todas las posibles cadenas de texto que pueda presentar la aplicación. Actualmente, y para realizar las pruebas, se soportan los siguientes idiomas: Inglés (*en*), Español (*es*) y Gallego (*gl*). El Inglés está configurado como idioma por defecto y en caso de que el usuario tenga elegido un idioma no soportado se utilizará este.

Capítulo 6

Planificación

En este capítulo se exponen la planificación y el presupuesto inicial del proyecto.

6.1 Planificación inicial

Una vez definido el alcance y objetivos del proyecto, se lleva a cabo una planificación inicial. Se hace un análisis y una estimación de los recursos disponibles para el desarrollo del proyecto. Debido a que tenemos un equipo de desarrollo unipersonal, contamos con una obvia limitación de recursos.

Al utilizar la metodología Scrum, el desarrollo se divide en *sprints*. Cada uno de los *sprints* los hemos estimado en una cifra alrededor de los 60 puntos. El desarrollo ha llevado un total de 5 *sprints* con un total de 298 puntos. Suponiendo que cada punto equivale aproximadamente a 1 hora de trabajo, entonces el proyecto se llevará acabo en 5 *sprints* con **298** horas de trabajo totales.

6.2 Presupuesto

Para evaluar los costes del proyecto es necesario tener en cuenta personal, material y licencias.

Para los costes de personal, sabemos que el proyecto dispone de dos recursos humanos: un **Jefe de Proyectos**, encargado de la supervisión del mismo; y un **Analista Programador**, responsable de Análisis, Diseño, Implementación y Pruebas. Según el portal *Glassdoor* [57], el salario bruto anual medio a nivel nacional del perfil de Jefe de Proyectos en 2021 se corresponde con unos 38000€, mientras que el de un Analista o Programador con unos 29000€. Suponiendo una jornada anual de 1800 horas, y un tiempo de dedicación al proyecto por parte del Jefe de Proyectos de 24 horas y del Analista Programador de las 298 horas totales de los *sprints*, tenemos el siguiente coste en personal:

$$\frac{38000€}{1800h} * 24h + \frac{29000€}{1800h} * 298h = 5307,\widehat{7}€ \approx \mathbf{5308}€$$

Con respecto al material, para obtener el coste hemos aplicado el método de amortización lineal o constante [58]. Siendo el precio del equipo personal 1000€ para una vida útil de 5 años; y su tiempo de utilización 5 *sprints* de 2 semanas de duración cada uno, es decir, 10 semanas, nos daría de resultado:

$$\frac{1000€}{5años * 52semanas/año} * 5sprints * 2semanas/sprint = 38,4615€ \approx \mathbf{38}€$$

Al utilizar herramientas de código abierto y gratuitas, el coste en herramientas software y licencias supone un total de **0**€.

Entonces, el coste total estimado para el presupuesto del proyecto sería:

$$5308€(personal) + 38€(material) + 0€(licencias) = \mathbf{5346}€$$

6.3 Seguimiento

Como se comentó previamente, el desarrollo del proyecto se dividió en un total de 5 *sprints* de una duración aproximada de 2 a 3 semanas, los cuáles describimos a continuación junto con historias de usuario implicadas (véase sección 4.2.2):

- **Sprint 1**: Se implementaron las historias de usuario de inicio de sesión (HU 1.1), registro (HU 1.2), cierre de sesión (HU 1.3) y borrado de cuenta (HU 1.4).
- **Sprint 2**: Se abordó el cambio de contraseña (HU 1.5) y todas las correspondientes a la Épica 2 (Gestión de Usuarios).
- **Sprint 3**: Se implementaron las historias de usuario correspondientes a la Épica 3 (Manejo de Colecciones).
- **Sprint 4**: Se implementaron las historias de usuario para la visualización del mapa (HU 4.1) y del dashboard (HU 4.2).
- **Sprint 5**: Se implementaron las historias de usuario restantes menos prioritarias: verificación de registro por email (HU 1.6) e inclusión del tesauro (HU 4.3).

Se comentan los detalles del trabajo realizado en cada *sprint* en el capítulo 7: Implementación.

Capítulo 7

Implementación

En este capítulo se comenta el trabajo realizado durante la implementación de la aplicación. Se detallan los aspectos técnicos más importantes ligados a la ejecución de cada uno de los *sprints* (para los detalles de cada historia de usuario véase sección 4.2.2).

7.1 Sprint 1

El primer *sprint* se llevó a cabo entre el 3 y el 20 de Mayo y se centró en las historias de usuario: 1.1, 1.2, 1.3 y 1.4. Estas historias de usuario implicaron básicamente la creación de la entidad "Usuario" en base de datos, junto con su persistencia y recuperación. El inicio y el cierre de sesión implican mantener un objeto con la información de usuario en Redux y borrarlo, respectivamente. Se han creado además formularios para el inicio de sesión y registro.

Para mantener la sesión del usuario es necesario mantener algún código de sesión que permita identificarlo en el estado de la aplicación de React. Para conseguir esto hemos utilizado JSON Web Tokens (JWT). JWT [59] es un estándar que permite compartir información como un objeto JSON encriptado. Estos objetos JSON pueden contener toda la información del usuario y se encriptan y desencriptan utilizando una clave que nosotros tendremos sólo disponible desde el servidor (véase figura 7.1).

Cuando el usuario inicia sesión, el servidor genera un código de acceso y lo envía al cliente con la respuesta. El cliente lo persiste en el estado de Redux y lo envía al servidor de vuelta con cada petición mientras el usuario mantenga su sesión iniciada. El servidor no puede mantener la sesión del usuario y para identificarlo con las peticiones que envía es necesario utilizar este código, de ésta manera evitamos compartir la contraseña del usuario en cada petición, enviándose sólo una vez al iniciar sesión.

En Redux tenemos dos almacenamientos posibles, uno se mantiene al cerrar el navegador (*Almacenamiento local*) y el otro sólo se mantiene en cada una de las pestañas y se destruye al cerrarla (*Almacenamiento de sesión*). De esta manera podemos implementar la opción

```
@Override
public String generateToken (final GenerateTokenParameters parameters, final
    boolean isAuthentication)
{
    return Jwts.builder()
               .setSubject(parameters.getUsername())
               .setIssuedAt(new Date())
               .setExpiration(new Date((new Date()).getTime() +
                                       (isAuthentication ?
    this.authenticationJwtExpirationMs :

    this.verificationJwtExpirationMs)))
               .signWith(SignatureAlgorithm.HS512,
                         isAuthentication ? this.authenticationJwtSecret :
    this.verificationJwtSecret).compact();
}
```

Figura 7.1: Método para generar códigos JWT

"Recuérdame" al iniciar sesión, almacenándola en el segundo si no se marca la opción (véase figura 7.2).

```
const login =
    ({ username, password, rememberMe }) =>
    (dispatch) =>
            . . .
            authClient
                .login(username, password)
                .then((user) => {
                    dispatch({
                        type: rememberMe
                            ? AUTH_LOGIN_REMEMBER_SESSION
                            : AUTH_LOGIN_NOT_REMEMBER_SESSION,
                        data: {
                            user: user,
                        },
                    });

                    resolve();
                })
                . . .
        });
```

Figura 7.2: *Dispatcher* de acciones de inicio de sesión en Redux

7.2 Sprint 2

El el segundo *sprint* tuvo lugar entre el 18 de Junio y el 9 de Julio y se abordaron las historias de usuario: 1.5, 2.1, 2.2, 2.3 y 2.4. Las funcionalidades de éstas últimas 4 historias sólo están disponibles para el usuario administrador y como tal se hace necesario introducir algún mecanismo para manejar roles y autorización. Implica la creación de la entidad "Rol" junto con sus relacionadas "Autoridad" y "Función".

Se han creado los 3 roles para usuarios identificados en base de datos (Administrador, Investigador y Colaborador). Para cada uno de ellos es necesario relacionarlo con las autoridades que representan las funcionalidades que tienen disponibles. Para un usuario Administrador se pueden ver en la figura 7.3.

