EXTRACTOR DE DATOS BIBLIOGRÁFICOS FLR

TRABAJO FIN DE GRADO
CURSO 2020-2021

AUTORES:
FRANCISCO RAFAEL GARCÍA ROFES
RUBÉN MARTÍN ACEBEDO
LUIS ANTONIO ROJAS RAMIREZ

DIRECTORES:
ENRIQUE MARTÍN MARTÍN
ADRIÁN RIESCO RODRIGUEZ

GRADO EN INGENIERÍA DE SOFTWARE
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

Bibliographic Data Extractor FLR

Trabajo de Fin de Grado en Ingeniería de Software
Departamento de Sistemas Informáticos y Computación

Autores:
Francisco Rafael García Rofes
Rubén Martín Acebedo
Luis Antonio Rojas Ramirez

DIRECTORES:
Enrique Martín Martín
Adrián Riesco Rodriguez

Convocatoria: Junio 2021

Grado en Ingeniería de software
Facultad de Informática
Universidad Complutense de Madrid

15 de Junio de 2021

RESUMEN

Data Extractor FLR

Este proyecto consiste en el desarrollo de una aplicación web donde los usuarios sean capaces de obtener de forma ordenada y sin redundancia los datos de investigadores de numerosos sitios institucionales, recursos académicos y documentos de investigación. Estos datos se extraen mediante un procedimiento consistente en *scraping*, y se devuelve dicha información en un formato JSON, accesible a través de una interfaz web, para su posterior uso.

Se puede acceder a la aplicación web y a todos los recursos del proyecto mediante los siguientes recursos:

La URL de la aplicación desplegada:

https://client-tfg.vercel.app

En este repositorio se encuentra la aplicación de la parte del servidor

https://github.com/rmartin177/server-tfg

En este repositorio se encuentra la aplicación de la parte del cliente:

https://github.com/rmartin177/client-tfg

Palabras clave

scraping, Google Scholar, ERA Core, JCR, scopus, GGS, DBLP, datos biográficos, Node.js, React, Puppeteer

ABSTRACT

Bibliographic Data Extractor FLR

This project consists on the development of a web application where users are able to obtain researcher data from numerous institutional sites, academic resources and research documents in an orderly and redundancy-free way. This data is extracted through a scraping procedure, and the information is returned in a JSON format, accessible through a web interface, for further use.

The web application and all project resources can be accessed through the following resources:

The URL of the deployed application:

https://client-tfg.vercel.app/

In this repository the server-side application can be found: https://github.com/rmartin177/server-tfg

This repository contains the client-side application:

https://github.com/rmartin177/client-tfg

Keywords:scraping, Google Scholar, ERA Core, JCR, scopus, GGS, DBLP, biographical data, Node.js, React, Puppeteer

Índice de contenidos

Capítulo 1 - Introducción

En esta sección desarrollaremos la motivación del proyecto, los objetivos que tenemos que alcanzar y nuestro plan de trabajo.

1.1 Motivación

En muchas ocasiones un investigador académico necesita elaborar informes para sus proyectos recopilando una serie de datos sobre las publicaciones realizadas por los miembros del equipo de investigación. Esta recopilación puede incluir restricciones, como rangos de fecha o selección concreta de fuentes de distintas plataformas tales como *Scopus*, *Google Scholar*, *JCR*, etc. El proceso para elaborar informes y reunir esta información actualizada es lento, tedioso y con numerosas posibilidades de equivocarse debido a la gestión de toda esta cantidad de datos entrelazados.

El objetivo de este proyecto es crear una aplicación web que permita recopilar información bibliométrica de diversas fuentes de contenido académico, con filtros y flexibilidad de opciones. Esta aplicación permitirá ahorrar una cantidad considerable de tiempo en la elaboración de documentos y evitar los posibles errores en su elaboración manual. Para ello sustituimos la metodología tradicional por un proceso automatizado donde a partir de los nombres de los autores deseados se obtendrá toda la información requerida, convenientemente ordenada y clasificada.

1.2 Objetivos

El principal objetivo del proyecto es crear una aplicación capaz de obtener toda la información necesaria de los diferentes autores requeridos y generar un fichero con dicha información. En concreto tenemos las siguientes metas:

- Crear una aplicación funcional y escalable, ya que si en el futuro se decide añadir nuevas páginas de información, la incorporación de estas debe ser sencilla.
- Investigar sobre los diferentes tipos de *scraping* actuales para obtener la mejor solución a la hora de llevar a cabo el proyecto.
- Lograr automatizar el proceso al 100%, es decir, desde las tareas más sencillas hasta una sucesión de tareas complejas.
- Conseguir una disminución considerable de tiempo a la hora de recabar este tipo de información, mediante el uso de la aplicación.
- Implementar una versión local del proyecto mediante el uso de la consola.
- Conseguir una aplicación web (parte visual), para aumentar la comodidad del usuario.

1.3 Plan de trabajo

Para el desarrollo del proyecto se ha fijado una planificación de tal forma que se cubran las fases de análisis previo del proyecto, investigación de recursos, análisis de requisitos, implementación, testing y escritura de la memoria.

En la figura 1.1 se puede observar el diagrama de Gantt. Los plazos de entrega de las tareas y sus respectivos tiempos vienen detallados en dicho diagrama.

Las tareas desglosadas consisten en:

Análisis proyecto: Estudio previo de la viabilidad del proyecto para su posterior realización.

Investigación: Recogida de información acerca de las distintas técnicas y herramientas existentes para la obtención de información. Esto también incluye herramientas para el diseño de la aplicación.

Análisis de requisitos: Estudio sobre qué lenguajes de programación y herramientas vamos a usar para el desarrollo del proyecto.

Implementación: Puesta en marcha del proyecto. En esta etapa del desarrollo se implementaría la aplicación web con su respectiva base de datos.

Pruebas: Comprobación del funcionamiento de la aplicación y resolución de las posibles incidencias y errores.

Memoria: Documento donde se refleja el desarrollo y resultados del proyecto.

