Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Desarrollo de buscador de documentos técnicos

Autor
José Antonio Font García

Director
Ramón Ruiz Carmena

Madrid
Julio 2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título DESARROLLO DE BUSCADOR DE DOCUMENTOS TÉCNICOS en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019/2020 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: José Antonio Font García Fecha: 11/07/2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Ramón Ruiz Carmena Fecha: 11/07/2020

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Desarrollo de buscador de documentos técnicos

Autor
José Antonio Font García

Director
Ramón Ruiz Carmena

Madrid
Julio 2020

DESARROLLO DE BUSCADOR DE DOCUMENTOS TÉCNICOS

Autor: Font García, José Antonio.
Director: Ruiz Carmena, Ramón.
Entidad colaboradora: Ennomotive.

RESUMEN DEL PROYECTO

Introducción

El proyecto se lleva a cabo dentro de la empresa *Ennomotive*, la cual dispone de una plataforma de innovación que conecta proyectos con usuarios dispuestos a resolverlos. En este contexto, se pretende llevar a cabo el desarrollo de un buscador de documentos técnicos que permita a los usuarios acceder a documentación de utilidad al afrontar los desafíos de la plataforma. Para conseguir esto, se necesita previamente un sistema que permita recoger de forma estructurada las valoraciones de los usuarios.

Para llevar a cabo el proyecto de forma eficiente, conviene realizar una panorámica de las diversas fuentes que puedan servir como referencia o guía. Por un lado, existen contribuciones recientes al campo de la clasificación de textos empleando redes neuronales convolucionales [1]. Además,revisar los últimos avances en metodologías *serverless* [2] y de microservicios [3] permitirá implementar los modelos de desarrollo más actuales.

De acuerdo a lo explicado, el objetivo fundamental del proyecto es el de desplegar una plataforma que permita recoger el *feedback* de los usarios y lo almacene de forma estructurada. Para lograr esto se deberá disponer de una plataforma que permita buscar documentos a través de diferentes fuentes. Por último, será conveniente llevar a cabo las pruebas necesarias para prevenir accesos no autorizados y garantizar la integridad y privacidad de los datos.

Metodología

Dadas las condiciones materiales en las que se ubica el proyecto, se ha optado por implementar una metodología ágil de desarrollo, lo cual ha permitido desarrollar prototipos con funcionalidades adicionales de manera prácticamente continua. Además, dicha metodología ha demostrado ser la que mejor se adapta a las condiciones del entorno disponible ya que es de integración sencilla y permite una adaptación rápida a los cambios.

Dicha metodología se basa en *sprints* que deben culminar en la presentación de un prototipo con funcionalidades añadidas sobre el anterior. Además, dicha presentación deberá ser acompañada de la elección de nuevos objetivos para el próximo ciclo.

Resultados

En lo tocante a los resultados, se ha conseguido recopilar un total superior a los 1200 documentos. En la primera prueba han participado 3 usuarios, los cuales han emitido un total cercano a las 40 valoraciones y reportes. El total de los datos ocupa 28MB, lo que demuestra un uso eficiente de los recursos. Además, el tiempo de respuesta a una búsqueda aleatoria entra dentro de los parámetros aceptables.

No obstante, se mantiene un tiempo de carga elevado a la hora de acceder a los documentos. Además, las búsquedas devuelven resultados de escasa utilidad y que, en ocasiones, fallan. Por otro lado, la aplicación web demuestra carencias en lo tocante a diseño y experiencia de usuario.

Conclusiones

Incluso con ciertas carencias, el proyecto cumple con los objetivos esperados y sienta las bases para el desarrollo de proyectos futuros. Pese a que no se han podido completar todas las pruebas deseadas, el sistema desarrollado cumple con la tarea de permitir la búsqueda de documentos y la recopilación estructurada de valoraciones. Además, ha quedado patente la necesidad de mejorar los mecanismos de búsqueda implementados, así como la importancia de implementar mecanismos para lidiar con el problema de los documentos fallidos. Adicionalmente, es recomendable dedicar recursos a la mejora de la experiencia de usuario en la página web.

Sin embargo, lo más importante es tomar el relevo del proyecto y desarrollar un sistema que utilice los datos recopilados para construir sistemas inteligentes para la clasificación de documentos. Esto sería una herramienta de gran utilidad para los usuarios de la plataforma no sólo a través del buscador, sino a través de funcionalidades para procesar las soluciones subidas por estos a la plataforma.

Referencias

[1] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification,ïn Proceedings of the 2016 conference

of the North American chapter of the association for computational linguistics: human language technologies, pp. 1480-1489, 2016.

[2] Diego Zanon. Building Serverless Web Applications. Packt Publishing Ltd, 2017.

[3] Johannes Thones. "Microservices". En: IEEE software 32.1 (2015), pags. 116-116.

DEVELOPMENT OF A SEARCHER FOR TECHNICAL DOCUMENTS

Introduction

The project is being developed inside of *Ennomotive*, a company that owns an innovation platform in which different projects are being connected to users with the means to solve them. In this context, the goal is to carry out the development of a searcher for technical documents that allows the user to access useful documentation when facing the challenges hosted in the website. To achieve this, a system to gather the feedback of the useres is needed.

In order to carry out the project in an efficient manner, it is convinient to give a panoramic view of various sources that could serve as a reference or guidance. On the one hand, there are recent contributions to the field of text classification in which convolutional neural networks [1] are being used. Furthermore, revisiting the latest developments on serverless methodologies [2] and microservices [3] will help implement the latest software development practices.

In regards to what has been covered in the text before, the main goal of the project is that of deploying a platform able to gather user feedback and store it in an orderly manner. To achieve this, a system to search for documents accross different sources will have to be made available. In addition, it will be convenient to carry out different test in order to prevent unauthorized access and to guarantee the integrity and privacy of the information stored.

Methodology

Given the material conditions in which the project is carried out, an agile methodology has been chosen. This, in turn, has allowed the team to develop new prototypes with new features one after the other. Moreover, this methodology has proven to be the one that best adapts to the condition of the available environment since it is easy to implement and allows for a quick adaptation to change.

The agile methodology is based on sprints which must result in a prototype with features built ontop of the previous ones. In addition, every prototype must be supplemented with the election of those objectives that need to be fulfilled by the next iteration.

Results

In regards to the results, over 1200 documents have been gathered in the system. In the first cycle, 3 users have partaken, providing over 40 rates and reports. The sum of the data sspans across 28 MB, which shows an efficient use of resources. In addition, the time the application takes to respond to a random query is within the acceptable values.

However, the time to load a document is still high. In addition, some searches return results with little connection to the query and, sometimes, do not work. Moreover, the app could improve its design and user experience.

Findings

In spite of a couple of shortcomings, the project delivers the expected results and helps lay the foundation for the development of other projects in the future. Even though some tests could not be completed, the system achieves the goal of allowing for documents to be searched and rates to be gathered. In addition, the necessity of improving the methods used for searching has been proven, as well as the importance of implementing mechanisms to deal with faulty documents. Moreover, it is advisable to improve the design of the web app and its user experience.

However, the most important task is to make use of the findings of the project in order to train systems capable of using them for document classification. This would prove to be a useful tool for the users of the platform not only through the searcher, but throught systems able to process the solutions uploaded by them.

References

[1] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification,ïn Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, pp. 1480-1489, 2016.

[2] Diego Zanon. Building Serverless Web Applications. Packt Publishing Ltd, 2017.

[3] Johannes Thones. "Microservices". En: IEEE software 32.1 (2015), pags. 116-116.

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

El desarrollo de las tecnologías de comunicación durante las últimas décadas ha permitido la conexión entre personas en contextos muy diferentes. Este nuevo panorama, sumado a los crecientes medios para el procesado de datos, ha contribuido a la revisión de los modelos tradicionales de desarrollo tecnológico, así como a la aparición de nuevos paradigmas. La génesis y desarrollo de movimientos como el del *software libre* y el *software abierto* durante las últimas décadas del siglo pasado ha traído consigo tecnologías como *GNU* y *Linux*, las cuales ocupan un lugar vital en la industria del *software* [1]. Esta evolución en las tecnologías y en las metodologías de desarrollo ha propiciado la aparición de una cultura participativa y una inteligencia colectiva que puede ser aprovechada a través del *crowdsourcing* [2].