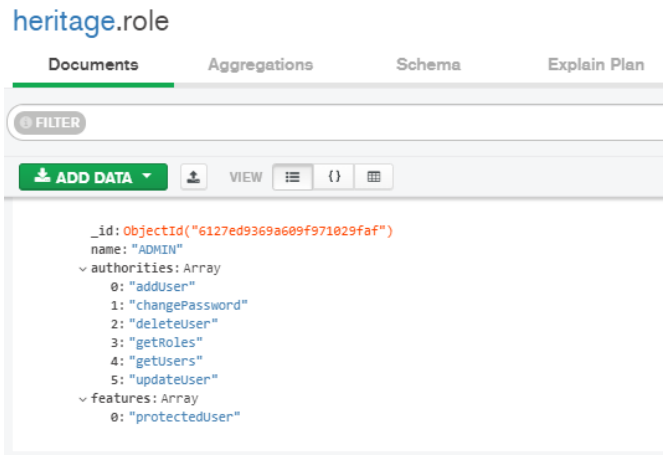


Figura 7.3: Autoridades del Rol Administrador en la base de datos de MongoDB

Con *Spring Security* podemos hacer uso de la anotación *PreAuthorize*. Esta anotación permite que el método del controlador se ejecute sólo si el usuario está identificado (tiene un código de acceso JWT) y cumple la condición especificada (véase figura 7.4).

```
@GetMapping
@ResponseBody
@PreAuthorize("hasAuthority(T(com.mffernandez.heritage.core.model.constant"
    + ".RoleAuthorities).GET_USERS)")
@SneakyThrows
public GetUsersPageResponse getUsers (@RequestParam("roleFeature") final
    String[] roleFeatures,
        @RequestParam final int page, @RequestParam final int pageSize,
        @RequestParam(defaultValue = "userId") final String sortField,
        @RequestParam(defaultValue = "ASC") final String sortDirection)
{
    final GetUsersPageData getUsersPageData =
            this.userService.getUsers(roleFeatures, page, pageSize,
    sortField, sortDirection);
    . . .
}
```

Figura 7.4: Métdo utilizando la anotación *PreAuthorize* para filtrar por rol de usuario

Como los usuarios administradores son los únicos que tienen esa autoridad, éste y otros métodos con autoridades asociadas con el rol Adminstrador sólo podrán ser ejecutados si un usuario de rol Adminstrador está identificado.

La lista de autoridades específicas de un rol también se envía al cliente cuando el usuario se identifica, pues son necesarias para comprobar qué elementos son visibles y cuáles no en

la Web. Para esta parte hemos creado un objeto JSON grande llamado *Modelo de Navegación*, que registra cada una de las rutas de la página y componentes y comprueba que roles pueden acceder a ellos mediante funciones almacenadas en él (véase figura 7.5).

```
const navigationModel = {
    . . .
    userManagement: {
        translate: true,
        path: () => "/users",
        icon: "manage_accounts",
        children: {
            addUserForm: {
                visible: ({ user }) => user.authorities.includes(ADD_USER),
            },
            updateRoleSelect: {
                visible: ({ user }) =>
    user.authorities.includes(UPDATE_USER),
            },
            . . .
        },
        sidebarVisible: ({ user }) => user.authorities.includes(GET_USERS),
        allowed: (params) =>
            navigationModel.userManagement.sidebarVisible(params),
    },
    . . .
}

```

Figura 7.5: Modelo de Navegación en el *Front-End*

7.3 Sprint 3

El tercer *sprint* se realizó entre el 15 y el 30 de Julio. Se implementaron las historias de usuario 3.1 y 3.2. De aquí cabe destacar la historia 3.1, que es la de la carga de la colección, que implicaba la extracción del archivo ZIP con los XML de prueba, su análisis sintáctico (*parsing*) de XML, NLP, obtención de entidades y coordenadas, y la persistencia de toda esta información.

Después de implementar los componentes de NLP utilizando las librerías CoreNLP y OpenNLP y probarlos, hemos decidido que el proyecto utilice el implementado con CoreNLP por su mayor fiabilidad de resultados (véase figura 7.6 y sección 2.1.1).

El procesado de la colección de prueba completa, aún haciéndolo de manera concurrente con bastantes hilos, es muy costoso y tarda mucho (aproximadamente 15 minutos). Debido a esto, se hace necesario que el servidor lance esta carga de la colección de manera asíncrona, enviando una respuesta según empieza al cliente y permitiendo a este seguir trabajando, mientras el servidor ejecuta el procesado de la colección y la persistencia de la información en segundo plano. Esto ha sido posible mediante una pequeña configuración en *Spring* y el uso de la anotación *Async* (véase figura 7.7).

```
. . .
processedPost.setPostId(post.getPostId());
processedPost.setAuthorId(post.getAuthorId());

final CoreDocument document = new CoreDocument(post.getBody());
pipeline.annotate(document);
processedPost.setEntities(extractEntities(entityTypes, thesaurusContainer,
    document.entityMentions()));
. . .
```

Figura 7.6: Procesado con CoreNLP

```
@Override
@Async
public void loadCollection (final LoadCollectionParameters parameters)
{
    . . .

    this.loadCollectionQueue.block(parameters.getCollectionId());
    boolean blocked = true;
    . . .
}
```

Figura 7.7: Ejemplo de método que se ejecuta en segundo plano con anotación *Async*

Para poder notificar al cliente que lanzó la petición cuando la carga de la colección está terminada hemos utilizado Server-Sent Events (SSE). Es un protocolo en el que un cliente puede suscribirse al servidor para así éste poder mandarle mensajes de tipo MIME *text/event-stream*. Hemos implementado SSE para éste y otros propósitos, haciendo que el cliente se suscriba al servidor y éste mande notificaciones cuando eventos importantes ocurran, como cuando se termine la carga de una colección (véase figura 7.8).

```
@Override
@Async
public void loadCollection (final LoadCollectionParameters parameters)
{
    . . .
    final SaveCollectionData saveCollectionData =

    this.collectionInternalService.saveCollection(saveCollectionParameters);

    mainTimeWatch.stopAndLog();

    final LoadCollectionNotification message = new
    LoadCollectionNotification();
    message.setEvent(Notifications.SUCCESSFULLY_LOADED_COLLECTION);
    this.adapterContainer.adapt(saveCollectionData, message);

    this.notificationService.sendMessage(saveCollectionData.getOwnerId(),
    message);
    . . .
}
```

Figura 7.8: Ejemplo de método con SSE

7.4 Sprint 4

El cuarto *sprint* se realizó entre el 2 y el 16 de Agosto, y abarcó la historias de usuario referentes a la épica del Análisis y Visualización de Colecciones (4.1 y 4.2). Implicó la obtención del mapa de MapBox y el pintado de las entidades de localización sobre él, así como la visualización de los gráficos en el *Dashboard*.

Para el caso del mapa tenemos las entidades de localización previamente persistidas junto con sus coordenadas (extraídas mediante la API de Geocodificación de MapBox). Éstas se envían ahora al cliente, para que el componente de mapa que utiliza Mapbox pueda pintar un puntero y un *pop-up* por cada una de las entidades (véase figura 7.9).

```
. . .
new mapboxgl.Marker(element)
    .setLngLat([entity.longitude, entity.latitude])
    .setPopup(
        new mapboxgl.Popup({
            closeButton: false,
            offset: 25,
        }).setHTML(
            `<h6>${entity.name}</h6><p>${
                entity.description
            }</p>${references.length > 0 ? references : ""}`
        )
    )
    .addTo(map.current);
. . .

```

Figura 7.9: Renderizado de las entidades en MapBox

Para el dashboard y poder pintar los gráficos, se ha hecho uso de la librería D3. Esta librería permite crear gráficos a partir de datos proporcionados, utilizando el elemento del DOM *svg* (véase figura 7.10).

7.5 Sprint 5

El quinto y último *sprint* se realizó entre el 23 de Agosto y el 6 de Septiembre, abarcando las 2 historias menos prioritarias pendientes (1.6 y 4.3). Estas historias implicaron la verificación de usuarios recién registrados por E-Mail y la clasificación de las entidades extraídas de una colección de referencia mediante términos de un tesauro.

Respecto al tesauro, en trabajos de investigación previos y publicados [5], para la extracción de hilos relevantes se utilizó un conjunto de palabras semilla de temática relevante sobre *Black Lives Matter* (véase figura 7.11). A partir de estas palabras semilla, hemos utilizado las que hacen referencia a entidades patrimoniales (e.g. *statue*, *monument*), y las hemos expandido utilizando la red WordNet [16]. WordNet es una red semántica para el idioma inglés, que

```
export const createChart = (node, data, t) => {
    nodeRoot = node;
    i18 = t;
    svg = d3
        .select(nodeRoot)
        .append("svg")
        .attr("width", setup.width)
        .attr("height", setup.height);

    const defs = svg.append("defs");
    createShadow(defs);

    createScales(data);
    createAxis();
    createLabels();
    createPopup(nodeRoot);
    createCircles(data);
    createLegend();
};
```

Figura 7.10: Renderizado de gráfico con D3

permite obtener todas las palabras que tengan alguna relación semántica con una dada. Para confirmar nuestro tesauro, hemos tomado para cada término original todos los términos que WordNet nos proporciona para las siguientes categorías:

- *direct hyponym / full hyponym*
- *has instance*
- *direct hypernym / inherited hypernym / sister term*
- *derivationally related form*

Así, de los 16 términos del tesauro original relacionados con entidades hemos elegido 6 que refieren precisamente a entidades patrimoniales (*monument*, *statue*, *tribute*, *memorial*, *plaque*, *bust*). Esos 6 términos es posible expandirlos mediante el método descrito y obtener un tesauro final de 228 términos.

Hemos construido un componente que permite analizar las entidades y ver si están relacionadas con los términos incluidos en este tesauro. Para ello, como CoreNLP nos devuelve anotado junto a cada entidad la oración a la que pertenece, resulta fácil buscar en esta oración los términos del tesauro y relacionarlos con la entidad (véase figura 7.12).