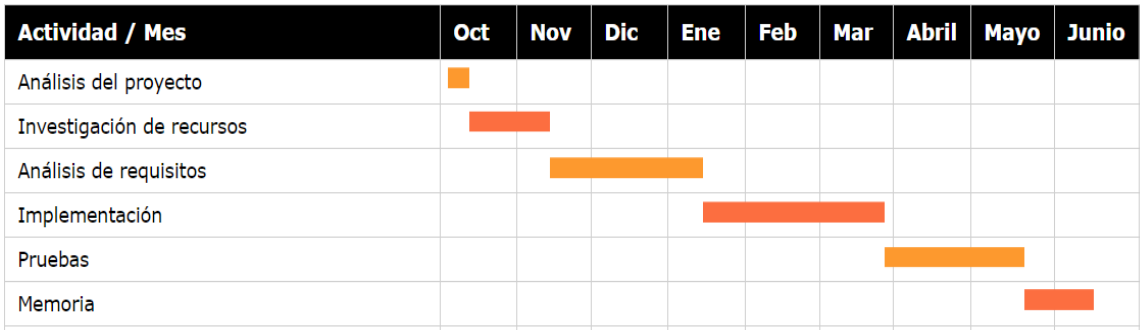


Diagrama de Gantt figura 1.1

1.4 Estructura de la memoria

El resto de la memoria se estructura como sigue:

- Preliminares: Presentación de las distintas herramientas y tecnologías que se han explorado en el proyecto.
- Arquitectura: Explicación de alto nivel de la estructura de la aplicación web y explicación de las dos componentes de dicha arquitectura.
- Despliegue y ejecución en local: Guía de uso de la aplicación.
- Contribuciones: Contribución detallada de cada integrante del proyecto.
- Conclusiones y trabajo futuro: Argumentación de los resultados obtenidos y el trabajo realizable para el futuro.

Capítulo 2 - Introduction

In this section we will develop the motivation of the project, the objectives we have to achieve and our future work plan.

2.1 Motivation

On many occasions an academic researcher needs to prepare reports for their projects by compiling a series of data on the publications made by the members of the research team. This compilation may include restrictions, such as date ranges such as Scopus, Google Scholar, JCR, etc. The process of reporting and gathering this up-to-date information is time consuming, tedious and it is likely to be mistaken due to the management of this amount of intertwined data.

The objective of this project is to create a web application that allows the collection of information from various sources of academic content, with filters and flexibility of options. This application will allow us to save a considerable amount of time in the elaboration of documents and avoid possible errors in their manual elaboration. To this end, we replace the traditional methodology with an automated process where by simply providing the name of the desired authors, all the required information is obtained in the desired order.

2.2 Objective

The main objective of the project is to create an application capable of obtaining all the necessary information from the different authors required and generate a file with this information. Specifically we have the following goals:

- Create a functional and scalable application, since if in the future it is decided to add new pages of information, their incorporation should be simple.
- To investigate the different types of current scraping to obtain the best solution to carry out the project.
- Achieve 100% automation of the process, that means, from the simplest tasks to a succession of complex tasks.
- Achieve a considerable reduction of time when collecting this type of information, through the use of the application.
- Implement a local version of the project by using the console.
- To achieve a web application (visual part), to increase the user's comfort.

2.3 Workplan

For the development of the project, a schedule has been established to cover the phases of pre-project analysis, resource investigation, requirements analysis, implementation, testing and the report.

Figure 2.1 shows the Gantt chart. The task deadlines and their respective times are shown in detail in the Gantt chart.

The tasks are broken down as follows:

Project analysis: Preliminary study of the feasibility of the project for its subsequent implementation.

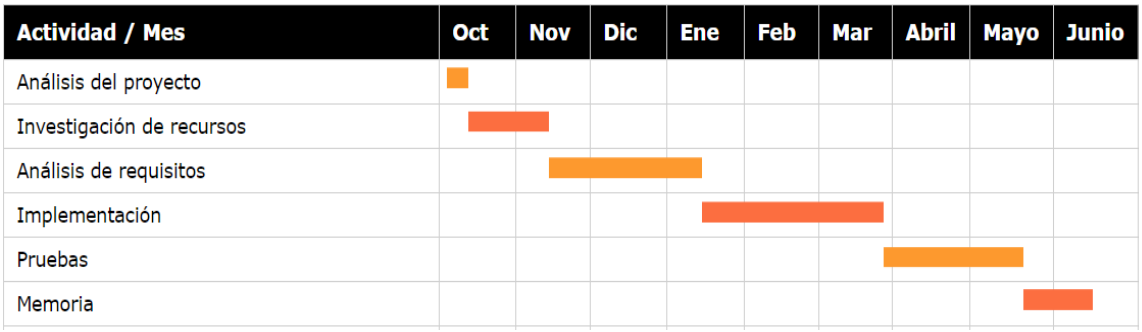
Research: Gathering of information about the different existing techniques and tools to obtain information. This also includes tools for the design of the application.

Requirements analysis: Study which programming languages and tools we are going to use for the development of the project.

Implementation: Implementation of the project. In this stage of the development the web application would be implemented with its respective database.

Tests: Checking the operation of the application and resolution of possible incidents and errors.

Report: Document where the development and results of the project are reflected.



Gantt chart figura 2.1

The rest of the report is structured as follows:

- Preliminaries: Presentation of the different tools and technologies that have been explored in the project.

- Architecture: High-level representation of the structure of the web application and explanation of the two components of this architecture.
- Deployment and execution in local: Guide to the use of the project.
- Contributions: Detailed contribution of each member of the application.
- Conclusions and future work: Argumentation of the results obtained and the work to be done in the future.

Capítulo 3 - Preliminares

En este capítulo presentaremos la idea principal del proyecto: scraping, las herramientas utilizadas y las webs para extracción de datos.

3.1 Scraping

El *scraping* [8] es el proceso automatizado de extraer información de páginas web, para su posterior procesamiento particular, simulando habitualmente la navegación de un usuario común en la web objetivo.

Debido a esto, se pueden recolectar datos de diferentes fuentes web de forma periódica y automatizada, procesarlos correctamente y usar esta información para la creación de nuevos servicios habitualmente analíticos. Muchas empresas de marketing y *big data* nacen de la aplicación de este proceso, siendo *Google* la principal referencia del sector en este ámbito para sus fines analíticos.

Entre los ejemplos más comunes de su utilización podemos encontrar la monitorización automatizada de precios, el análisis de competencia, el rastreo y protección de marca, o la creación de bots automáticos, entre otros.

La forma habitual de extraer la información es el análisis HTML y su interacción automatizada con la web, de donde podremos obtener información mostrada por la web aprovechando su estructura y composición.

3.2 Herramientas utilizadas

Antes de empezar con el desarrollo de la interfaz realizamos una profunda investigación para elegir la tecnología con la que se iba a desarrollar el proyecto. Para ello nos planteamos varias opciones que explicamos a continuación:

Uso de plantillas y CSS

La primera de las opciones con las que nos planteamos realizar el proyecto fue haciendo uso de tecnologías que ya dominamos, como es el uso de los denominadas plantillas.

Las plantillas son archivos en los cuales puedes crear una estructura a través de HTML pero cuentan con la ventaja de que puedes insertar pequeños trozos de código Javascript para hacer que tus sitios web sean dinámicos, es decir, que con una misma página web puedas hacer que se rendericen distintos datos que son pasados a través de variables. Algunos ejemplos de ello los podemos encontrar en tecnologías como *Ejs,Pug (Jade) o Flave* entre muchas.