Ennomotive surge con el objetivo de aprovechar este fenómeno para generar innovación industrial, así como soluciones viables y tangibles con un valor real para las empresas. En la plataforma, existe una comunidad global de ingenieros formada por individuos con interés en encontrar proyectos para resolver, obteniendo diversos beneficios. Por otro lado, diferentes empresas o individuos tienen la opción de negociar proyectos para ofertar en la plataforma. Así, se pone en contacto a empresas con la comunidad, lo que da lugar a una relación entre proyectos particulares y la inteligencia colectiva. Esto tiene como producto un proceso en el cual, a través de una serie de fases, se da respuesta a las especificaciones del proyecto con unsa solución lista para ser implementada[3].

Con el fin de agilizar este proceso, se ha optado por desarrollar una herramienta que permita a los usuarios de la plataforma recopilar documentación de carácter técnico para resolver los problemas planteados. Para ello, se ha comenzado el desarrollo de una aplicación *web* que permita buscar documentos utilizando *queries*, así como darles una clasificación en función de su utilidad. El presente documento describe detalladamente el proceso de desarrollo de un buscador que permita recopilar información para ser utilizada en el entrenamiento de los algo-

ritmos de clasificación, así como servir de base sobre la que construir el resto de funcionalidades deseadas.

En esta primera sección, se explica el contexto del proyecto así como sus elementos fundamentales. A continuación, se encuentra una explicación de las diferentes tecnologías empleadas para el desarrollo de este. Posteriormente, en el estado de la cuestión, se hace una visión panorámica de las soluciones a proyectos similares, así como los avances en ingeniería del *software* que han guiado el desarrollo del proyecto. Adicionalmente, se exponen los detalles que han determinado el trabajo: motivación, objetivos, métodos, etc. Para continuar, se realiza una descripción pormenorizada de las características técnicas de la solución desarrollada, así como un análisis de los resultados obtenidos. Por último, se encuentra una sección con las conclusiones extraídas y aquellas necesidades que deban ser cubiertas con trabajos futuros.

Capítulo 2

Descripción de las Tecnologías

Para describir detalladamente el proyecto, es preciso enumerar y desarrollar las tecnologías seleccionadas para su elaboración y despliegue. A raíz la naturaleza puramente digital del proyecto, se pueden derivar tres categorías para clasificar las tecnologías empleadas. Por un lado, han sido necesarias tecnologías para solventar las necesidades del proceso de desarrollo: control de versiones, automatización de tareas, etc. Por otro lado, existen tecnologías que han sido utilizadas para cubrir las necesidades técnicas: servidores, bases de datos, librerías, etc. Por último, son necesarias tecnologías para dar respuesta a las necesidades estructurales de la aplicación: *hosting*, balance de carga, etc.

2.1. Tecnologías para el proceso de desarrollo

A lo largo de la siguiente sección se presentan y explican las diferentes tecnologías que han servido como herramienta para la actividad de desarrollo de la aplicación. Se trata, por lo tanto, de tecnologías cuyo propósito consiste en agilizar el desarrollo automatizando diferentes tareas, centralizando la información o asegurando la integridad del código de la aplicación.

2.1.1. Gitlab

Gitlab es un gestor de repositorios *Git*. Dada su flexibilidad, cubre numerosas funciones de diferente importancia. En primer lugar, cumple los requisitos característicos de un sistema control de versiones: distribución del código, asegurar la integridad de los archivos e implementar las herramientas para el desarrollo ramificado [4]. Además, permite centralizar las tareas de planificación y documentación a través de *issues*, tareas individuales que son publicadas en tableros *to do/doing*, y *wikis*. Por último, permite definir *pipelines* para automatizar tareas tras cada

actualización: verificar el código, ejecutar pruebas, distribuirlo automáticamente, etc [5].

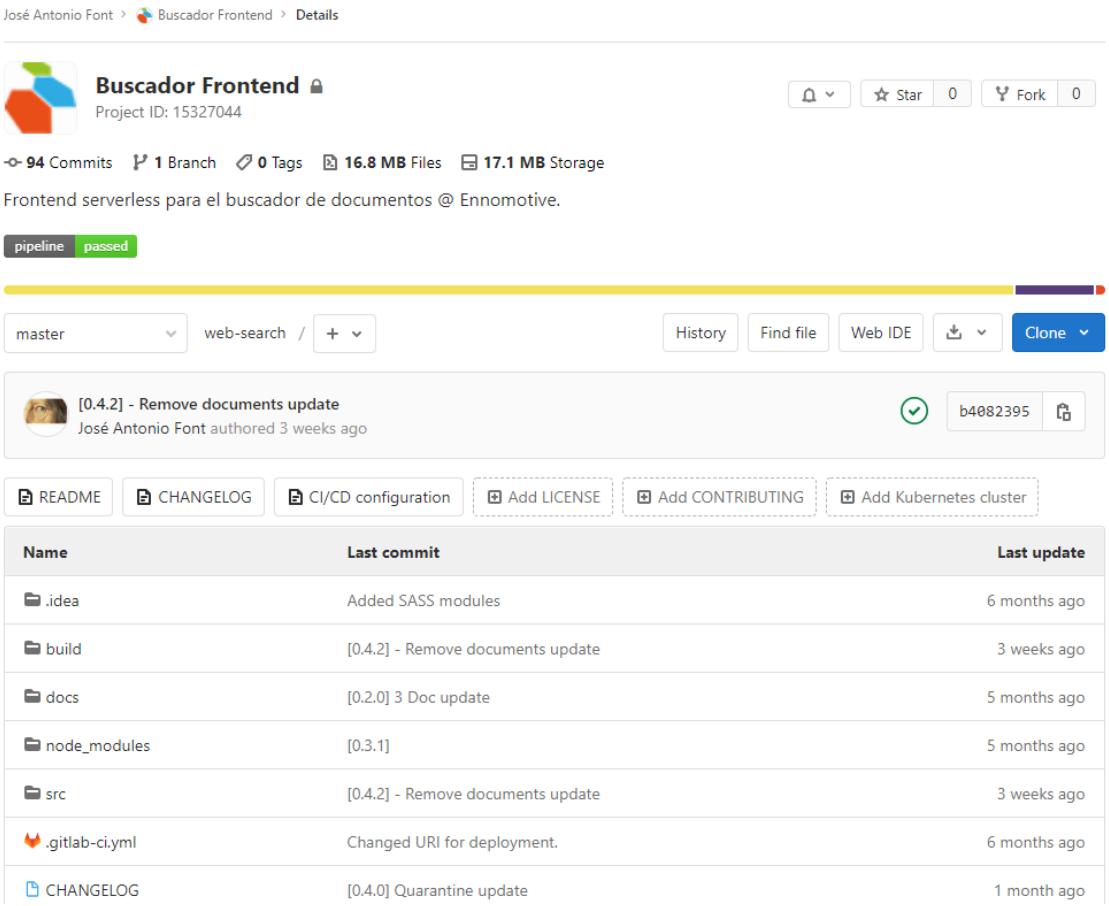

Figura 2.1: Captura de pantalla de un repositorio del proyecto.

Como cualquier tecnología basada en *Git*, funciona a través de comandos que pueden ser ejecutados en máquinas remotas para interactuar de forma segura con el código del programa. Gracias a esto, se implanta una dinámica de trabajo que permite controlar los cambios introducidos en el código de forma segura. Por lo tanto, a lo largo del transcurso del proyecto, dichos comandos han sido extensamente utilizados[4].