Table 1. Lists of terms for both groups. Terms in the same row does not mean we have use them together.

Conflicts related terms	Entity related terms
anti-black	monument
blacklives	removed
blacklivesmatter	removal
police brutality	statue
polive violence	tribute
abuse of authority	memorial
racism	plaque
racial bias	bust
anti-racism	take down
george floyd	beheaded
slavery	desecrated
slave	vandalized
-	vandalism
-	vandals
-	protests
-	protesters

Figura 7.11: Palabras semilla del tesauro original para la extracción de colecciones de referencia sobre *Black Lives Matter*

```
. . .
for (final String term : this.thesaurus.getTerms()) {
    scanForTerm(sentenceTokens, relatedTerms, term);
}
. . .
private static void scanForTerm (final List<String> sentenceTokens, final
    Set<String> relatedTerms, final String term)
{
    final String[] termTokens = term.split("\\s+");
    for (int i = 0; i < sentenceTokens.size(); i++) {
        boolean containsTerm = true;
        for (int j = 0; j < termTokens.length && i + j <
    sentenceTokens.size(); j++) {
            if (!sentenceTokens.get(i + j).equalsIgnoreCase(termTokens[j])) {
                containsTerm = false;
                break;
            }
        }
        if (containsTerm) {
            relatedTerms.add(term);
        }
    }
}
```

Figura 7.12: Búsqueda de términos del tesauro

Capítulo 8

Pruebas

En este capítulo se comentan las distintas pruebas realizadas durante el proyecto. Las pruebas son una parte fundamental en la Ingeniería de Software. Están pensadas para ayudar a detectar errores, no para demostrar la ausencia de ellos. Como tal no podemos demostrar de ninguna manera que la aplicación final no tenga errores, pero gracias a las distintas pruebas que hemos realizado sí aumenta la confianza y la calidad del producto. Se han realizado pruebas de unidad, integración, aceptación y calidad.

8.1 Pruebas de Unidad

Las pruebas de unidad permiten probar un componente de software de manera aislada, sin necesidad de sus dependencias. Para ello es necesaria la construcción de componentes simulados (*mocks*) de estas dependencias, que devuelven el valor esperado a sus llamadas. En el lado del servidor, para esto se utilizó la librería *Mockito* (sección 2.1.1) junto con *Junit*.

Las pruebas de unidad son un componente fundamental de TDD (véase 3.2), y como tal las hemos ido realizando durante el desarrollo del código del lado de servidor siguiendo la premisa *Red –Green –Refactor*.

Para el cliente hemos realizado también pruebas de unidad utilizando la librería *Jest* junto con *Testing Library*. *Jest* permite la construcción de *mocks* en JavaScript y *Testing Library* hacer evaluaciones sobre el árbol DOM de manera fácil. Sobre el lado del cliente no hemos seguido TDD debido a que eran una pruebas algo más complejas que las del lado del servidor, y necesitábamos conocer como estaban los elementos de HTML dispuestos, por lo que no podíamos hacer las pruebas *a priori*.

8.2 Pruebas de Integración

Las pruebas de integración sirven para probar simultáneamente varios componentes del sistema interconectados. Aquí ya no es necesario el uso de librerías para crear componentes simulados.

Nuestras pruebas de integración las hemos realizado sólo sobre el lado del servidor, y consistieron básicamente en probar cada una de las operaciones de la API REST.

Debido al uso de *Spring* y su sistema de inyección de dependencias, necesitamos algún mecanismo para simular la aplicación de producción con resolución de dependencias en un entorno de pruebas. Esto se consigue de forma fácil con *Spring Test* y más concretamente con la anotación *SpringBootTest*, que permite lanzar la aplicación *SpringBoot* en un clase de *Testing*. Hemos utilizado también *RestTemplate*, que permite hacer peticiones REST sobre código Java.

Se han utilizado bases de datos específicas para *Testing*, diferentes a las que utilizamos para las pruebas de aceptación.

8.3 Pruebas de Aceptación

Las pruebas de aceptación consisten en probar todo el conjunto de la aplicación desde el punto de vista del usuario final.

Hemos realizado pruebas de aceptación que han consistido básicamente en probar cada uno de los criterios de aceptación definidos para las historias de usuario (véase sección 4.2.2). Una histora de usuario se da como validada cuando cumple correctamente y sin errores con todos sus criterios de aceptación.

Las pruebas de aceptación han sido dirigidas por el director del proyecto.

8.4 Pruebas de Calidad

Las pruebas de calidad consisten en buscar malas prácticas o errores (pruebas de caja blanca) sobre el código implementado. Se ha configurado un servidor de *SonarQube* para esto (véase 2.1.4), junto con la extensión de *SonarLint* para *IntellijIDEA* sincronizada con este servidor (figura 8.1).

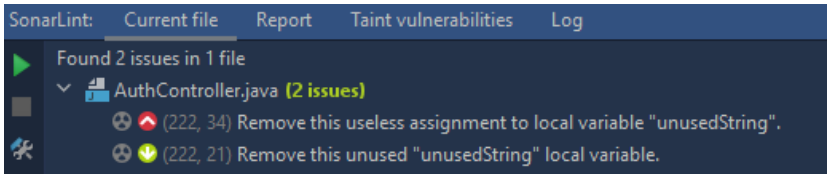


Figura 8.1: Ejemplo de error identificado por la extensión *SonarLint*

Capítulo 9

Producto Final

En este capítulo se muestran las funcionalidades de la aplicación implementada y se detalla su funcionamiento. Esta sección servirá como guía de uso y describirá las características principales del software desarrollado. Se acompañará de capturas que servirán de apoyo visual para la explicación.

9.1 Inicio de Sesión y Registro

Para el inicio de sesión y el registro, el usuario dispone de formularios sencillos (figura 9.2) y accesibles cómodamente desde la página principal de acceso (figura 9.1).

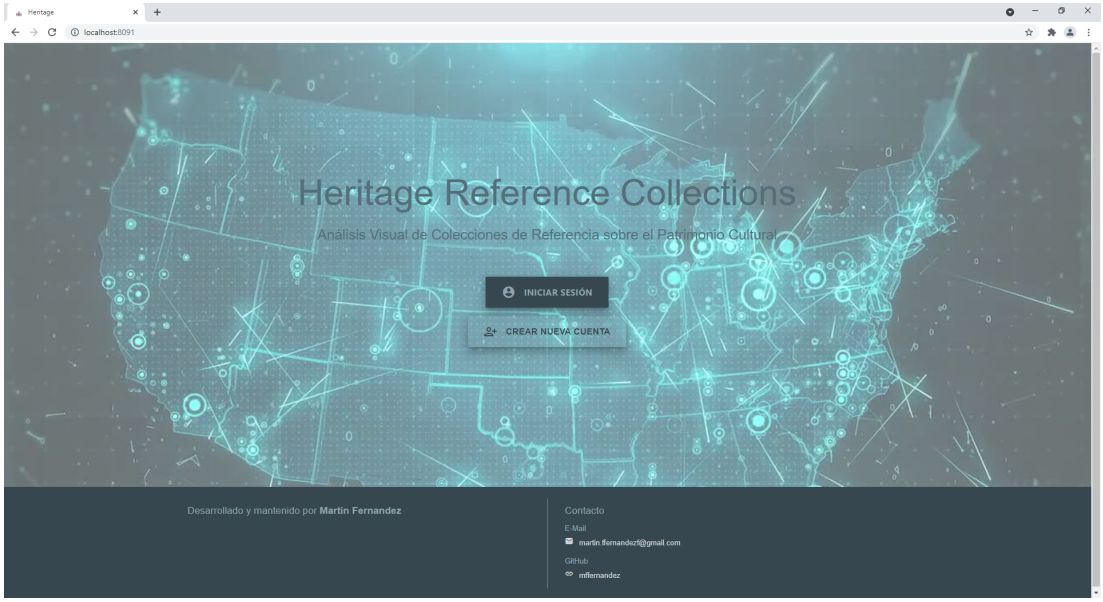
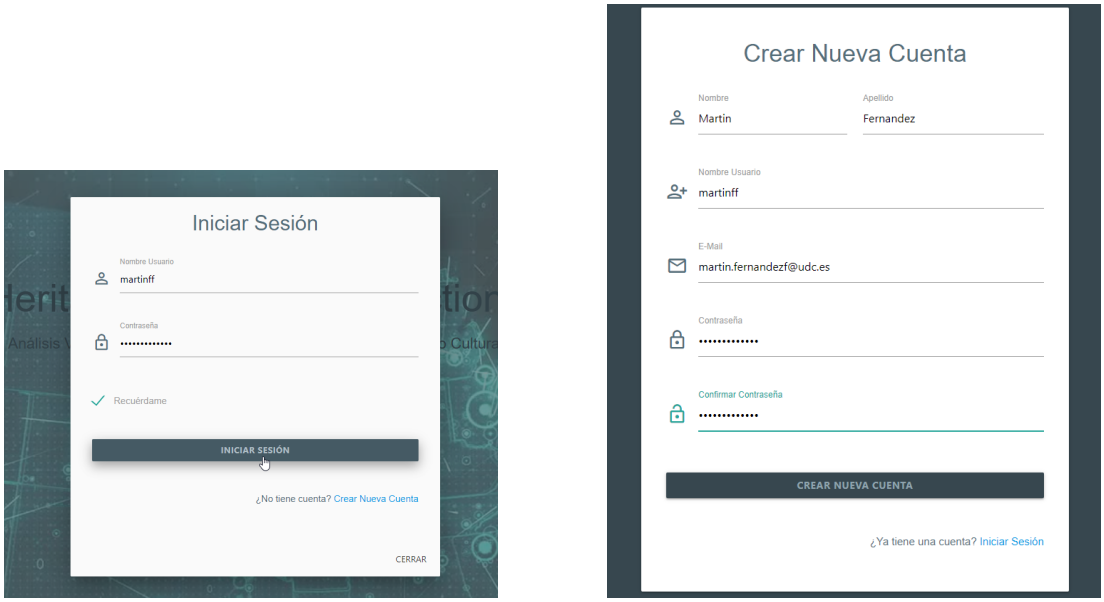