Aunque esta tecnología la hemos aprendido en la asignatura de Aplicaciones Web de la carrera de Ingeniería del Software, finalmente decidimos desechar esta opción, ya que tanto a nivel personal como a nivel laboral hay opciones más interesantes y novedosas para abordar este tema.

Vue.js y Angular.js

La segunda de las opciones que investigamos fue hacer uso de *Vue.js* [5] o *Angular.js* [6], las cuales son unos entornos desarrollados en Javascript para el desarrollo de interfaces de usuario.

En primer lugar decidimos rechazar *Angular.js* debido a que está basado particularmente en una variante de Javascript llamada *TypeScrip*t, lo cual exigía un esfuerzo inicial elevado para familiarizarnos con esa variante y por eso fue desechada.

Vue.js y *React* [2,3] fueron las opciones que más valoramos para nuestro proyecto, ya que ambos están pensados para mostrar información sin apenas manipularla, por lo tanto ambas tecnologías son apropiadas para la aplicación en cuestión. Finalmente, nos decantamos por React debido a que ya teníamos planteados otros proyectos con esta herramienta. Además, al estar desarrollado por Facebook, la comunidad de desarrolladores que tiene detrás es muy grande, haciendo así que en caso de tener cualquier tipo de problema pudiéramos encontrar soluciones más fácilmente.

React.js

Como ya hemos explicado anteriormente, *React* es un *framework* desarrollado en *Javascript* y creado por Facebook. Con esta tecnología se pueden desarrollar interfaces para páginas web de manera fácil, intuitiva y reutilizable.

Para nuestro proyecto hemos elegido esta herramienta por varios motivos:

1) Encaja perfectamente con los requisitos del proyecto, ya que es una *Single-Page-Application,* es decir, no van a existir varias rutas.

2) En la parte cliente de nuestra aplicación no se hace una manipulación de datos, ya que de esa parte se encarga el cliente de la aplicación, es simplemente mostrar esos datos de forma ordenada, para lo cual la mejor opción es *React*.

3) El proyecto hará un consumo importante de *APIs* para la obtención de datos, y para esto *React* está optimizado.

4) Nos interesa conocer este *framework* debido a su potencial y a su gran demanda en el mercado laboral.

5) Cuenta con el soporte de una gran comunidad de desarrolladores.

Javascript

JavaScript es el lenguaje de programación encargado de conseguir cierto dinamismo en las páginas web. Este lenguaje se ejecuta directamente en el navegador, no necesita de un compilador para su ejecución. El navegador lee directamente el código, sin necesidad de software de terceros. Por tanto, se le reconoce como uno de los tres lenguajes nativos para el desarrollo web junto a *HTML* (encargado del contenido y estructura del sitio web) y a CSS (diseño o estilo de la web).

Node.js

Node.js [1] es un entorno de tiempo de ejecución de *JavaScript* (de ahí proviene su terminación en .js). Este entorno de tiempo de ejecución incluye todo lo que se necesita para ejecutar un programa escrito en *JavaScript*, se podría decir que es como un tipo de entorno de ejecución o intérprete.

Node.js fue creado por los desarrolladores de *JavaScript*. Transformaron este lenguaje de programación, algo que en un principio solo podía ejecutarse en el navegador, en algo que se podía ejecutar en cualquier ordenador. Gracias a la creación de *Node.js* se puede ir un paso más allá en la programación con *JavaScript*.

Puppeteer

Puppeteer [4] es una biblioteca de *Node.js* que proporciona una API de alto nivel para controlar *Chrome* o *Chromium* a través del protocolo *DevTools* de manera *headless*; lo que significa que interactuamos con *Chrome* sin la interfaz gráfica, es decir, sin un navegador como tal.

Elegimos *Puppeteer* frente a otras técnicas de *scraping* debido a diferentes opciones:

- El lenguaje de programación: para *Puppeteer* el lenguaje que se utiliza es *Javascript*, lenguaje de programación con el cual nos sentimos cómodos y hemos hecho nuestras últimas prácticas.
- Automatización: había otro tipo de biblioteca que también utilizaban *Javascript* (*Cheerio*), pero ninguna ofrece la automatización potente que nos ofrecía Puppeteer.
- Cancelación de CSS y animaciones: a la hora de ahorrar tiempo de ejecución Puppeteer ofrece una opción con la cual cancelar todos los CSS, imágenes y animaciones de las páginas web para que la carga de las mismas sea mínima.
- Curva de aprendizaje: la curva de aprendizaje era corta, la documentación estaba muy bien explicada y al ser desarrollada por Google había mucho apoyo vía tutoriales, foros, etc.

Google Firebase

Firebase [7] de *Google* es una plataforma en la nube para el desarrollo de la parte de servidor para aplicaciones tanto web como *mobile*. Está disponible para distintas plataformas con lo que facilita el desarrollo. Los servicios de los que disponen son variados, van desde bases de datos en vivo, hasta servicio de hostings y mantenimiento para todo tipo de aplicaciones.

Google Cloud Function

Google Cloud Functions es uno de los productos que puedes encontrar dentro de la *Google Cloud Platform*. Sirve para crear aplicaciones dentro de la infraestructura de *Google* que den respuesta a la demanda de eventos que puedan ocurrir en cualquier lugar. Lo bueno de este servicio es que pagarás únicamente por lo que utilices, es decir, el tiempo que tu código se esté ejecutando, de manera que es ideal para soluciones pequeñas.

Las características principales de *Google Functions* por las cuales hemos elegido esta plataforma son: su sencillez, su buena documentación en base a su desarrollo, pero sobre todo por la duración máxima de una petición (hasta nueve minutos). Si comparamos esta herramienta con, por ejemplo, *Heroku* (uno de sus competidores), esta apenas te proporciona treinta segundos máximos de petición, y en scraping esta es la característica principal que se debe tener en cuenta, ya que condiciona a las demás.

Como contábamos con experiencia previa en *Heroku*, optamos al inicio por desarrollar en esa plataforma, y según nos dimos cuenta que el scraping lanza peticiones con una duración realmente alta tuvimos que buscar alternativas y migrar el proceso. En este punto tuvimos problemas a la hora de revisar las opciones disponibles de despliegue ya que no hay muchos hostings que soporten scrapping de forma gratuita y con desarrollo en Puppeteer y *Node.js*, lo cual nos llevó a dos opciones finales: *Amazon AWS* o *Google Firebase*. No obstante, el proceso de iniciación en *Amazon AWS* es sensiblemente más complejo que *Firebase,* además no era suficiente el tiempo máximo por petición, lo cual nos llevó directamente a descartarla y elegir *Google Firebase*, donde desplegamos finalmente el servidor y la *API REST*.