Adicionalmente, la capacidad para definir y ejecutar *pipelines* ha sido de vital importancia. A través de un fichero en el repositorio llamado *.gitlab-ci.yml*, se pueden definir varias etapas consecutivas formadas por trabajos paralelos. Dentro de cada trabajo se incluyen elementos de control, variables y el texto del *script* que vaya a ejecutarse. Dichos *scripts* se ejecutarán en el *runner* que se haya configurado con el proyecto [6]. En el *listing* 2.1 se encuentra un ejemplo de *pipeline* utilizada

en el proyecto.

Listing 2.1: Ejemplo de pipeline.

```
image: chybie/node-aws-cli


variables:
        URI: "http://URL-SERVIDOR"

Build Master:
        variables:
                DEV_BUILD: "No"
        stage: build
        script:
                - echo "Build Master"
                - npm install gulp-cli -g
                - npm rebuild node-sass
                - gulp build
                - aws s3 cp ./build s3://URL-SERVIDOR/ --
                    recursive --acl public-read
        only:
                - master



Build Staging:
        stage: build
        script:
                - echo "Build staging"
                - npm install gulp-cli -g
                - npm rebuild node-sass
                - gulp build
        only:
                - staging
```

2.1.2. Gulp.js

Gulp es un sistema modular construido con *Node.js* que, mediante un fichero de instrucciones *JavaScript*, permite automatizar, organizar y ejecutar tareas relacionadas con el desarrollo de aplicaciones web [7]. En el proyecto, es utilizado en la parte más expuesta al usuario en la página web y por lo tanto implementa tareas

de ofuscación de ficheros de código, compresión de imágenes, etc. Además, permite generar distintas versiones de los archivos finales para ser utilizados en pruebas locales o en su despliegue final, eliminando los mensajes de *debug* o sustituyendo las direcciones del servidor y base de datos local por las finales. En el *listing* 2.2 se muestra el archivo *gulpfile.js* empleado en el proyecto.

Listing 2.2: Archivo de Gulp.

```
const devBuild = process.env.DEV_BUILD ? process.env.
    DEV_BUILD : true;
const uri = process.env.URI ? process.env.URI : 'http://
    localhost:3000/';

console.log(`DEV_BUILD is ${devBuild}`);
console.log(`URI is ${uri}`);

const gulp = require('gulp');
const newer = require('gulp-newer');
const imagemin = require('gulp-imagemin');
const noop = require('gulp-noop');
const htmlclean = require('gulp-htmlclean');
const concat = require('gulp-concat');
const deporder = require('gulp-deporder');
const terser = require('gulp-terser');
const stripdebug = devBuild === "No" ? null : require('gulp
    -strip-debug');
const sourcemaps = devBuild === "No" ? require('gulp-
    sourcemaps') : null;
const preprocess = require("gulp-preprocess");
const sass = require('gulp-sass');

src = 'src/';
build = 'build/';



function images() {
const out = build + 'images/';
return gulp.src(src + 'images/**/*')
.pipe(newer(out))
.pipe(imagemin({ optimizationLevel: 5}))
.pipe(gulp.dest(out));
```

```
}

function html() {
const out = build;
return gulp.src(src + 'html/**/*')
.pipe(newer(out))
.pipe(devBuild === "No" ? noop() : htmlclean())
.pipe(gulp.dest(out));
}

function js_index() {
return gulp.src(src + 'js/index/**/*')
.pipe(preprocess({ context: { URI: uri}}))
.pipe(sourcemaps ? sourcemaps.init() : noop())
.pipe(deporder())
.pipe(concat('index.js'))
.pipe(stripdebug ? stripdebug() : noop())
.pipe(terser())
.pipe(sourcemaps ? sourcemaps.write() : noop())
.pipe(gulp.dest(build + 'js/'));
}

function js_reader() {
return gulp.src(src + 'js/reader/**/*')
.pipe(preprocess({ context: { URI: uri}}))
.pipe(sourcemaps ? sourcemaps.init() : noop())
.pipe(deporder())
.pipe(concat('reader.js'))
.pipe(stripdebug ? stripdebug() : noop())
.pipe(terser())
.pipe(sourcemaps ? sourcemaps.write() : noop())
.pipe(gulp.dest(build + 'js/'));
}

function js_jquery() {
return gulp.src(src + 'js/jquery.js')
.pipe(preprocess({ context: { URI: uri}}))
.pipe(sourcemaps ? sourcemaps.init() : noop())
.pipe(deporder())
.pipe(concat('jquery.js'))
```

```
.pipe(stripdebug ? stripdebug() : noop())
.pipe(terser())
.pipe(sourcemaps ? sourcemaps.write() : noop())
.pipe(gulp.dest(build + 'js/'));
}

function css() {
return gulp.src(src + 'scss/**/*')
.pipe(concat('style.css'))
.pipe(gulp.dest(build + 'css/'));
}

exports.js_index = js_index;
exports.js_index = js_jquery;
exports.js_reader = js_reader;
exports.images = images;
exports.html = gulp.series(images, html);
exports.build = gulp.parallel(images, html, js_index,
   js_reader, js_jquery, css)
```

Tras unas primeras secciones para declarar variables útiles, importar librerías y emitir mensajes de *debug*, se encuentra un fragmento en el que se declaran las funciones que implementan las tareas deseadas. En primer lugar, se comprimen las imágenes de la página para reducir los tiempos de carga y el tamaño de la aplicación. A continuación se limpia el código *html*, minimizando los ficheros. Después, existen una serie de funciones para ofuscar los ficheros *JavaScript*, eliminar los mensajes de *debug* y unificar los ficheros. Por último, se procesan y unifican los archivos *CSS*.

2.2. Tecnologías funcionales

En esta sección se encuentran aquellas tecnologías que han sido escogidas para dar respuesta a las necesidades técnicas del proyecto. Por lo tanto, se encuentran aquí las principales librerías o sistemas para los servidores, las bases de datos o el cumplimiento de determinadas tareas en tiempo de ejecución. Además, son una parte integradora del producto final, a diferencia de las tecnologías enumeradas en otras secciones.

2.2.1. Express.js

Express.js es un *framework* para aplicaciones web basado en *Node.js*[8], lo cual ha servido para construir una *API* que de respuesta a las peticiones de los usuarios de la aplicación.

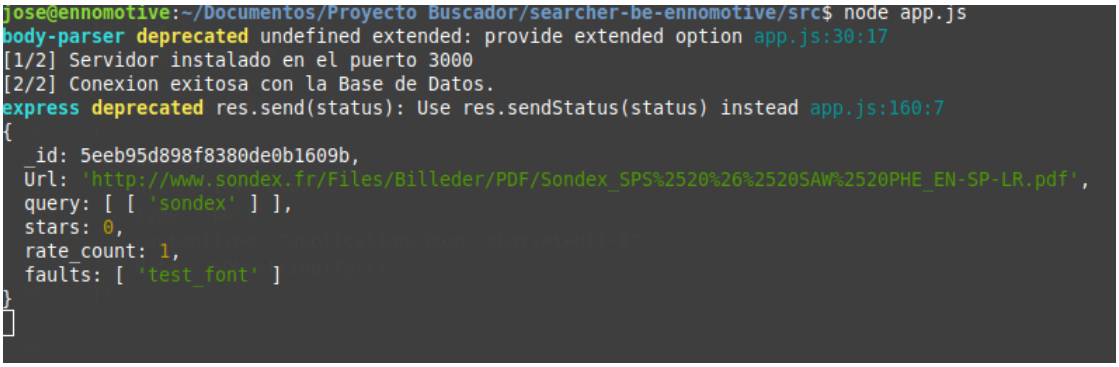

Figura 2.2: Servidor Express en ejecución local.

Se trata de una tecnología altamente flexible que permite construir aplicaciones con funcionalidad completa en pocas líneas de código. Como se puede comprobar en el ejemplo 2.3, se necesitan una serie de funciones para determinar las terminaciones de la *url* que debe escuchar, así como una llamada para iniciar la escucha.