Figura 9.1: Página de Inicio

Cuando el usuario inicia sesión con una cuenta verificada se redirige a la página de *Home*



(a) Formulario de Inicio de Sesión

(b) Formulario de Registro

Figura 9.2: Formularios de Inicio de Sesión y Registro

(figura 9.3).

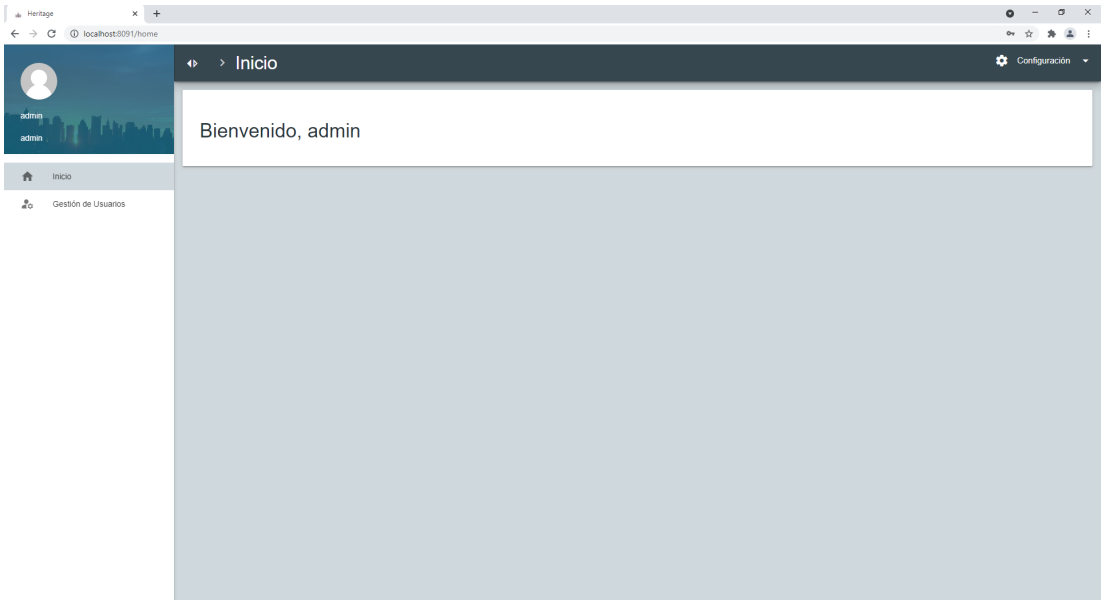


Figura 9.3: Página de Inicio de Usuario Identificado

Si por el contrario, se acaba de registrar o inicia sesión sin tener aún verificada su cuenta de correo electrónico, se muestra un aviso de que primero tiene que realizar este paso (figura 9.4).

Figura 9.4: Aviso para que el usuario verifique su cuenta de Email

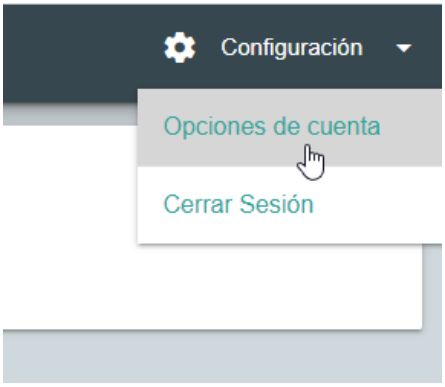


Figura 9.5: Menú de Configuración y Cierre de Sesión

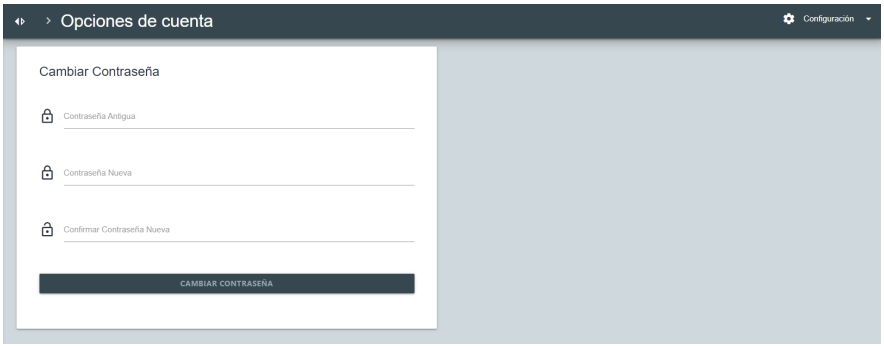
9.2 Configuración de la Cuenta

Todo usuario identificado tiene un menú para acceder a la configuración de su cuenta o al cierre de sesión (figura 9.5).

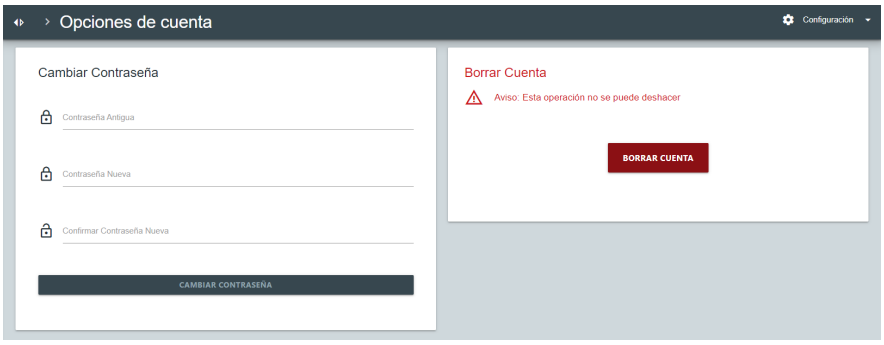
El usuario adminstrador no puede borrar su cuenta, así que no tiene la función de "eliminar cuenta" disponible (figura 9.6).

9.3 Gestión de Usuarios

El usuario administrador dispone de las funciones para ver los demás usuarios investigadores o colaboradores, así como borrarlos o cambiarles de categoría (rol). Los usuarios se muestran paginados de 7 en 7 (lo suficiente para llenar el espacio de la página sin necesidad de hacer *scroll*) y se pueden ordenar por nombre de usuario, nombre o apellido (figura 9.7).

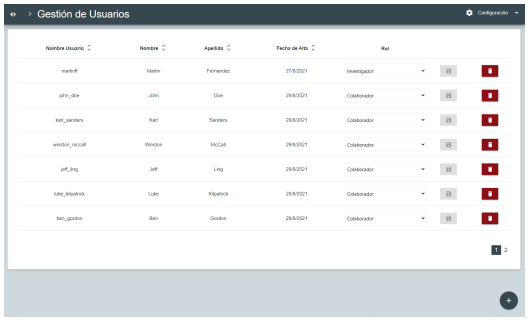


(a) Configuración de cuenta para Usuario Administrador

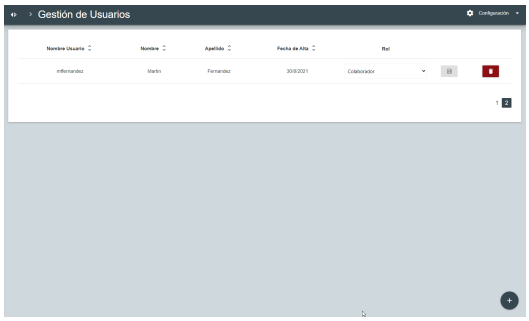


(b) Configuración de cuenta para Usuario NO Administrador

Figura 9.6: Configuración de cuenta



(a) Lista de Usuarios Página 1



(b) Lista de Usuarios Página 2

Figura 9.7: Lista de Usuarios

A parte de esto, un administrador también puede añadir usuarios nuevos a la aplicación con el rol que quiera especificar (Investigador o Colaborador), sin necesidad de introducir para el usuario una cuenta de E-Mail y eliminando así el paso de la verificación. Dispone para ello de un formulario similar al de registro (figura 9.8).