Vercel

Vercel es una plataforma en la nube para sitios estáticos y funciones sin servidor (*serverless*) que se adapta perfectamente a cualquier flujo de trabajo. Permite a los desarrolladores alojar sitios web y servicios web que se implementan instantáneamente, escalan automáticamente y no requieren supervisión, todo sin configuración.

Evaluamos otras opciones, como *Netlify* que ofrece soluciones similares, pero la comodidad de poder desplegar en segundos con *Vercel* mediante *git*, su seguridad con *SSL*, rapidez de ejecución y el hecho de que sea la plataforma que más conocemos de usos anteriores decantó la balanza por *Vercel*, la cual creemos que será el hosting *serverless* referencia en el futuro.

3.3 Webs para extracción de datos

DBLP

El repositorio bibliográfico *DBLP* es la referencia en línea para la información bibliográfica de las principales publicaciones de informática. Ha pasado de ser un pequeño servidor web experimental a convertirse en un popular servicio de datos abiertos para toda la comunidad informática. La misión de *DBLP* es apoyar a los investigadores de ciencias de la computación en sus esfuerzos diarios proporcionando acceso gratuito a datos bibliográficos de alta calidad y enlaces a las ediciones electrónicas de las publicaciones.

DBLP ha pasado de ser un servidor a pequeña escala destinado a probar la tecnología web y a servir sólo a una pequeña comunidad local a ser un sitio web utilizado por miles de personas en todo el mundo.

CORE

La Computing Research and Education Association of Australasia, Core Inc., es una asociación sin ánimo de lucro de departamentos universitarios de informática de Australia y Nueva Zelanda.

La asociación tiene fines como ayudar y hacer avanzar la investigación en informática y tecnología de la información en la enseñanza superior y en los institutos de investigación, promover la enseñanza de la informática y la tecnología de la información en la educación superior o promover la cooperación y el enlace con otros

grupos y organizaciones que tengan propósitos y actividades relacionados o complementarios.

Las clasificaciones de las conferencias se determinan mediante una combinación de indicadores, entre los que se incluyen los índices de citación, los índices de presentación y aceptación de trabajos, y la visibilidad y el historial de investigación de las personas clave que organizan la conferencia y gestionan su programa técnico.

Se ofrece una clasificación con los siguientes posibles rangos: A*, A, B y C.

GGS

Las ponencias de los congresos son importantes para los informáticos. La evaluación de la investigación es importante para las universidades y los responsables políticos. Esta iniciativa está patrocinada por el *GII* (Grupo de Profesores Italianos de Ingeniería Informática), el *GRIN* (Grupo de Profesores Italianos de Informática) y la *SCIE* (Sociedad Española de Informática) formando el acrónimo de *GGS*. El objetivo de esta iniciativa es desarrollar una clasificación unificada de los congresos de informática.

Se pueden obtener 3 tipos de clases:

- Clase 1 (rating A++, A+) para conferencias catalogadas como excelentes
- Clase 2 (rating A, A-) para conferencias catalogadas como grandes eventos con alta calidad
- Clase 3 (rating B, B-) para conferencias catalogadas como eventos de buena calidad

Google Scholar

Google Académico es un buscador elaborado por *Google* especializado en la búsqueda de contenido y bibliografía científico-académica. El sitio indexa editoriales, bibliotecas, repositorios, bases de datos bibliográficas, entre otros; utiliza diferentes metodologías de medición. Las citas que calcula el número de citas totales que han tenido todas las publicaciones. El índice h que es el mayor número h, de forma que h publicaciones se han citado al menos h veces. Por último el índice i10 que recoge las publicaciones que han sido citadas al menos diez veces.

JCR

Journal Citation Reports agrega las conexiones significativas de las citas creadas por la comunidad investigadora a través de una gama de datos, métricas y análisis independientes de las editoriales de las revistas de mayor impacto del mundo incluidas en el *Science Citation Index Expanded* (SCIE) y el *Social Sciences Citation Index* (SSCI), parte de la Web of Science Core Collection.

Journal Citation Reports es el único informe de revistas de este tipo que es completo y selectivo desde el punto de vista editorial; contiene todos los datos necesarios para comprender los componentes que indexan el valor y el impacto de cada revista. Los datos estructurados son curados por un equipo global de expertos que evalúan y seleccionan continuamente las colecciones de revistas, libros y actas de conferencias incluidas en la *Web of Science Core Collection* para garantizar la precisión en la evaluación del impacto de las revistas.

Estos conocimientos de los expertos le permiten explorar los factores clave del valor de una revista, haciendo un mejor uso del amplio conjunto de datos y métricas disponibles en los *Journal Citation Reports*, incluido el Factor de Impacto de la Revista (JIF). que permite determinar de una manera sistemática y objetiva la importancia relativa de las principales revistas de investigación del mundo dentro de sus categorías temáticas.

Scopus

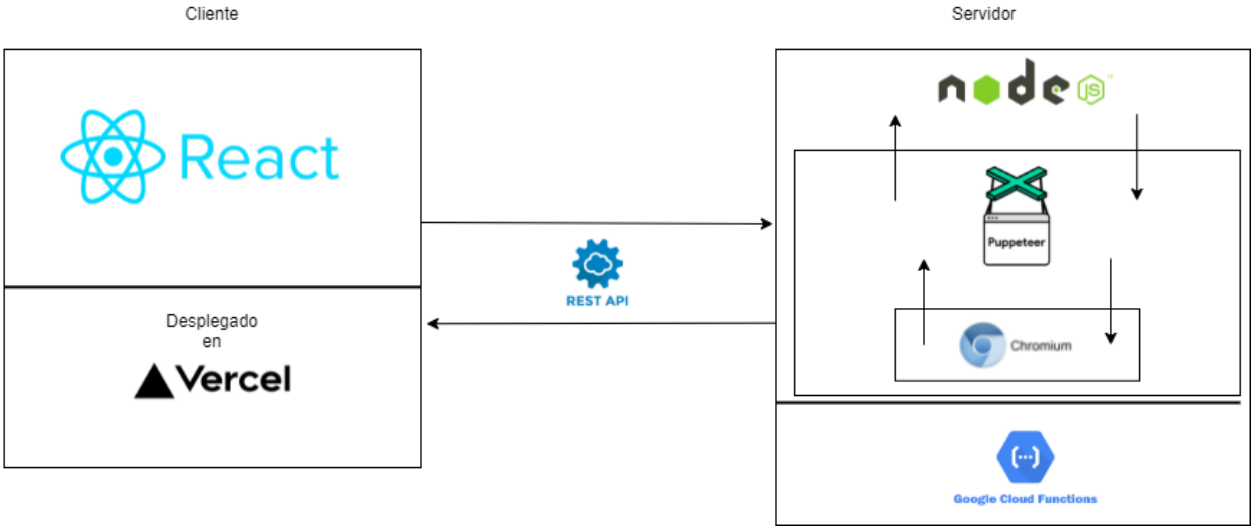
Scopus combina de forma única una base de datos de resúmenes y citas exhaustiva y curada por expertos con datos enriquecidos y literatura académica vinculada en una amplia variedad de disciplinas.