Listing 2.3: Ejemplo de código para Express.

```
const express = require('express');

app.get('/', function(req, res){
        // Funcionalidad definida por el usuario.
        res.send(200);
        return;
});

app.listen(port, () => console.log(`[1/2] Servidor
   instalado en el puerto ${port}`));
```

2.2.2. Flask

De manera similar a la sección anterior, *Flask* es un *framework* para aplicaciones web. La diferencia radica en que emplea *python* para su funcionamiento [9]. Su función en el proyecto es la de dotar a un *worker* de la capacidad de interactuar con la plataforma en la que se aloja.

De manera análoga, el funcionamiento es similar al de *Express.js*. Se emplean funciones especiales para declarar las rutas en las que la aplicación debe escuchar. Sin embargo, en este caso se debe ejecutar la aplicación con un comando específico. Esto se demuestra en el *listing* 2.4.

Listing 2.4: Ejemplo de código para Flask.

```
from flask import Flask, escape, Response
from flask import request as rq

application = Flask(__name__)

@application.route('/', methods=['GET', 'POST'])
def hello_world():
        return json.dumps({'success':True}), 200, {'
            ContentType':'application/json'}
```

2.2.3. MongoDB

MongoDB es una base de datos de propósito general potente, flexible y escalable. Su naturaleza no relacional permite almacenar objetos tipo *JSON* a través de diferentes colecciones. Su papel central en el contexto de desarrollo de *software* actual facilita su implementación a través de librerías de uso común [10]. En el *listing* 2.5 se encuentra un ejemplo de objeto.

Listing 2.5: Ejemplo de objeto MongoDB.

```
{"_id":{"$oid":"5e3ab29512d2973f5e9b3e62"},"id":"rate",
 "rate":"Error en el documento de url Tutorial_EDIT.pdf.
 \"bugs.python.org refused to connect\""}
```

2.3. Tecnologías de despliegue

En la siguiente sección se enumeran las tecnologías necesarias para albergar los diferentes elementos de la aplicación y permitir la conexión de los usuarios. Se trata en su totalidad de servicios externos que, en ocasiones, han determinado la elección de tecnologías en las secciones anteriores.

2.3.1. Amazon Web Services

La mayoría de las tecnologías en esta categoría se encuentran en los servicios web de *Amazon*. La elección de estos se ha debido principalmente a la presencia de

servicios previamente contratados en la misma compañía. De entre las diferentes opciones disponibles, se ha optado por las que se listan a continuación.

S3

S3 es el servicio de almacenamiento simple (*Simple Storage Service*) de *Amazon*. Permite el almacenamiento de archivos de cualquier tipo, así como su distribución[11]. Por lo tanto, puede ser utilizado para albergar archivos *html, css* y *JavaScript*, lo que sirve para desplegar la parte frontal de una aplicación web sin necesidad de gestionar directamente un servidor. Adicionalmente, se usa como almacenamiento para el código y ficheros estáticos del resto de elementos de la aplicación.

EC2

EC2 es una infraestructura de computación escalable que permite ejecutar, entre otras cosas, contenedores *Docker*[12]. Esto sirve, por un lado, para ejecutar las *pipelines* tras cada actualización y, por otro, albergar el servidor para la *API* construida con *Express.js* y el *worker Flask*.

Elastic Beanstalk

Elastic Beanstalk es un servicio de Amazon que facilita la tarea de desplegar aplicaciones web. Esto se consigue a través de de funcionalidades como el escalado automático, el envío de notificaciones, la monitorización, etc[13].

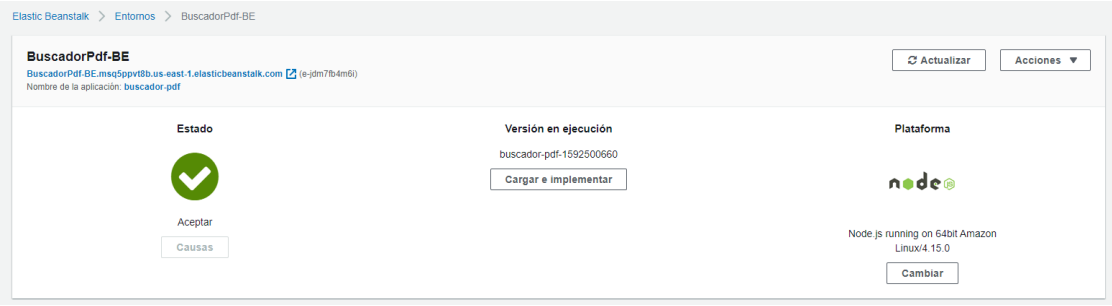

Figura 2.3: Captura de pantalla de la interfaz web de Elastic Beanstalk.

2.3.2. Atlas

Atlas es un servicio de gestión para bases de datos *MongoDB* en la nube [14]. Si bien no es la única opción disponible para albergar la base de datos, la facilidad

para elaborar prototipos y el hecho de tratarse del mismo equipo de desarrollo de *MongoDB* jugaron un papel clave en la elección de este servicio. Además de albergar los datos, ofrece funcionalidades para el acceso directo a los datos con el fin de revisarlos y editarlos.

Capítulo 3

Estado de la Cuestión

Antes de abordar los detalles del proyecto, conviene dedicar un espacio a revisar los últimos aportes teóricos y prácticos al campo del desarrollo de *software*. Por lo tanto, en la siguiente sección se llevará a cabo una síntesis de aquellas referencias que hayan servido en la planificación y elaboración del proyecto.

Por un lado, existen numerosas contribuciones recientes relacionadas con nuevos modelos de despliegue para aplicaciones en la red. Concretamente, se ha usado el modelo *serverless* para eliminar la responsabilidad de configurar, mantener y actualizar un servidor para aquellas partes de la aplicación donde no es necesario [15]. Además, se ha aplicado la arquitectura de microservicios para aislar funciones en aplicaciones específicas con un único propósito [16] y utilizando, donde ha sido posible, servicios ya existentes, permitiendo concentrar la actividad de desarrollo en las necesidades específicas del proyecto.

Adicionalmente, se ha prestado atención a la literatura reciente sobre clasificación de documentos [17]. Esto permite implementar lo más temprano posible las mejores prácticas durante el proceso de desarrollo para agilizar la clasificación en etapas posteriores del proyecto. Además, ha sido necesario revisar métodos para la búsqueda de documentos [18] con tal de minimizar los tiempos de búsqueda y ofrecer los mejores resultados.

Capítulo 4

Definición del Trabajo

Con los precedentes del proyecto cubiertos, conviene definir los aspectos que lo han motivado y delimitado. Este proyecto se engloba dentro de otro superior cuya finalidad es desarrollar una aplicación web la cual, a través de sistemas de inteligencia artificial, permita a los usuarios acceder a documentación de carácter técnico a través de un buscador. A continuación se detalla la justificación para elaborar el proyecto, los objetivos de este y la metodología y planificación para llevarlo a cabo.

4.1. Justificación

El elemento central del proyecto principal es un sistema de inteligencia artificial que permita aproximar una valoración subjetiva a partir de un documento de texto. Para cumplir con este objetivo, es necesario entrenar el sistema con un número significativo de valoraciones aportadas por los usuarios.

Además, dicho sistema debe ir acompañado de toda la infraestructura necesaria para dotar al conjunto de una interfaz web, almacenamiento de datos, validación de usuarios, preprocesado de documentos, etc. Es decir, además del sistema central dotado de inteligencia se necesitan otros sistemas que permitan al conjunto funcionar como una aplicación web completa.

Para satisfacer dichas necesidades previas, se ha optado por desarrollar un prototipo que cumpla con los requisitos básicos del sistema y permita recopilar valoraciones de un grupo selecto de usuarios. Consecuentemente, se necesita previamente un prototipo que sirva, a través de un sistema de control de acceso, como buscador primigenio con tal de dotar al usuario de documentos sobre los que hacer valoraciones.

4.2. Objetivos

De la justificación anterior se pueden sintetizar los objetivos fundamentales que debe cubrir el proyecto, así como metas secundarias que completar si se dispone de los recursos. El objetivo principal consiste en recoger de forma estructurada las valoraciones de los usuarios. Para esto, se debe crear una aplicación web que permita buscar documentos y valorarlos, así como un sistema que permita el registro y la validación de usuarios.