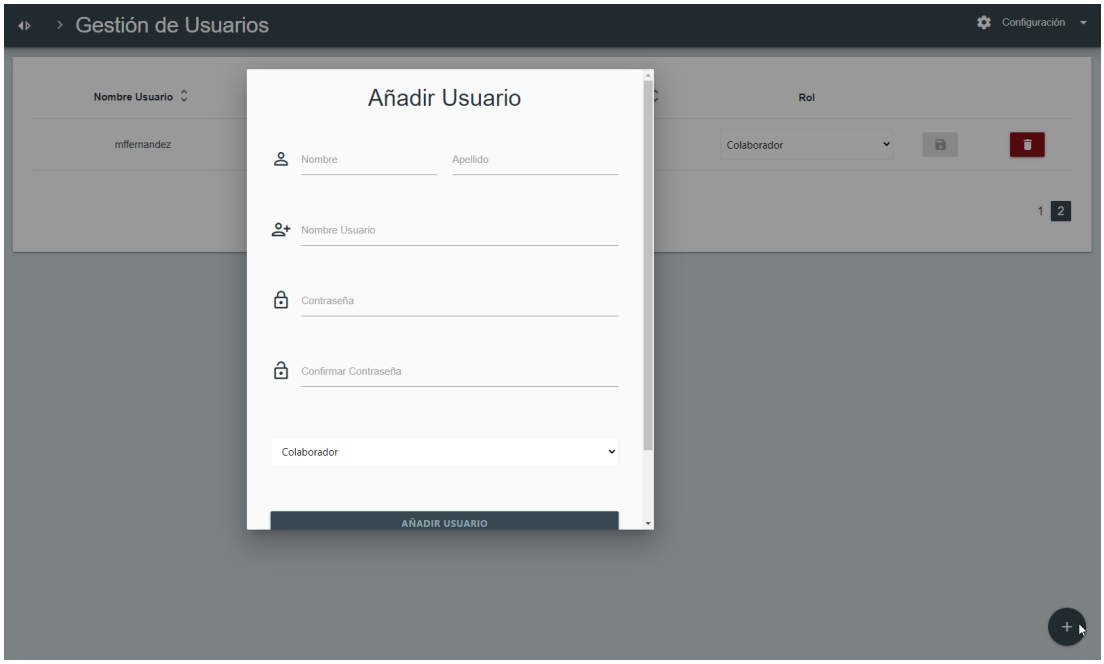


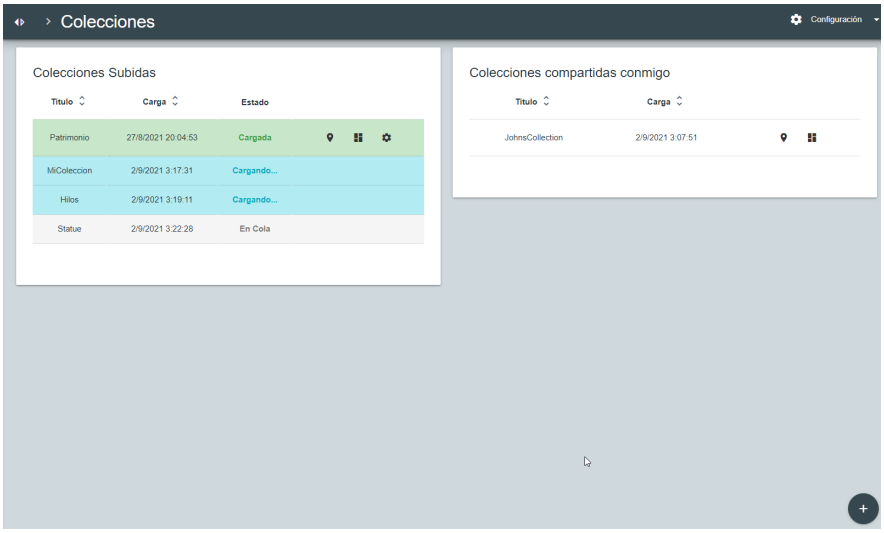
Figura 9.8: Formulario para añadir usuario nuevo desde cuenta de administrador

9.4 Manejo de Colecciones

Los usuarios investigadores y colaboradores pueden ver las listas de colecciones de referencia a las que tienen acceso para analizar. El usuario investigador tiene 2 listas, una con las colecciones propias subidas y otra con las que le han compartido; mientras que el colaborador como no puede subir colecciones sólo tiene una lista con las colecciones que le han compartido (figura 9.9).

El usuario investigador puede subir colecciones mediante un pequeño formulario que tiene disponible con un botón al fondo de la misma página (figura 9.10). La colección se añade a la lista con el estado “Cargando” si no había ninguna colección cargándose antes en el servidor, o “En Cola” en caso contrario. Pasará a “Cargando” cuando el servidor la empiece a procesar. Cuando esté lista, la aplicación avisa al usuario y pasa la colección a “Cargada”, permitiendo el análisis de ésta (figura 9.11).

Un usuario que es propietario de una colección y que esté cargada correctamente puede compartir esa colección con otros usuarios investigadores o colaboradores con un enlace (en forma de rueda de “Opciones”) junto a la colección deseada, lo que le redirige a un formulario para especificar estos usuarios o eliminarlos (figura 9.12).



(a) Listas de Colecciones disponibles para el Usuario Investigador

(b) Lista de Colecciones Compartidas disponibles para el Usuario Colaborador

Figura 9.9: Listas de Colecciones

9.5 Análisis de Colecciones

Tanto los usuarios investigadores como colaboradores pueden hacer el análisis de las colecciones a las que tienen acceso en sus listas siempre que ya hayan sido cargadas correctamente. Para ello disponen de dos enlaces junto a cada una de las colecciones que le permiten acceder al mapa o al *dashboard*.

La vista de mapa se muestra como un Mapamundi con una chincheta por cada una de las entidades extraídas de tipo localización (figura 9.13). Si se pincha en una de las chinchetas se pueden ver varios detalles relativos a la entidad, como el nombre, una dirección, y las referencias a dicha entidad en la colección, formadas por hilo, post y autor, asi como los términos relacionados del tesauro con dicha entidad en esa referencia, si los hubiere (figura 9.14).

La vista de *dashboard* se ve como un conjunto de estadísticas y gráficos sobre la colección. El primer gráfico de burbujas representa los datos de cada uno de los hilos de la colección, en la que se puede comprobar para cada uno el número de autores participando (eje de las X), el

Figura 9.10: Formulario para subir colección

Figura 9.11: Aviso de colección cargada

número de posts en el hilo (eje de las Y), número de entidades extraídas del hilo (tamaño de la burbuja) y el número de posts relevantes (color según la leyenda). Si se pasa el ratón por una de las burbujas se ve información detallada (figura 9.15).

Los otros dos gráficos son: un gráfico de sectores que muestra la cantidad y la proporción de cada tipo de entidad respecto a las referencias de las mismas (incluye menciones repetidas de las entidades); y otro gráfico de burbujas para ver el número de ocurrencias de términos del tesauro relacionados con entidades, similar al gráfico de los hilos pero sin ejes. El tamaño de la burbuja y el color escalan únicamente por el número de ocurrencias del término al que