Scopus encuentra rápidamente investigaciones relevantes y autorizadas, identifica a los expertos y proporciona acceso a datos, métricas y herramientas analíticas fiables sobre el progreso de la investigación, la enseñanza o la dirección de la investigación y las prioridades, todo ello desde una base de datos y con una única cuenta.

En nuestro proyecto utilizamos *Scopus* para extraer el número de citas totales del autor y además obtener el índice-h explicado anteriormente en *Google Scholar*.

Capítulo 4 - Arquitectura

En esta sección vamos a explicar la estructura del proyecto, que está dividida en dos componentes: el cliente y el servidor.



Representación gráfica de la arquitectura del proyecto. Figura 4.1

La arquitectura del proyecto está formada por dos bloques: cliente y servidor, como se puede observar en la figura 4.1. El bloque de cliente está formado por *React*, el cual se encarga de la parte visual y el envío de peticiones, además todo este código está alojado en la plataforma *Vercel*. Por otro lado tenemos el bloque del servidor donde el eje central de este es *Node.js* que es el encargado de comunicarse con *Puppeteer* que a su vez se comunica con *Chromium*. Las solicitudes a la parte de Cliente están alojadas en *Google Cloud Functions*.

4.1 Cliente

En esta ocasión vamos a hablar sobre la parte cliente del proyecto, de cómo se ha implementado además de una pequeña guía de uso.

4.1.1 Implementación

En esta sección hablaremos sobre la estructura del proyecto, qué partes lo conforman y cómo lo hemos planteado.

- **Formulario**

Para que el usuario sea capaz de introducir los datos hemos creado una página principal para recoger dicha información. El formulario está dividido en varias partes que explicaremos a continuación:

  - -User/Password para acceder a JCR y/o Scopus: Esta sección se muestra y se oculta automáticamente en función de si se han seleccionado los filtros de *JCR y Scopus.*
  - -Start Year/ End Year: En esta sección hemos usado dos *sliders* inspirados en *Material Design para seleccionar*….
  - -Filtros: Esta sección ha sido desarrollada con el uso de *checkbox* que hemos podido personalizar a nuestro gusto, además lleva una capa lógica en *Javascript* para hacer el efecto de apagado cuando un usuario hace *click*.
  - -Botón Add Author: Este botón también hace uso de una capa lógica de *Javascript* para poder posicionar el nuevo elemento en la posición deseada dentro del DOM.

- **Homónimos**

Cuando realizamos una búsqueda de un autor y resulta que al hacer el análisis hemos detectado que existen varias personas con el mismo nombre, damos al usuario la opción de elegir sobre cuáles desea que se obtenga información.

- **Proceso de carga**

Para mostrar que estamos realizando la consulta solicitada por el usuario, se ha decidido mostrar un *spinner* de carga que se cerrará automáticamente una vez que el proceso de búsqueda de información haya concluido.

- **Resultado de la búsqueda**

Una vez que todos los campos están correctos (inclusive la elección de homónimos, si los hubiera) se procederá a mostrar el resultado obtenido en formato tabla.

El resultado se muestra en una pantalla que contiene las siguientes partes:

  - Información filtrada: la información solicitada por el usuario es mostrada de forma ordenada mediante una tabla con los distintos campos seleccionados por el usuario.
  - Información de filtros seleccionados: para recordar al usuario el acceso a los filtros que ha elegido, hemos insertado en la parte superior de la tabla.
  - Búsqueda: otro de los elementos incorporados para facilidad del usuario ha sido la adición de un campo de búsqueda mediante el cual se filtran los resultados obtenidos. La búsqueda en autores se realiza por nombre y la búsqueda en las publicaciones se realiza mediante el título.
  - Paginación: Todos los resultados incluyen paginación para poder moverse fácilmente entre los datos, además la tabla contiene un pequeño campo para cambiar el número de entradas que se desean mostrar.
  - Pestañas: contamos con 3 ventanas y un sistema de pestañas para poder moverse entre ellas situado en la parte superior de la aplicación.

- **Descarga de datos**

Otra de las características que hemos incorporado en la aplicación, es la posibilidad de descargar la información filtrada mediante un archivo *JSON*, este archivo en un principio se guardaba en la aplicación para su posterior descarga, pero realizamos una mejora para que se formase dinámicamente, procedemos a dar una explicación de nuestro archivo *JSON*.

Formato del JSON

Para mostrar la información del usuario hemos decidido mostrarlo mediante un *JSON* divido en 3 partes (authors, publications y errors) en cual procederemos a explicar. Cabe destacar que para la extracción de datos en las diferentes clasificaciones hemos seleccionado la clasificación del año más próximo sin exceder el año de publicación.

Formato del apartado authors:

```
"authors":[
    {
      "name":"Alberto Verdejo",
      "indices":{
        "indice_h_total google scholar": 16,
        "indice_h_5_years_google_scholar": 7,
        "indice_i10_total_google_scholar": 26,
```

```
    "indice_i10_5_years_google_scholar": 6,
    "índice h total scopus": 9        },
  "citas":{
    "citas total google scholar": 995,
    "citas_total_5_years google scholar": 211,
    "citas_total_scopus":"364 Citations by 231 documents"
  },
  "jcr":{
    "numero publicaciones q1": 0,
    "numero publicaciones q2": 2,
    "numero publicaciones q3": 0,
    "numero publicaciones q4": 5
  },
  "ggs":{
    "numero publicaciones class": 1.0,
    "numero publicaciones class": 2.0,
    "numero publicaciones class": 3.9
  },
  "core":{
    "numero publicaciones _AA": 0,
    "numero publicaciones A": 4,
    "numero publicaciones B": 1,
    "numero publicaciones C": 2
```

```
      }
    }
  ],
```

Lo primero que se muestra es el nombre del autor del cual se ha realizado la búsqueda. Después se muestran diferentes índices que han sido extraídos de *Google Scholar* y *Scopus* (totales, 1 año y 5 años), a continuación se muestra el conteo de las citas totales de Google Scholar y Scopus y de los últimos 5 años de *Google Scholar.*

Después de la información extraída de *Google Scholar* y Scopus mostramos los datos de las siguientes páginas:

JCR, *GGS* y *Core* : la información se divide en diferentes grupos de calidad, los cuales calculan las propias clasificaciones *JCR*, *GGS* y *Core*. Estos grupos muestran la cantidad de artículos que pertenecen a dichos grupos.

Formato del apartado *publications*:
Hay tres tipos de publicaciones con la mayoría de datos similares pero con sus diferencias.