Por otro lado, sería deseable implementar mecanismos de seguridad actualizados, así como elaborar pruebas que permitan asegurar la integridad, disponibilidad y privacidad de los datos almacenados en la plataforma. No obstante, al tratarse de una etapa prematura del desarrollo puede atrasarse el cumplimiento de esta tarea si las condiciones lo exigieran.

4.3. Metodología y planificación

Para el desarrollo del proyecto se ha optado por implementar una metodología ágil. Se ha determinado dicha metodología como la más adecuada por múltiples razones. En primer lugar, ha permitido disponer de prototipos con relativa rapidez sobre los cuales implementar nuevas funcionalidades según haya dictado la conveniencia. Además, se trata de un proceso sencillo de implementar, lo cual se adapta a las condiciones del equipo de desarrollo. Por último, permite aprovechar al máximo las condiciones de estrecha relación entre todas las partes involucradas en el proyecto.

Dicha metodología se basa en *sprints* al final de los cuales se debe presentar un prototipo con objetivos a corto plazo. A raíz de la práctica y del análisis de los resultados se extraen nuevos objetivos que deben ser cubiertos con el próximo ciclo de desarrollo [19]. Esto permite a equipos pequeños trabajar, a través de incrementos progresivos, en aquellas necesidades que tengan mayor urgencia en cada momento del proceso.

La naturaleza cambiante y adaptativa de la metodología de desarrollo dificulta la tarea de establecer una planificación clara a largo plazo. No obstante, es posible seguir un hilo de dependencias con el fin de determinar qué tareas deben ser completadas como requisito para otras. Gracias a esto y a una aproximación del tiempo necesario para cada tarea particular, se puede establecer un cronograma que sirva como referencia para el desarrollo del proyecto entre los meses de enero y julio de 2020. Dichas tareas comienzan con la creación de los servicios que albergarán las diferentes partes de la aplicación, así como las *pipelines* que permitan automatizar las tareas de actualización del código. Adicionalmente, será necesario crear un prototipo de la librería encargada del procesamiento de los documentos,

Tarea	Enero	Febrero	Marzo	Abril	Mayo	Junio
Crear servicios AWS	X					
Pipelines	X					
Librería Procesador	X					
API, Worker y Frontend		X				
Implementación Datatables		X				
Autenticación usuarios			X			
Mejora crawler				X		
Mejora búsquedas					X	
Documentación						X

Cuadro 4.1: Cronograma del proyecto

requisito para la introducción de información en el sistema. Seguidamente, se deben crear los prototipos para las partes elementales del sistema: *API*, *worker* y *frontend*. A continuación, se deberá iterar sobre estos prototipos con la finalidad de implementar nuevas funcionalidades y mejorar las presentes. Por último, se deberá recabar la información obtenida a lo largo de todo el proceso y documentarlo. El cronograma del proyecto se encuentra en la tabla 4.1.

Capítulo 5

Sistema Desarrollado

Tras haber cubierto los precedentes del proyecto, es preciso continuar con una descripción detallada del trabajo realizado. El primer paso para esto es definir de forma concreta los requisitos que debe satisfacer el sistema y los límites dentro de los cuáles debe desenvolverse. A continuación, se definen las decisiones que se han llevado a cabo para su consecución, presentando los extractos del código del programa má representativos. Por último, se detallan las condiciones de despliegue del proyecto para su funcionamiento.

5.1. Análisis del Sistema

Para abordar la tarea de diseño de forma adecuada, es imprescindible elaborar un análisis previo de las tareas que se debe cubrir con el sistema y de las restricciones que limitan la operación de este. Por lo tanto, es necesario establecer un listado de los requisitos de alto nivel.

De acuerdo con lo establecido anteriormente, uno de los requisitos fundamentales consiste en permitir introducir una cadena de texto para buscar documentos. Además, para controlar al acceso a la plataforma, se deberá permitir a un usuario cualquiera registrarse para ser verificado manualmente y validar sus credenciales si ya dispone de ellas. Por otro lado, se dispondrá de una herramienta para emitir reportes con el fin de descubrir con rapidez los errores y recibir sugerencias. Una vez se ha resuelto la búsqueda de documentos, la aplicación deberá devolverle al usuario una lista de estos junto con sus parámetros esenciales, como la dirección o la puntuación. Una vez se selecciona un documento, se deberá abrir un lector en otra pestaña y marcar el documento como leído. Dentro del lector, se dotará al usuario de una interfaz para leer el documento y la opción de enviar una valoración.

Adicionalmente, el sistema deberá ser capaz de buscar nuevos documentos si una búsqueda determinada se repite, así como eliminar cualquier documento que

sea reportado como erróneo o no disponible. Además, la aplicación deberá ser capaz de procesar los documentos para extraer parámetros útiles como su ratio de números por palabra, lo que ayuda a determinar la naturaleza del documento.

5.2. Diseño

Para abordar el diseño de la plataforma se optó por dividirla entre varios subsistemas interconectados con un objetivo específico. En primer lugar, existe un *frontend* cuya finalidad es servir como interfaz con el usuario final y presentar los documentos. Interacciona a través de peticiones *http* con la *API*, cuyo objetivo es el de hacer de intermediario entre el *frontend* y la base de datos, además de incluir funcionalidad propia para el control de los usuarios y buscar documentos fuera de la aplicación. La base de datos, lógicamente, gestiona el almacenamiento de los diferentes tipos de objeto que fluyen a través de la aplicación. Por último, el *worker* toma documentos de la base de datos y los somete a un procesado extenso pero costoso, extrayendo toda la información posible para utilizarla en el futuro. En la figura 5.1 se encuentra un esquema de la arquitectura.

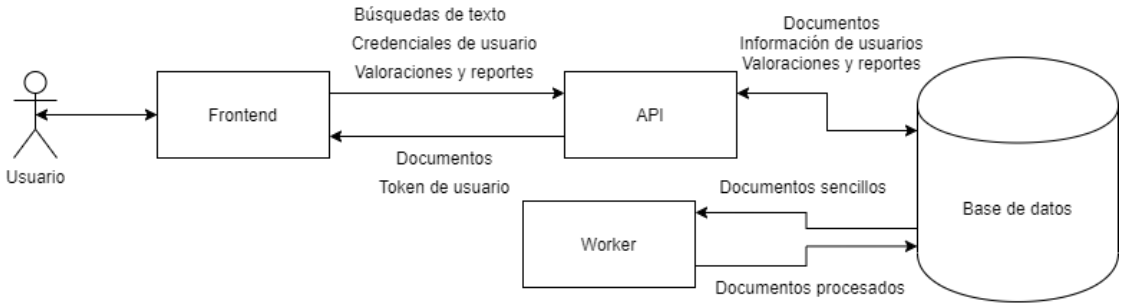

Figura 5.1: Arquitectura del sistema.

5.2.1. Frontend

El *frontend* está formado por una serie de archivos *html*, *CSS*, *JavaScript* e imágenes a los cuales los usuarios pueden acceder libremente a través de un navegador. Son archivos estáticos que se interpretan y ejecutan en la máquina desde la que accede el usuario. Dada la exposición al público de esta parte, cualquier tipo de validación o acceso a la base de datos debe realizarse a través de intermediarios, por lo que tiene la capacidad para hacer peticiones al *backend* y permanecer a la espera de resultados.