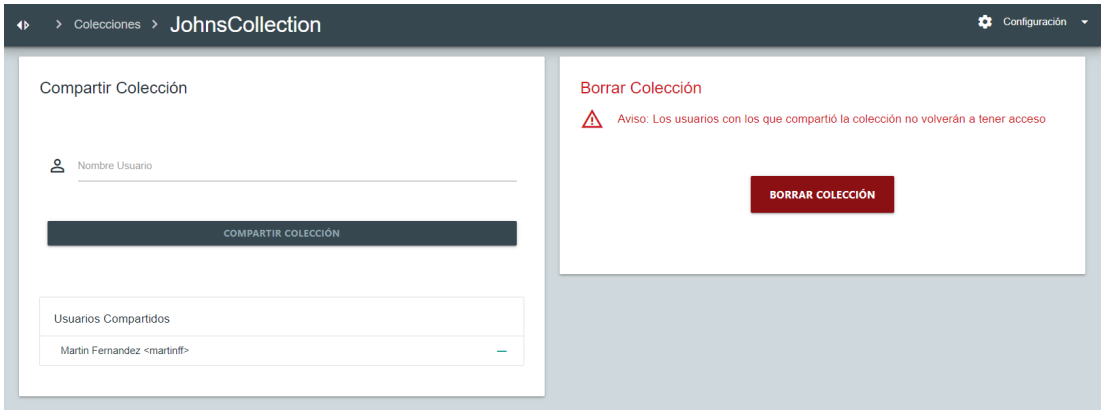


Figura 9.12: Página para compartir o dejar de compartir una colección

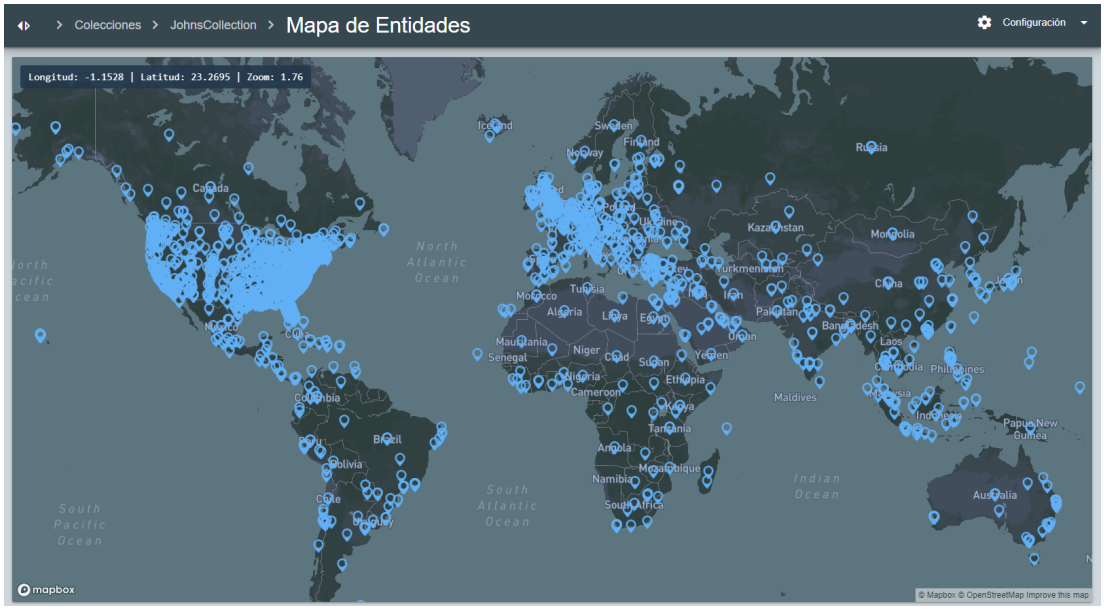


Figura 9.13: Vista global del Mapa de Entidades

representa la burbuja (figura 9.16).

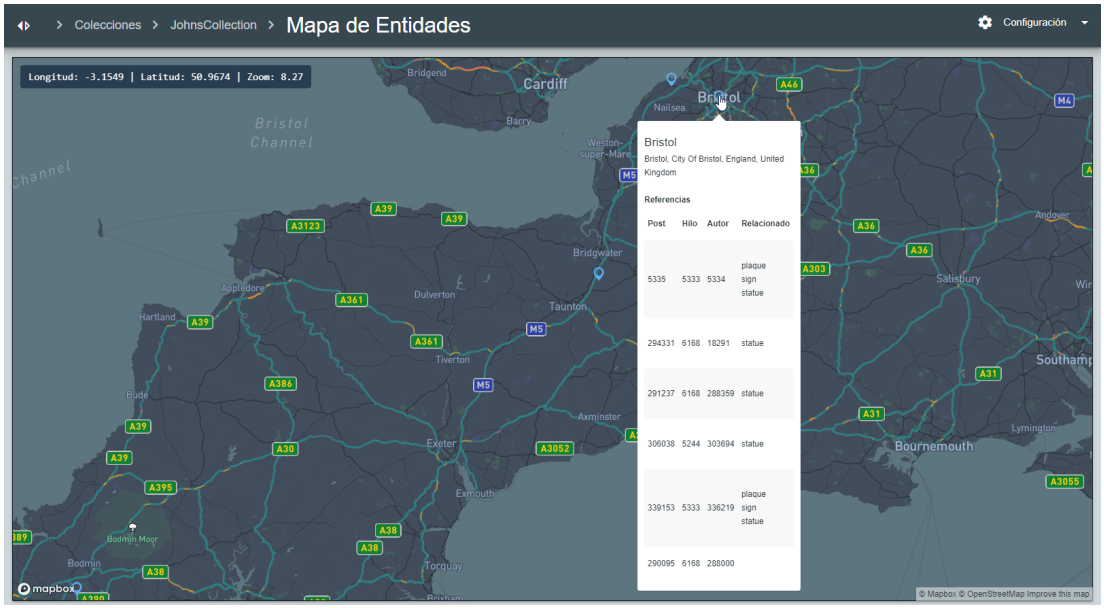


Figura 9.14: Vista de detalle de una de las entidades del mapa

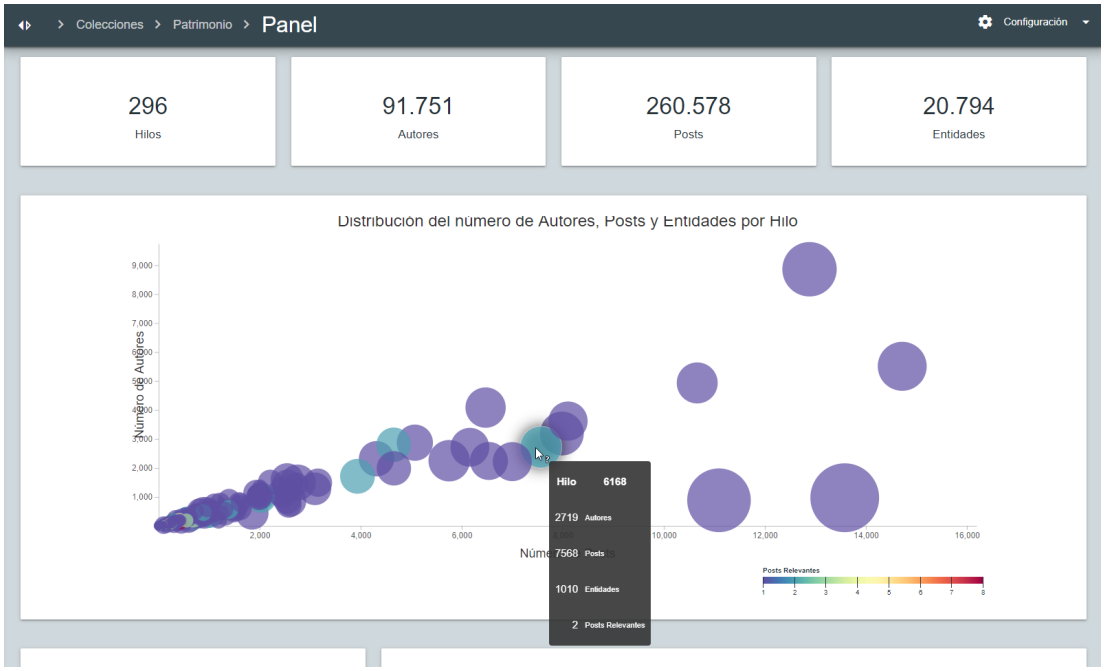


Figura 9.15: Vista del Dahsboard con el Gráfico de Burbujas de Hilos

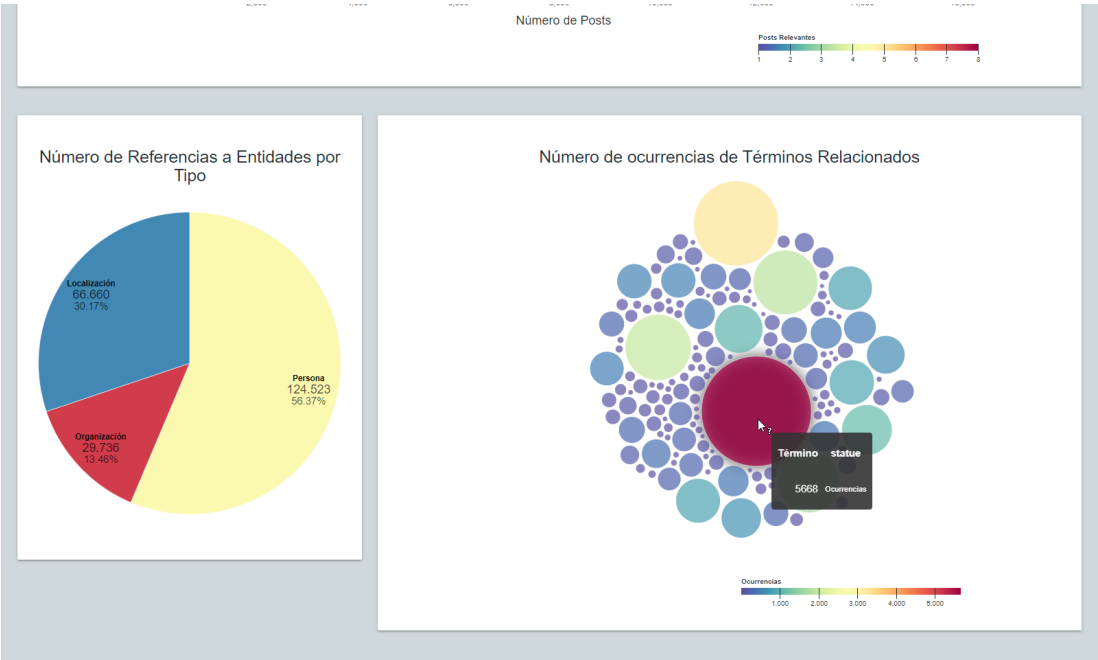


Figura 9.16: Vista del Dahsboard con el Gráfico de Tipos de Entidad y Ocurrencias de Términos

Capítulo 10

Conclusiones

En este último capítulo se hará una evaluación crítica sobre el desarrollo del proyecto y se detallarán líneas de mejora futuras que aportarían mayor valor al producto.