Formato de una entrada articles:

```
"publications":[
    {
      "authors":[
```

```
        "Óscar Martín",
        "Alberto Verdejo",
        "Narciso Martí-Oliet"
      ],
      "citas":{
        "número citas google scholar": 3,
        "número citas scopus": 0
      },
      "type":"Articles",
      "title":"Compositional Specification in Rewriting Logic.",
      "pages":"44-98",
      "year":"2020",
      "volume":"20",
      "journal":"Theory Pract. Log. Program.",
      "issue":"1",
      "jcr":{
        "categoría":"LOGIC",
        "impact_factor": 1.076,
        "position": 84/108,
        "quartile":"Q4"
      }
    },
```

En primer lugar se muestran los autores que participan en el artículo. Después se muestra el número de citas de *Google Scholar* y *Scopus*. A continuación se muestran datos del propio artículo extraídos de DBLP (Type, title, pages, year, volume, journal, issue). Por último se muestra la información extraída de *JCR* en un objeto. Este objeto contiene la categoría en la cual el artículo tiene una puntuación más alta, el factor de impacto, la posición y el cuartil.

Formato de una entrada *inproceedings*:


```
"authors":[
        "Óscar Martín",
        "Alberto Verdejo",
        "Narciso Martí-Oliet"
    ],
    "citas":{
      "número citas google scholar": 7,
      "número citas scopus": 2
    },
    "type":"Inproceedings",
    "title":"Synchronous Products of Rewrite Systems.",
    "pages":"141-156",
    "year":"2016",
    "acronym":"ATVA",
    "url":"db/conf/atva/atva2016.html#MartinVM16",
```

```
    "book_title":"Automated Technology for Verification and Analysis - 14th
International Symposium, {ATVA} 2016, Chiba, Japan, October 17-20, 2016,
Proceedings",
    "ggs":{
      "year":2015,
      "class":3
    },
    "core":{
      "core_year": 2014,
      "core_category":"A"
    }
  },
```


En la publicación de tipo *inproceedings* se le añaden diferentes campos, el Acronym el cual se extrae de *DBLP*. Después se extraen los datos tanto de *GGS* como de *Core*, del primero se extrae el año de publicación y la clase estipulada por *GGS*, por otro lado de *Core* se extrae el año de la publicación y la categoría asignada por dicha clasificación.

Formato de una entrada *Incollection*:


```
"authors":[
    "Manuel Clavel",
    "Francisco Durán",
```

```
      "Steven Eker",
      "Patrick Lincoln",
      "Narciso Martí-Oliet",
      "José Meseguer",
      "Carolyn L. Talcott",
      "Miguel Palomino",
      "Alberto Verdejo"
    ],
    "citas":{
      "número citas google scholar": 9,
      "número citas scopus": 3
    },
    "type":"Incollection",
    "title":"Playing with Maude.",
    "pages":"159-184",
    "year": 2007,
    "acronym":"All About Maude",
    "url":"db/conf/maude/maude2007.html#ClavelDELMMTPV07"
  },
```


Para las Incollection la información mostrada es estándar, lo único que ha sido extraído fuera de *DBLP* han sido las citas de *Google Scholar* y *Scopus*.

Formato del apartado *errors*:

```
"errors":[
   [
      "Ha ocurrido un error con JCR en el artículo undefined con nombre de revista:
 Journal of Algorithms cuyo autor es: Alberto Verdejo
 link:https://dblp.org/db/journals/jal/jal62.html#Marti-OlietPV07"
   ]
],
```

Por último, hemos añadido una sección de errores, en la cual se muestra información acerca de los errores que suceden a la hora de ejecutar la aplicación.

- **Tratamiento de errores en el proceso**

Para los errores que hayan sucedido al realizar el procesamiento de la información, hemos desarrollado una pestaña llamada errores que mostrará si ha ocurrido algún inconveniente durante la ejecución.

Para facilitar la visualización de errores, hemos hecho que la pestaña cambie automáticamente de color en función de si hay o no errores.

4.2 Servidor

Antes de poder procesar cualquier petición, hay que hacer unas configuraciones previas de cara a optimizar y evitar errores en su procesamiento. Durante el lanzamiento del servidor se inicia la instancia del navegador interno que se usará para el *scraping* chromium aportado en el framework de *Puppeteer* de tal manera que estará listo para cualquier petición y siempre se usará esta instancia gracias al patrón singleton aplicado.

Desde el punto de configuración, el servidor está listo para responder a cualquier petición recibida.

Cuando procesa una petición primero hace un ejercicio de optimización de la ventana de trabajo interna del navegador, podemos pensar en Puppeteer como un framework que emula totalmente el navegador que tiene cualquier usuario en su ordenador habitual. Esta optimización nos ayudará a eliminar todos los pesos innecesarios en cada web de la cual visitamos para realizar su extracción de datos, tales como las imágenes, fuentes de letras y la correcta configuración de la resolución de pantalla para lograr que no haya cambios en este punto que puedan alterar el correcto funcionamiento de la extracción de datos debido a un diseño *responsive*.

4.2.1 Flujo del back-end

El servidor recibe los datos de los autores enviados por la interfaz web hacia nuestro end-point, donde son almacenados y procesados en orden para crear el *JSON* de respuesta en función de los filtros proporcionados.

Se empieza el *scraping* en función de los filtros, los cuales explicaremos en detalle a continuación.

4.2.2 Scraping de DBLP

El scraping de *DBLP* es un filtro permanente ya que es la fuente de datos fundamental de nuestra aplicación, sobre ella extraemos en una primera iteración la información relativa a los posibles autores con homónimos que no se haya especificado correctamente en la interfaz, por ejemplo si hay varias personas con el mismo nombre, la aplicación detectará esto y se ofrecerán las opciones al usuario para seleccionar al autor correcto.

Tras ese paso, se procede a realizar el proceso automático donde encontramos el archivo *XML* en *DBLP* con la información relativa al autor proporcionado por el usuario, donde están listadas sus publicaciones (articles, incollections e inproceedings).Dependiendo de los selectores y del tipo de publicación se extraen los siguientes datos:

i. Para las incollections se extrae: authors, type, title, pages, year, acronym, url.
ii. Para las inproceedings se extrae: authors, type, title, pages, year, acronym, url.
iii. Para los articles se extrae: authors, type, title, pages, year, volume, journal, issue, url.