El fichero a través del cual se accede a la plataforma es *index.html*. En él, se describe la página inicial y se cargan los *scripts* que permiten ejecutar búsquedas,

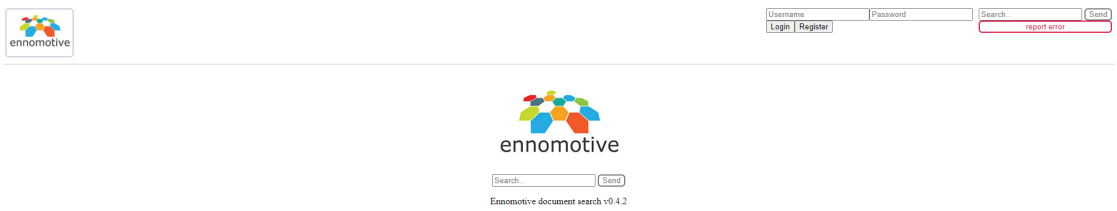

Figura 5.2: Vista de inicio del fichero index.html .

registrarse como usuario, validar las credenciales y emitir reportes. Esto se puede observar en la captura de pantalla presente en la figura 5.2. Tras emitir una búsqueda y recibir una respuesta, se utiliza la librería *DataTables* para mostrar una tabla con los documentos recibidos, como se demuestra en la figura 5.3.

Figura 5.3: Vista index.html tras recibir documentos (usuario validado).

Tras oprimir el botón *open*, se carga en una pestaña nueva el fichero *reader.html* como se muestra en la figura 5.4. Una vez dentro, el lector carga el fichero desde su *url* y se muestra embebido en la página. Por encima, se muestra un recuadro donde se ofrece la opción de emitir una valoración, además de el recuadro para indicar si ha habido algún problema al cargar el documento.

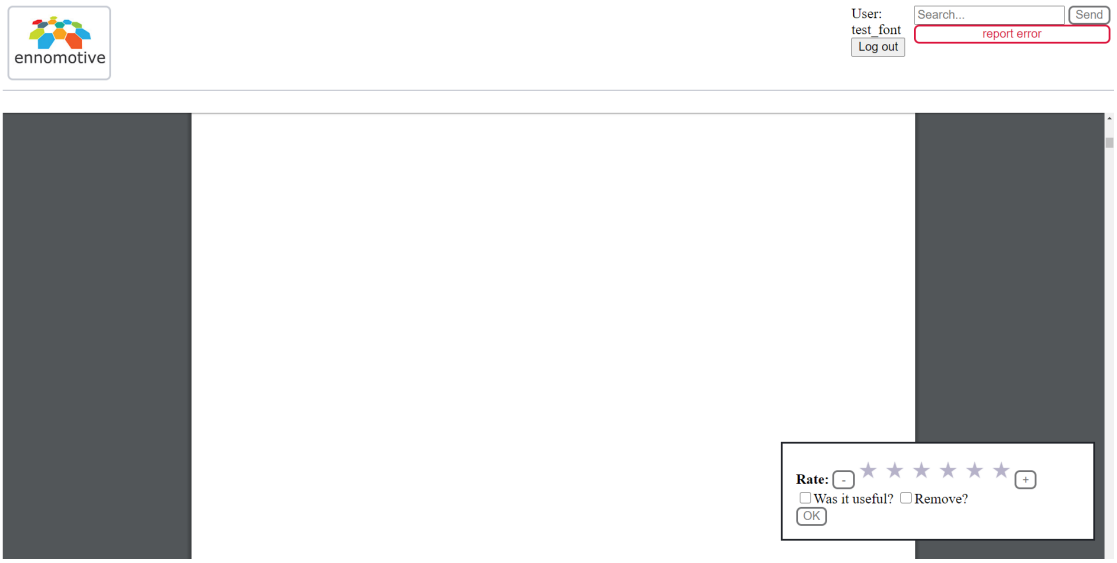

Figura 5.4: Vista del fichero reader.html.

Además de la librería *DataTables*, se utiliza *jQuery* para facilitar muchas de las tareas. Además de permitir cambiar elementos con facilidad, esta librería permite intercambiar información con el *backend* usando *ajax*, como se muestra en el *listing* 5.1. Adicionalmente, se utilizan *cookies* para almacenar localmente la información del usuario y los documentos visitados.

Listing 5.1: Función que emplea Ajax y Cookies para enviar un reporte.

```
function send_report() {
        let field = $("#report_text").val();
        var rate={"id": "rate", "rate": field, "user":
           getCookie("user"), "tok": getCookie("tok")};
        $.ajax({
                url: URI + 'rate',
                type: 'post',
                contentType: "application/json; charset=utf
                   -8",
                data: JSON.stringify(rate)});
        $("#reporter").remove();
}
```

Ruta	Acción	Función
/	GET	Buscar documentos de acuerdo a la query presente en el cuerpo del mensaje.
/user	POST	Registrar/Validar las credenciales de usuario presentes en el cuerpo del mensaje.
/rate	POST	Publicar una valoración o reportar un error.

Cuadro 5.1: Resumen API

5.2.2. API

Este elemento tiene la función de servir como intermediario entre la base de datos y el *frontend*, de verificar los *tokens* de usuario y de buscar documentos entre diferentes fuentes. Lo primero se consigue dando respuesta a las peticiones que recibe en las rutas en las que se encuentra escuchando. Las rutas disponibles se resumen en la tabla 5.1. Emplea la librería *mongodb* para intercambiar información con la base de datos. La verificación de los usuarios se realiza validando una serie de *tokens* creados con la librería *jsonwebtoken*. Dichos *tokens* se crean utilizando una clave local que sólo puede ser verificada por la aplicación, lo que previene los accesos no autorizados. La tarea de obtener documentos externos se consigue utilizando *Axios* para ejecutar *requests* de manera automática y *Cheerio* para *parsearlas* en busca de documentos. Esto se demuestra en el *listing* 5.2.

Listing 5.2: Parseo de documentos.

```
axios(url)
        .then(response => {
                const html = response.data;
                const $ = cheerio.load(html);
                const embeds = $('a');
                var links = [];
                embeds.each(function (){
                        links.push($(this).attr('href'));
                });
        });
```

5.2.3. Worker

La función del *worker* consiste en encontrar documentos en la base de datos y extraer, tras procesar el texto, una serie de parámetros que puedan servir para su clasificación y búsqueda en iteraciones posteriores del proyecto. Al tratarse de un proceso costoso, no puede integrarse en la interacción habitual de los usuarios, por lo que debe permanecer procesando documentos de forma permanente. Además, se pueden ejecutar múltiples tareas de manera simultánea si los recursos se encuentran

disponibles.

Utiliza numerosas librerías como *PyPDF2* y *pdfminer* para extraer texto e imágenes de documentos *pdf* y los interpreta para determinar la presencia de índices o bibliografía, el idioma, el número de ecuaciones, la cantidad de unidades económicas como *euros* o *dólares*, etc. Además, está preparado para integrar con facilidad herramientas de clasificación construidas con *sklearn*, con la finalidad de poder dar una clasificación al texto como parte del procesado en el futuro.

En la fase actual del proyecto, este elemento se encuentra en una etapa más primitiva. No obstante, servirá como esqueleto para la construcción de funcionalidades adicionales con rapidez, dado que está diseñado con la intención de devolver objetos *JSON*, a los cuales se les puede añadir información de manera muy sencilla.

5.2.4. Base de datos

La base de datos almacena objetos tipo *JSON* a través de diferentes colecciones. Cada colección cumple un propósito determinado, lo cual ayuda a mantenerla ordenada y a encontrar los objetos deseados con facilidad. La colección principal es la de documentos, llamada *Files*. En ella se almacenan los documentos, procesados y sin procesar, que posteriormente se servirán a los usuarios. Por otro lado, existe una colección para los datos de los usuarios, llamada *Users* y otra para almacenar valoraciones, reportes y mensajes del sistema llamada *Logs*. Por último, existe una colección llamada *Querys* destinada a almacenar las búsquedas de los usuarios, con el objetivo de analizarlas para mejorar el funcionamiento del buscador en iteraciones futuras.

5.3. Implementación

Con la excepción de la base de datos, a la que se dedicará una sección específica, todos los elementos de la plataforma han sido desplegados a través de los servicios web de Amazon. Adicionalmente, se han implementado los mecanismos para poner a prueba el código y subirlo a su destino tras cada actualización en la rama principal de sus respectivos repositorios. Para lograr esto, ha sido necesario disponer de una serie de recursos configurados previamente los cuales se detallan a continuación.