10.1 Desarrollo del proyecto

Una vez terminado el desarrollo del proyecto se ha cumplido con el objetivo de éste y se han implementado todas las historias de usuario. Hemos construido una aplicación Web que es capaz de procesar colecciones de referencia en un formato *machine-readable* (XML), que resulta poco amigable para el análisis por parte de investigadores de campos que no pertenecen a las Tecnologías de la Información (TI). También es capaz de presentarlas de una manera que permite facilitar la visualización y el análisis de las mismas. En concreto:

- Permite la identificación, creación y gestión de usuarios en tres roles para tener acceso a las distintas funcionalidades de la aplicación.
- Permite a los usuarios administradores una forma fácil de poder gestionar usuarios y añadir nuevos usuarios por ellos mismos.
- Proporciona una forma sencilla y cómoda de subir, procesar y almacenar información de colecciones de referencia.
- Permite contemplar de manera intuitiva las colecciones de referencia a las que un usuario puede tener acceso para realizar el análisis, así como compartir y dejar de compartir una colección propia con otros usuarios de una manera simple.
- Proporciona una vista de mapa para poder visualizar todas las entidades de tipo localización de una colección.
- Permite visualizar información sobre la colección de referencia en diferentes estadísticas y gráficos.

10.2 Lecciones aprendidas

Durante el desarrollo del proyecto se profundizó en el dominio de herramientas anteriormente utilizadas (Java, Spring, Hibernate, etc.) y se alcanzó un importante conocimiento en algunas herramientas nuevas (JavaScript, React, Redux, Gradle, etc.). Herramientas como Java, Spring y React son tecnologías muy demandadas en el mercado laboral, por lo que la realización de éste proyecto ha sido una experiencia muy positiva en cuanto a conocimientos aprendidos se refiere. React es una tecnología que cada vez tiene más impacto en aplicaciones Web empresariales, junto con otras tecnologías del lado de servidor como Java y Spring Boot.

También hemos obtenido una formación en las tecnologías de Procesamiento de Lenguaje Natural (NLP). Este proyecto ha sido una primera toma de contacto con este campo, puesto que durante el grado no hemos realizado trabajo realizado con él (al menos en el itinerario de Ingeniería de Software), y consideramos que también nos ha aportado mucho este aspecto y su manera de abordarlo de una manera práctica. Los algoritmos que hemos utilizado (CoreNLP y OpenNLP), gracias a estar disponibles para la plataforma Java, han sido muy cómodos de integrar con las demás tecnologías de nuestro proyecto.

En cuanto a las metodologías utilizadas, Scrum, TDD y *Git-Flow*, son prácticas muy utilizadas en proyectos modernos ya que permiten un desarrollo mucho más ágil y centrado en el producto, alejándose de las viejas metodologías centradas en procesos y documentación. El conocimiento y profundidad en estas metodologías también es muy positivo.

También hay aspectos que deberíamos mejorar en este proyecto, y que nos gustaría incluir en líneas futuras de trabajo. Entre ellos, la inclusión de más buenas prácticas para Ingeniería de Software como *DevOps*: eg. herramientas de integración continua como *Jenkins*, para llevar un control automático del software que vamos desarrollando, enfocado a un entorno de trabajo real. También sería necesaria una evaluación del rendimiento general de la aplicación, utilizando herramientas de monitorización como JMeter, así como mejorar el rendimiento de algunas consultas a base de datos y de algunos elementos de la Web como el mapa.

10.3 Líneas de trabajo futuras

Aunque se haya alcanzado el objetivo del proyecto, creemos que hay una serie de funcionalidades y aspectos que se podrían incluir en la aplicación, a trabajar en un futuro:

- Mejorar la gestión de los usuarios por parte del administrador: filtros en las listas de usuarios, por nombre o apellido. Actualmente permite la ordenación y la paginación de éstos y es capaz de proporcionar una buena experiencia de usuario, pero creemos que podrían añadirse estos aspectos.

- Mejorar el rendimiento de la carga de la colección. La carga de la colección es una operación lenta y costosa. Creemos que se puede investigar en busca de herramientas o utilidades que podamos aplicar a este aspecto para intentar mejorar en rendimiento.
- Mejorar la eficiencia de algunas de las consultas que lanzamos contra bases de datos.
- Mejorar el rendimiento de la interfaz Web del mapa para intentar ganar en escalabilidad. La colección tiene muchas entidades de tipo localización y esto hace que interactuar con el mapa a veces sea algo lento. Creemos que en un futuro, con colecciones más grandes, podría dar más problemas.
- Incluir nuevas visualizaciones de datos al *dashboard*, como por ejemplo, un grafo que permita visualizar las relaciones de los autores a través de los hilos de la colección de referencia analizada.
- Incluir nuevas herramientas para ayudar y automatizar el desarrollo, por ejemplo, de integración continua como Jenkins, o herramientas de monitorización como JMeter.
- Incluir soporte para más idiomas en la internacionalización: actuamente sólo disponemos de inglés, español y gallego.
- Realizar estudios para medir el grado de eficiencia, productividad y satisfacción del usuario final, enfocados a perfiles investigadores y no tecnológicos [60].

Apéndices

Lista de Acrónimos

ACID Atomicity, Consistency, Isolation and Durability; Cuando una base de datos soporta transacciones. 18, 19

AJAX Asynchronous JavaScript and XML. 14

AOP Programación Orientada a Aspectos. 7, 8

API Interfaz de Programación de Aplicaciones. 3, 7, 17, 50, 51, 53, 62, 66

BSON Binary JSON (JavaScript Object Notation). 19

CSS Cascade Style Sheets. 15, 16

DAO Data-Access Object. 51, 53

DOM Document Object Model. 15, 18, 32, 62, 65

DTO Data-Transfer Object. 11, 51

HTML Hyper-Text Markup Language. 14, 15, 32, 65

IDE Entornos de Desarrollo Integrado. 11, 19, 20, 23

IoC Inversion of Control. 7, 8

JDBC Java Database Conectivity. 7

JPA Java Persistence API. 7, 10

JSON JavaScript Object Notation. 9, 20, 54, 57, 60

JWT JSON Web Tokens. v, 57, 58

MIME Multipurpose Internet Mail Extensions. 61

MVC Modelo Vista Controlador. 7

NER Reconocimiento de Entidades Nombradas. 3, 11, 12

NLP Procesamiento de Lenguaje Natural. 3, 11, 12, 33, 53, 60, 78

NPM Node Package Manager. 22

OGM Mapeado Object/Grid. 10

ORM Mapeado Objeto-Relacional. 10

SASS Syntactically Awesome Style Sheets. 16, 54

SCM Gestión de Código Fuente. 20, 30

SGBD Sistema de Gestión de Bases de Datos. 18, 19

SPA Single-Page Applications o Aplicaciones de una Sóla Página. 14, 15

SQL Structured Query Language. 18

SSE Server-Sent Events. vi, 61

TDD Test Driven Development. 25, 29, 30, 32, 65, 78

TI Tecnologías de la Información. 1, 77

UI User Interface. 47

UML Unified Modeling Language. 24

UX User Experience. 47

VCS Sistema de Control de Versiones. 20, 21, 25

XML Extensible Markup Language. v, 1, 3, 21, 22, 33, 34, 40, 60, 77

Glosario

DevOps Conjunto de prácticas que agrupan el desarrollo de software (Dev) y las operaciones de TI (Ops). 30, 78

Geocodificación Proceso de extraer las coordenadas de un determinado lugar o localización a partir del nombre de éste. 62

InfoVis Visualización de Información o Information Visualization, representación con gráficos y otras técnicas de visualización en una interfaz de usuario de una gran cantidad de datos. 1

Open-Source Herramientas o tecnologías software de código abierto. 6, 21

responsive Diseño de interfaces adaptable, diseño de una interfaz web que se "adapta" y es accesible apara multitud de dispositivos. 15

REST Representations State Transfer, estilo de comunicación de dos componentes software utilizando el protocolo HTTP. 7, 17, 51, 53, 54, 66

Bibliografía

[1] J. Pybus, "Social networks and cultural workers: towards an archive for the prosumer," *Journal of Cultural Economy*, vol. 6, pp. 137–152, 2013.