4.2.3 Procesamiento de la clasificación GGS

La propia web The GII-GRIN-SCIE (GGS) Conference Rating - Search the GGS Rating 2018 proporciona los datos en formato csv según el año. Para procesarla, hemos

obtenido esos archivos de la web, los hemos saneado de tal manera que dejemos un formato *CSV* con las columnas útiles y después hemos utilizado la biblioteca "convert-excel-to-json" para transformar el *CSV* a *JSON*.
Este proceso lo realizamos para cada año disponible en GGS y dejamos cada uno en un formato JSON depurado y limpio para consultar como si de una base de datos se tratase. Como cada archivo es muy extenso, para optimizar considerablemente el tiempo de acceso aplicamos el algoritmo de divide y vencerás previa ordenación de cada archivo *JSON* de forma alfabética. La lógica es bastante simple en este punto, con el año y acrónimo obtenido previamente en el *XML*, usamos filtros para encontrar el archivo correcto y los datos requeridos.

4.2.4 Scraping de la clasificación CORE

El scraping del Core en http://portal.core.edu.au/conf-ranks es una parte compleja, donde la optimización de reducción de peso suprimiendo imágenes ha conseguido reducir en un alto porcentaje el tiempo de ejecución siendo de cualquier manera elevado por el alto número de pasos automatizados para cada artículo. Aquí se han tenido que aplicar todos los recursos de scraping disponibles como parametrizar las URLs de donde extraer la información para ir directamente al acrónimo del artículo en cuestión, esto evita un paso de unos 2 segundos de ejecución previo de búsqueda (esto multiplicado por todos los artículos es fácil que haga una media de 1-2 minutos reales de tiempo ahorrado al procesamiento de cada petición).

Para realizar esta sección necesitamos contar con el acrónimo y el año previamente extraído en *DBLP*, con estos datos se busca directamente el acrónimo parametrizado en la *URL* y obtenemos la información de los selectores (complejos una vez más, con

pocas clases de *CSS* donde guiarse y similares) para filtrar la información que nos interesa en función del año.

Nos dimos cuenta de que era bastante habitual que varios artículos coincidieran en el acrónimo, y dado que es un punto realmente lento de procesar, el crear una estructura de datos dinámica para almacenar cada acrónimo y guardar su información para usarlo posteriormente ha reducido considerablemente el tiempo de procesamiento.

En este punto exploramos distintas alternativas sobre usar una base de datos para si el acrónimo es nuevo haga el proceso de scraping y si lo tenemos almacenado en la *BBDD* solo hacemos una consulta. En teoría esto nos hubiese proporcionado una rapidez todavía mayor pero la incertidumbre del tipo de información a obtener (dinámica cada año) nos hizo descartar debido a que en el momento que se actualice el Core tendríamos la base de datos desactualizada, siendo en los artículos nuevos incorrecta, por lo que decidimos mantener la solución actual con el almacenamiento de los datos de forma dinámica.

4.2.5 Scraping de Google Scholar

El scraping de *Google Scholar* se hace desde la siguiente web: https://scholar.google.es/. Después de investigarla nos dimos cuenta que no hay ninguna forma de extraer toda la información del autor en un único archivo y por lo tanto hay que hacer un scrapping total de la propia web. Para ello introducimos el nombre del autor en el buscador y seguidamente clicamos en la primera página que aparece después de dicha búsqueda, que es la del perfil del propio autor. El primer problema que aparece es que los artículos solo se mostraban de 20 en 20 y hubo que

crear un bucle para desplegar todos los artículos ya que con un solo clic podemos obtener solo 40 de estos. Después de realizar dicho bucle obtenemos todos los nombres y números de citas de los artículos ayudándonos de un desplazamiento vertical , pues hasta que no se implementó daba problemas a la hora de recoger todos los artículos y sus citas Además, observamos que *Google* Scholar tenía en su base de datos más artículos de los que los propios autores habían publicado, ante este problema decidimos recoger todos los artículos en un objeto y realizar una búsqueda binaria en la que se conservaban unicamente los artículos recogidos en *DBLP*. Después de esto mediante un scraping mucho más sencillo obtenemos el número total de citas y el número de citas en los últimos 5 años proporcionado por *Google Scholar*.

4.2.6 Scraping de JCR

El scraping de *JCR* se realiza desde la siguiente web: http://jcr-incites.fecyt.es/. Esta página ha sido la que más problemas ha dado; los detallamos a continuación. En primer lugar nos reunimos con los tutores para concretar cómo iba a ser la búsqueda: para encontrar los datos en esta web se debe realizar una consulta por cada revista, y se decidió que se buscaría por el nombre completo de la revista. Más adelante, al intentar implementarla, nos dimos cuenta de que había algunas revistas que tenían nombres muy parecidos y que *JCR* devolvió muchos resultados diferentes, y después de otra reunión se decidió utilizar el *ISSN* (International Standard Serial Number es un número de serie de ocho dígitos que se utiliza para identificar de forma exclusiva una publicación) de cada revista. Volvimos a encontrar problemas al utilizar este método de búsqueda, ya que las revistas más actuales no tienen *ISSN*, por lo cual se decidió que se harían ambas comprobaciones, la primera con el *ISSN* (el cual era el método

más eficaz) y si este no tenía efecto se comprobaría con el nombre. Sin embargo, aun después de manejar esta casuística aparecieron varios errores:

i. Revistas sin *JCR*: había ciertas revistas las cuales no contaban con una valoración *JCR*, y aunque es un número muy pequeño esto ralentiza la app ya que hay que hacer las dos comprobaciones de todos modos.
ii. Revistas con mismo *ISSN*: aunque el *ISSN* es un número único de cada revista, *JCR* en determinados casos creaba páginas duplicadas de la misma revista para diferentes años. Por ejemplo, la revista Journal of Algorithms tiene dos páginas en *JCR*: una desde el 2000 al 2018 y otra desde 2019 a 2020. Esto provocó un problema ya que el buscador te redirige a una página intermedia la cual dejaba estancado el scraping.

Una vez solventados todos estos problemas, continuamos con la parte de extraer los datos de la página de cada revista.. Concretamente, se necesitaban dos tablas las cuales traíamos al servidor completas para su posterior análisis. De estas tablas necesitamos solo la fila del año más próximo sin pasarse del año de publicación del artículo, y una vez encontrábamos esa fila en la primera tabla extraemos su factor de impacto. Para la segunda tabla necesitábamos la fila con los mismos términos que en la tabla anterior y además las categorías que se encontraban en ciertas columnas, para ello se diseñó un algoritmo que selecciona la categoría con más índice de impacto y la recoge en el objeto que se mostraría en el *JSON*.

4.2.7 Scraping de Scopus

El scraping de *Scopus* se realiza desde la siguiente web: https://www.scopus.com/home.uri, desde el principio tuvimos que reunirnos ya que

observamos un fallo grave que ocurría con la mayoría de los autores:a la hora de buscar un autor por nombre aparecen varias páginas de autores que empezasen por las mismas letras, mismos apellidos, etc. Después de una reunión con los tutores se decidió que lo más adecuado sería hacer una búsqueda por *ORCID* (número único para cada autor), el cual poseen la mayoría de estos. Después de definir la táctica con la que íbamos a extraer datos surgió el problema de que al necesitar autenticación, al igual que en JCR, había problemas al extraer los datos en una misma consulta, la forma de solventar dicho fallo fue mediante la realización de dos códigos a la hora de loguearse, uno en el cual se realizaría todo desde cero y un segundo caso en el cual nos saltaremos el paso del login. A continuación conseguimos encontrar la forma de que *Scopus* nos facilitase un documento del cual se puede coger tanto titulo como número de citas.