En primer lugar, es necesaria una instancia *EC2* a la que conectar los repositorios para ejecutar las *pipelines* correspondientes tras cada actualización. Además de esta primera instancia, se necesitan otras por cada aplicación individual que se quiera ejecutar en la nube. En el caso de este proyecto se trata de dos, el *worker* y el *backend*, dado que el *frontend* se despliega de una manera distinta. En la figura 5.5 se pueden distinguir las dos instancias para las aplicaciones *BuscadorPdf-BE* y

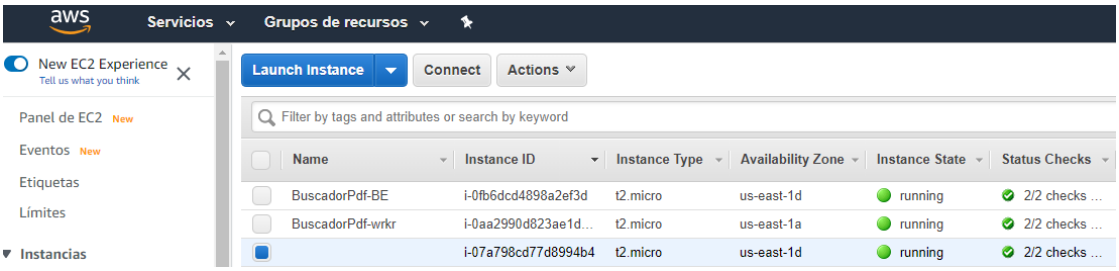

Figura 5.5: Captura de las instancias EC2.

BuscadorPdf-wrkr, así como una instancia sin nombre que sirve como *runner* para las *pipelines* de los repositorios.

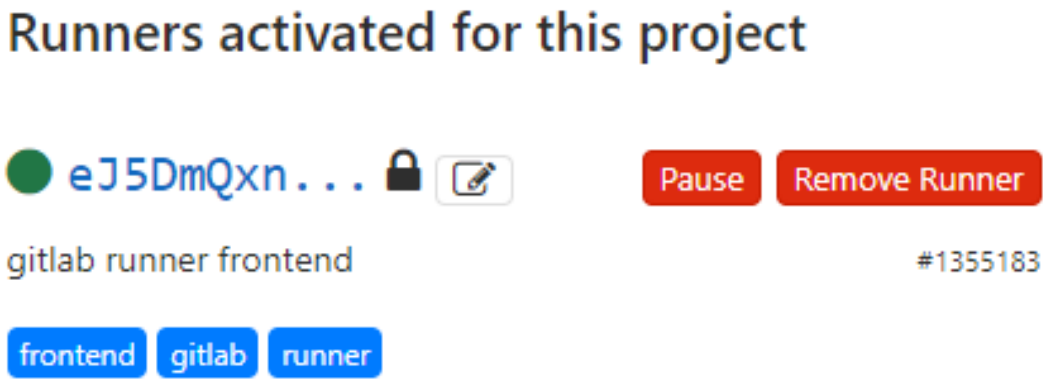

Figura 5.6: Captura del runner configurado en el repositorio del frontend.

A continuación, se deben configurar los entornos en *Elastic Beanstalk* para ambas aplicaciones. Dicho entorno servirá para cargar automáticamente el código en la instancia, comprobar el estado, acceder a los registros, establecer alarmas y gestionar automáticamente el consumo de recursos de cada una de las aplicaciones. De todas las funcionalidades, la más utilizada durante el proceso de desarrollo ha sido la de acceder a los registros, ya que ha permitido diagnosticar el funcionamiento de cada aplicación e identificar errores en el lanzamiento. Un ejemplo de dicho registro se puede encontrar en 5.7.

Por último, la aplicación de un modelo *serverless* al *frontend* ha permitido desplegarlo en un *bucket S3*. Al tratarse de un servicio de almacenamiento de ficheros, permitía cargar los archivos actualizados con relativa facilidad. No obstante, al ofrecer una *url* pública permite el acceso a cualquier usuario que disponga del enlace. Esto se encuentra ejemplificado en la figura 5.8.

```
----------------------------------------
/var/log/nodejs/nodejs.log
----------------------------------------
Fri, 03 Jul 2020 22:02:42 GMT body-parser deprecated undefined extended: provide extended option at app.js:30:17
[1/2] Servidor instalado en el puerto 8081
[2/2] Conexion exitosa con la Base de Datos.
Fri, 03 Jul 2020 22:28:47 GMT express deprecated res.send(status): Use res.sendStatus(status) instead at app.js:45:9
Fri, 10 Jul 2020 22:33:32 GMT express deprecated res.send(status): Use res.sendStatus(status) instead at app.js:158:7


----------------------------------------
/var/log/nginx/error.log
----------------------------------------



----------------------------------------
/var/log/eb-activity.log
----------------------------------------
  + rm -rf /var/app
[2020-07-03T22:02:40.074Z] INFO  [3108]  - [Application deployment buscador-pdf-1592500660@27/StartupStage1/AppDeployEnactHook/30app_deploy.sh] : Starting activity...
[2020-07-03T22:02:40.465Z] INFO  [3108]  - [Application deployment buscador-pdf-1592500660@27/StartupStage1/AppDeployEnactHook/30app_deploy.sh] : Completed activity. Result:
```

Figura 5.7: Captura de un registro reciente en el entorno del Backend

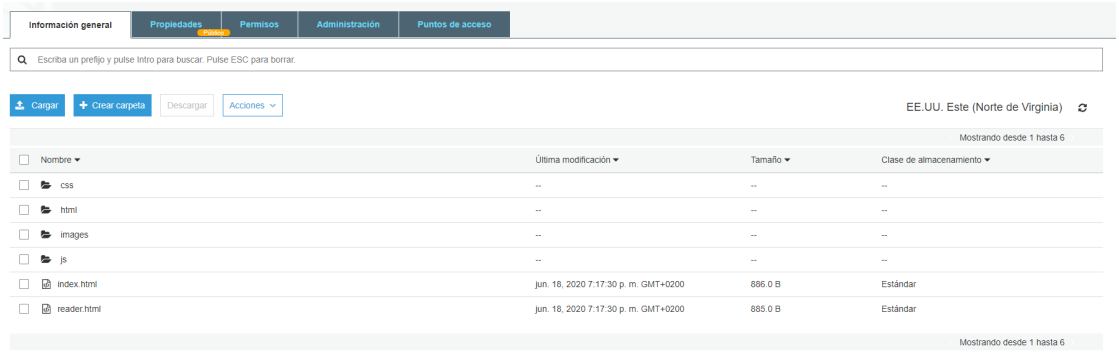

Figura 5.8: Captura del bucket que alberga el frontend.

Por otro lado, la base de datos es accesible a través del servicio *Atlas*. La única complicación consiste en el acceso seguro de las aplicaciones. Para ello, se dispone de una *uri* con los datos de conexión, incluyendo la contraseña. Para impedir el acceso no autorizado, dicha contraseña se encuentra protegida en variables de entorno a las cuales sólo las aplicaciones en ejecución tienen acceso. Tras obtener la contraseña de las variables de entorno, es introducida en la *uri* y utilizada para crear una conexión.

Capítulo 6

Análisis de Resultados

Tras concluir el capítulo dedicado al desarrollo del sistema y una vez se han alcanzado las últimas etapas del proyecto, conviene dedicar un espacio a analizar los resultados. Esto permitirá extraer información valiosa que haya pasado desapercibida durante el desarrollo y que sirva como guía para proyectos futuros.

En términos cuantitativos, se han recolectado más de 1200 documentos con tan solo 3 usuarios. Además, se han recopilado cerca de 40 valoraciones o reportes emitidos por los usuarios. En total, el espacio ocupado por los datos recolectados asciende hasta los 28 MB. Esto es indicativo de un uso significativamente eficiente del espacio de almacenamiento, ya que un único fichero, antes de ser parametrizado, suele tener un tamaño cercano al *MegaByte*. Sin embargo, el número de conexiones concurrentes a la base de datos oscila alrededor de los 50, lo que indica un uso mejorable de los recursos computacionales.