[2] A. A. Acker and J. R. Brubaker, "Death, Memorialization, and Social Media: A Platform Perspective for Personal Archives," *Archivaria*, vol. 77, pp. 1–23, may 2014. [En línea]. Disponible en: https://archivaria.ca/index.php/archivaria/article/view/13469

[3] A. Maria Vallès and J. Soler Jiménez, "El tractament arxivístic de Twitter," 2016. [En línea]. Disponible en: https://hdl.handle.net/2072/268022

[4] V. Ruiz Gómez and A. Maria Vallès, "#Cuéntalo: the path between archival activism and the social archive(s)," *Archives and Manuscripts*, vol. 48, no. 3, pp. 271–290, 2020. [En línea]. Disponible en: https://doi.org/10.1080/01576895.2020.1802306

[5] D. Otero, P. Martin-Rodilla, and J. Parapar, *Building Cultural Heritage Reference Collections from Social Media through Pooling Strategies: The case of 2020's tensions over race and heritage*, M. Hedges, E. Goudarouli, and R. Marciano, Eds. ACM Journal on Computing and Cultural Heritage, 2021, paper aceptado in press en Special Issue on "Computational Archival Science". [En línea]. Disponible en: http://hdl.handle.net/2183/28052

[6] J.-D. Fekete, J. van Wijk, J. Stasko, and C. North, *The Value of Information Visualization*, jul 2008, vol. 4950, pp. 1–18. [En línea]. Disponible en: http://doi.org/10.1007/978-3-540-70956-5_1

[7] "George Floyd: What happened in the final moments of his life," *BBC News*, jul 2020. [En línea]. Disponible en: https://www.bbc.com/news/world-us-canada-52861726

[8] C. A. Atuire, "Black Lives Matter and the Removal of Racist Statues. Perspectives of an African," 2020. [En línea]. Disponible en: http://doi.org/10.11588/xxi.2020.2.76234

[9] N. Pfosi, "Protesters tear down Christopher Columbus statue in Saint Paul, Minnesota," *Reuters*, jun 2020. [En línea]. Disponible en: https://www.reuters.com/article/us-minneapolis-police-saint-paul-statue-idUKKBN23I04X

[10] "Tensions rise over race and heritage as more statues are attacked," *The Guardian*, jun 2020. [En línea]. Disponible en: https://www.theguardian.com/us-news/2020/jun/11/fears-of-violence-stop-london-racism-protest-as-statue-attacks-continue

[11] T. Munir, "Influence of Social Media on the Black Lives Matter Movement," 2020. [En línea]. Disponible en: http://doi.org/10.13140/RG.2.2.29213.87527

[12] P. Martin-Rodilla and D. Otero, *Estrategias de recuperación de información mediante pooling para la construcción de colecciones de referencia desde redes sociales: caso de estudio durante las tensiones raciales de 2020*, Santiago de Compostela, España, Octubre 2021, paper aceptado en V congreso de la Sociedad Internacional de Humanidades Digitales Hispánicas (HDH 2021).

[13] F. Windhager, P. Federico, E. Mayr, G. Schreder, and M. Smuc, "A Review of Information Visualization Approaches and Interfaces to Digital Cultural Heritage Collections," nov 2016. [En línea]. Disponible en: http://ceur-ws.org/Vol-1734/fmt-proceedings-2016-paper9.pdf

[14] P. Nadkarni, L. Ohno-Machado, and W. Chapman, "Natural Language Processing: An introduction," *Journal of the American Medical Informatics Association : JAMIA*, vol. 18, pp. 544–551, sep 2011. [En línea]. Disponible en: http://doi.org/10.1136/amiajnl-2011-000464

[15] R. Jain, A. Sharma, G. S. Mishra, P. Nad, and S. Chakraborty, "Named Entity Recognition in English Text," *Journal of Physics: Conference Series*, vol. 1712, p. 012013, dec 2020. [En línea]. Disponible en: https://doi.org/10.1088/1742-6596/1712/1/012013

[16] "WordNet: A Lexical Database for English." [En línea]. Disponible en: https://wordnet.princeton.edu/

[17] Oracle, "Documentación Java EE." [En línea]. Disponible en: https://www.oracle.com/es/java/technologies/java-ee-glance.html

[18] "Spring Overview." [En línea]. Disponible en: https://docs.spring.io/spring-framework/docs/current/reference/html/overview.html

[19] "Spring Boot Documentation." [En línea]. Disponible en: https://spring.io/projects/spring-boot

[20] "Hibernate ORM Documentation." [En línea]. Disponible en: https://hibernate.org/orm/

[21] "Project Lombok." [En línea]. Disponible en: https://projectlombok.org/

[22] "Stanford CoreNLP Site." [En línea]. Disponible en: https://stanfordnlp.github.io/CoreNLP/

[23] "Apache OpenNLP Manual." [En línea]. Disponible en: https://opennlp.apache.org/docs/1.9.3/manual/opennlp.html

[24] "JUnit 5 User Guide." [En línea]. Disponible en: https://junit.org/junit5/docs/current/user-guide/

[25] "Mockito Framework Site." [En línea]. Disponible en: https://site.mockito.org/

[26] "MDN Web Docs - JavaScript." [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[27] "ECMAScript 2015 Language Specification." [En línea]. Disponible en: https://262.ecma-international.org/6.0/6

[28] "React Documentation." [En línea]. Disponible en: https://reactjs.org/docs/

[29] "Redux User Guide." [En línea]. Disponible en: https://redux.js.org/usage/index

[30] "Materialize Documentation." [En línea]. Disponible en: https://materializecss.com/

[31] "Material Design." [En línea]. Disponible en: https://material.io/design

[32] "SASS documentation." [En línea]. Disponible en: https://sass-lang.com/documentation

[33] "MapBox Documentation." [En línea]. Disponible en: https://docs.mapbox.com/

[34] "D3 Site." [En línea]. Disponible en: https://d3js.org/

[35] "Jest Site." [En línea]. Disponible en: https://jestjs.io/

[36] "Testing Library." [En línea]. Disponible en: https://testing-library.com/docs/

[37] "MySQL Documentation." [En línea]. Disponible en: https://dev.mysql.com/doc/refman/5.7/en/

[38] "The MongoDB 4.2 Manual." [En línea]. Disponible en: https://docs.mongodb.com/v4.2/

[39] "IntellijIDEA Features." [En línea]. Disponible en: https://www.jetbrains.com/idea/features/

[40] "Visual Studio Code Docs." [En línea]. Disponible en: https://code.visualstudio.com/docs

[41] "About Git." [En línea]. Disponible en: https://git-scm.com/about

[42] "GitHub Features." [En línea]. Disponible en: https://github.com/features

[43] "Gradle Documentation." [En línea]. Disponible en: https://docs.gradle.org/5.6.4/

[44] "About NPM." [En línea]. Disponible en: https://docs.npmjs.com/about-npm

[45] "Webpack Documentation." [En línea]. Disponible en: https://webpack.js.org/concepts/

[46] "SonarQube." [En línea]. Disponible en: https://docs.sonarqube.org/latest/

[47] "Draw.io." [En línea]. Disponible en: https://www.diagrams.net/

[48] "Pencil Project." [En línea]. Disponible en: https://pencil.evolus.vn/

[49] "Taiga." [En línea]. Disponible en: https://www.taiga.io

[50] K. Schwaber and J. Sutherland, *The Scrum Guide; The Definitive Guide to Scrum: The Rules of the Game.* scrum.org, nov 2020. [En línea]. Disponible en: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

[51] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," 2001. [En línea]. Disponible en: http://www.agilemanifesto.org/

[52] K. Beck, *Test Driven Development. By Example.* Addison-Wesley Longman, Amsterdam, 2002.

[53] ——, *Extreme Programming Explained: Embrace Change.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[54] F. Santacroce, *Git Essentials (Second Edition)*, nov 2017.

[55] A. Sajedi, M. Mahdavi, A. Pourshirmohammadi, and M. M. Nejad, "Fundamental Usability Guidelines for User Interface Design," jun 2008, pp. 106 – 113. [En línea]. Disponible en: https://doi.org/10.1109/ICCSA.2008.45

[56] "i18next." [En línea]. Disponible en: https://www.i18next.com/

[57] "Glassdoor: Guía Salarial." [En línea]. Disponible en: https://www.glassdoor.es/Sueldos/index.htm

[58] R. Cooper and R. S. Kaplan, *Design of Cost Management Systems.* USA: Ed. Prentice-Hall International, 1989.

[59] "JWT: Introduction." [En línea]. Disponible en: https://jwt.io/introduction

[60] P. Martin-Rodilla, I. Panach, C. Gonzalez-Perez, and O. Pastor, "Assessing data analysis performance in research contexts: An experiment on accuracy, efficiency, productivity and researchers' satisfaction," *Data & Knowledge Engineering*, vol. 116, pp. 177–204, jun 2018. [En línea]. Disponible en: https://doi.org/10.1016/j.datak.2018.06.003