El tiempo de respuesta para una búsqueda aleatoria se encuentra cerca de los dos segundos, lo que supone un tiempo de espera aceptable [20]. Sin embargo, el tiempo acceso a los documentos es significativamente mayor. Esto se debe, en primer lugar, al tamaño de los propios documentos y, en segundo lugar, a la heterogeneidad de las localizaciones de estos. Además, cualquier cambio en la ubicación original del documento puede originar la pérdida del mismo. Actualmente, no existen estimaciones de qué porcentaje de documentos ha perdido validez ni con qué frecuencia ocurre, pero se han implementado medidas para reportarlos y ocultarlos.

En términos cualitativos, las búsquedas en ocasiones devuelven documentos que tienen poco o nada que ver con los términos introducidos, lo que apunta a una simplicidad excesiva de los métodos de búsqueda. Además, en ocasiones los resultados parecen oscilar entre lo puramente académico y meros panfletos comerciales. Por otro lado, los nombres que se muestran para los documentos en la página web aportan muy poca información sobre su contenido. Adicionalmente, la página web carece de algunas medidas para mejorar la experiencia del usuario como

la adaptación a pantallas de dispositivos móviles o la ocultación de la contraseña durante la validación.

Capítulo 7

Conclusiones y Trabajos Futuros

Habiendo concluido el análisis de los resultados del proyecto, es adecuado sintetizarlos en una serie de conclusiones y utilizarlas para proponer objetivos que puedan ser satisfechos por las etapas futuras del proyecto. En definitiva, se trata de un sistema que cumple con su objetivo de recopilar valoraciones, gracias en parte a la funcionalidad efectiva de validación de usuarios. Además, sirve como base sobre la que edificar un sistema más elaborado dado que dispone, aunque sea de forma primitiva, de las partes necesarias para evolucionar sin necesidad de grandes reestructuraciones. No obstante, se ha extendido más de lo deseable y ha faltado el tiempo para realizar todas las pruebas para garantizar la seguridad, por lo que conviene restringir el acceso exclusivamente a usuarios de confianza hasta que se hayan podido llevar a cabo.

De las conclusiones se extrae la necesidad de mejorar los métodos de búsqueda, ya que los resultados que ofrecen actualmente no son satisfactorios. Además, conviene investigar sistemas para prevenir la ruptura del enlace de los documentos, así como la detección y eliminación automática de estos. Por otro lado, sería beneficioso explorar alternativas para solucionar el problema de los tiempos de carga de los documentos. Por último, se recomienda dedicar recursos a la mejora del diseño y la experiencia de usuario del *frontend*.

Sin embargo, es de vital importancia aprovechar los datos extraídos a través de la aplicación para construir sistemas de inteligencia artificial que sean útiles para los usuarios de *Ennomotive*. Estos datos podrían ser utilizados, además de para buscadores, para calificar automáticamente las soluciones enviadas por los usuarios así como indicarles qué mejoras podrían implementar en estas.

Apéndice A

Alineación del proyecto con los ODS

De acuerdo a lo establecido en la normativa, el proyecto cumple con aquellos objetivos de desarrollo sostenible establecidos por la *ONU* que aplican a la naturaleza del proyecto y a la actividad de la empresa en la que se ha realizado, *Ennomotive*. En el siguiente anexo se pretende argumentar a favor de la alineación del proyecto con dichos objetivos.

Por un lado, el objetivo fundamental del proyecto es el de ayudar a los usuarios de la plataforma a resolver los problemas que en esta se plantean. En dicha plataforma, han sido publicados en numerosas ocasiones desafíos de carácter solidario y beneficiosos para la humanidad, los cuales habitualmente contribuyen al cumplimiento de los objetivos propuestos por la Organización de las Naciones Unidas. Como ejemplo de esto se encuentra el desafío para diseñar respiradores en respuesta a la crisis del *COVID-19* [21], lo cual se alinea con el objetivo tercero de salud y bienestar en el epígrafe octavo, a través del abaratamiento de los costes para el tratamiento de enfermedades respiratorias en general y del *COVID-19* en particular. Esto sirve como prueba para argumentar que los resultados del proyecto serán utilizados en desafíos futuros, de entre los cuáles existirán muchos que trabajen activamente por cumplir estos objetivos de desarrollo sostenible.

Además, el propio funcionamiento de los desafíos en los que se incluirá la plataforma desarrollada contribuye al cumplimiento de los *ODS*. Por un lado, se contribuye al cumplimiento del objetivo *8.3*, permitiendo a usuarios de todo el mundo obtener beneficios al resolver problemas industriales, especialmente jóvenes (epígrafe 8.6). Además, el carácter virtual de la participación contribuye al cumplimiento del objetivo 10.3 al considerarse las soluciones aportadas como factor decisivo independientemente del sexo, raza, religión, etc. del usuario.

A través de los argumentos anteriormente expuestos, puede demostrarse el impacto positivo del proyecto en los esfuerzos de la *ONU* por alcanzar sus metas de

desarrollo sostenible. Si bien es complicado medir exactamente de qué magnitud es dicho impacto, cabe destacar que se trata de un esfuerzo colaborativo cuyo primer logro consiste en conseguir el avance, por pequeño que sea, hacia un futuro más sostenible.

Bibliografía

[1] David Bretthauer. "Open source software: A history". En: (2001).

[2] Daren C Brabham. *Crowdsourcing*. Mit Press, 2013.

[3] ENNOMOTIVE SL. *Ennomotive*. `https://www.ennomotive.com/es/`. 2020.

[4] Jon Loeliger y Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. .º'Reilly Media, Inc.", 2012.

[5] Jonathan M Hethey. *GitLab Repository Management*. Packt Publishing Ltd, 2013.

[6] GITLAB. *Gitlab Pipeline*. `https://docs.gitlab.com/ee/ci/pipelines/`. 2020.

[7] Travis Maynard. *Getting Started with Gulp–Second Edition*. Packt Publishing Ltd, 2017.

[8] Azat Mardan. *Express. js Guide: The Comprehensive Book on Express. js*. Azat Mardan, 2014.

[9] Miguel Grinberg. *Flask web development: developing web applications with python*. .º'Reilly Media, Inc.", 2018.

[10] Kristina Chodorow. *MongoDB: the definitive guide: powerful and scalable data storage*. .º'Reilly Media, Inc.", 2013.

[11] James Murty. *Programming amazon web services: S3, EC2, SQS, FPS, and SimpleDB*. .º'Reilly Media, Inc.", 2008.

[12] Simson Garfinkel. "An evaluation of Amazon's grid computing services: EC2, S3, and SQS". En: (2007).

[13] Jurg Vliet y col. *Elastic beanstalk*. .º'Reilly Media, Inc.", 2011.

[14] MongoDB. *MongoDB Atlas*. `https://docs.atlas.mongodb.com//`. 2020.

[15] Diego Zanon. *Building Serverless Web Applications*. Packt Publishing Ltd, 2017.

[16] Johannes Thönes. "Microservices". En: *IEEE software* 32.1 (2015), págs. 116-116.

[17] Zichao Yang y col. "Hierarchical attention networks for document classification". En: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, págs. 1480-1489.

[18] J Shane Culpepper y col. "Top-k ranked document search in general text databases". En: *European Symposium on Algorithms*. Springer. 2010, págs. 194-205.

[19] Agile Manifesto. "Agile manifesto". En: *Haettu* 14 (2001), pág. 2012.

[20] Fiona Fui-Hoon Nah. "A study on tolerable waiting time: how long are web users willing to wait?" En: *Behaviour & Information Technology* 23.3 (2004), págs. 153-163.

[21] ENNOMOTIVE. *Respiradores UCI válidos para COVID-19: Desafío de Diseño*. `https://www.ennomotive.com/es/respiradores-uci-covid-19//`. 2020.