Una vez realizado el scraping para cada artículo obtenemos el número total de citas en scopus el cual también añadiremos al *JSON*.

Capítulo 5 - Guía de uso y despliegue

Para realizar la parte del despliegue se ha requerido numerosas horas de investigación sobre las mejores plataformas y condiciones que podían soportar las características dadas por el conjunto de la plataforma en la red.

Para esto desglosamos el despliegue en dos puntos principales, el despliegue web y el local, ya que por temas de capacidad máxima de respuesta del servidor alojado es posible que se realicen peticiones que sobrepase el tiempo máximo del servidor en *Firebase*, por lo que estas deberán realizarse en un servidor local sin máximo de tiempo.

A continuación procedemos a explicar las características de cada punto.

5.1 Despliegue web

Debido a que la aplicación consta de tecnologías distintas tanto para la parte de cliente como para parte servidor, hemos usado dos plataformas diferentes. Para el despliegue del cliente *Vercel* y para la del servidor *Google Firebase*.

5.1.1 Cliente

El despliegue del cliente se ha realizado desde el principio sobre la plataforma de Vercel, la cual es un host para aplicaciones *serverless* desarrolladas en *React/Next.js*.

Esto se adapta a la perfección a nuestro proyecto, un host que proporciona un *SSL* gratuito y seguro, con una velocidad de respuesta notable y sin ningún tipo de inversión previa, el único punto negativo es que trabajar en grupo dentro de la

plataforma está limitado a una versión de pago, pero se pudo solventar configurando los ajustes de despliegue via sincronización con *git*. De esta manera, pese a no tener una versión de pago, todo el equipo de desarrollo puede tener acceso a la rama master y en cualquier momento podrá desplegar con un commit.

5.1.2 Servidor

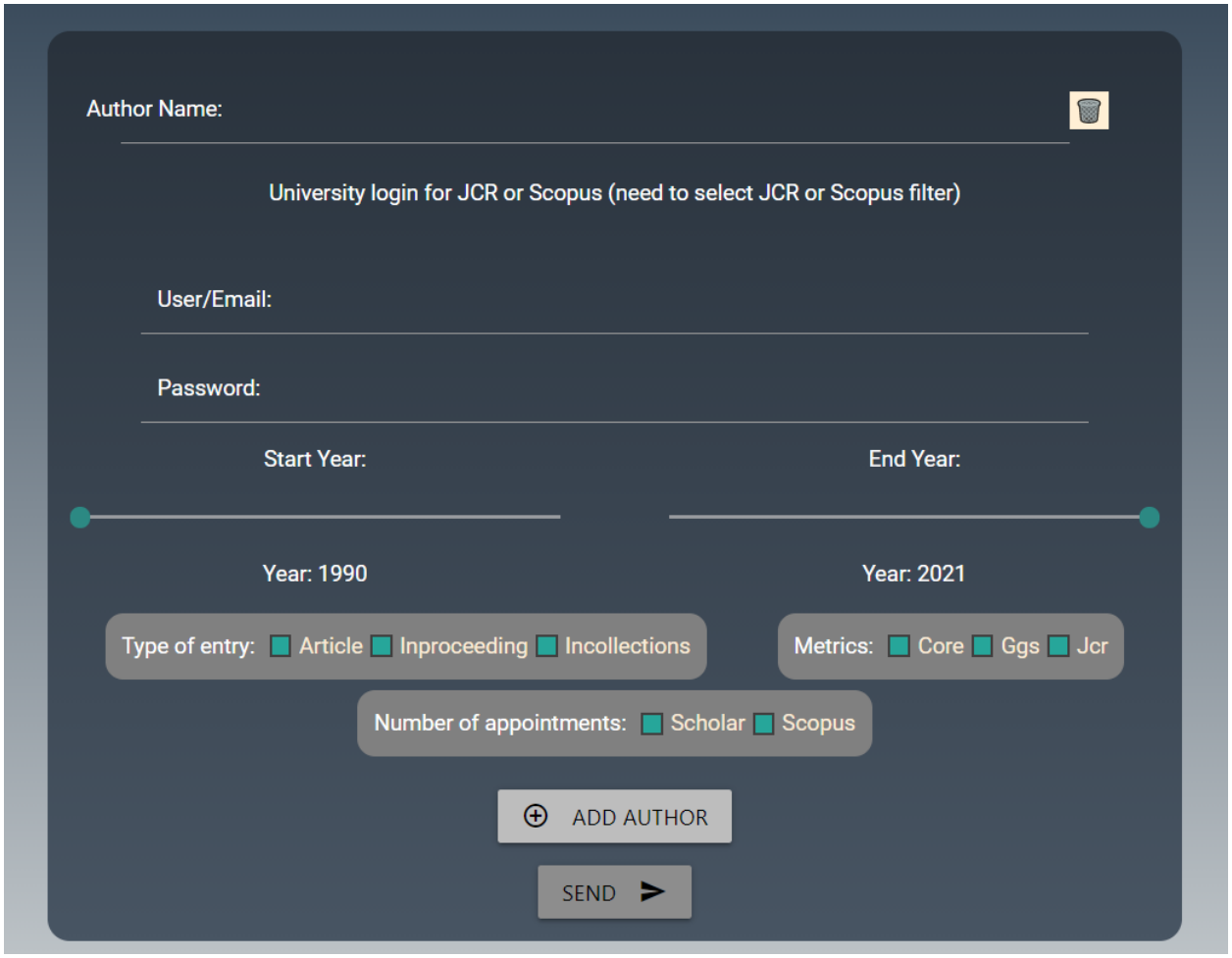
El despliegue del servidor está realizado sobre el *hosting* de *Firebase*, más concretamente sobre su opción de *Google Functions*. Esta herramienta es de pago pero basa el importe en la demanda de peticiones. Al ser un proyecto para un sector muy específico como son los investigadores con publicaciones académicas,el coste estimado anual es inferior a un euro, por lo que no lo contemplamos como un impedimento para su desarrollo.

5.1.3 Guia de uso cliente

En esta sección haremos una pequeña guía de uso del programa y de todas las partes que conforman dicha interfaz.

- **Formulario**

El primer paso que debe realizar un usuario para usar nuestra aplicación es ingresar el autor sobre el que quiere realizar las búsquedas en el formulario mostrado en la figura 4.2.

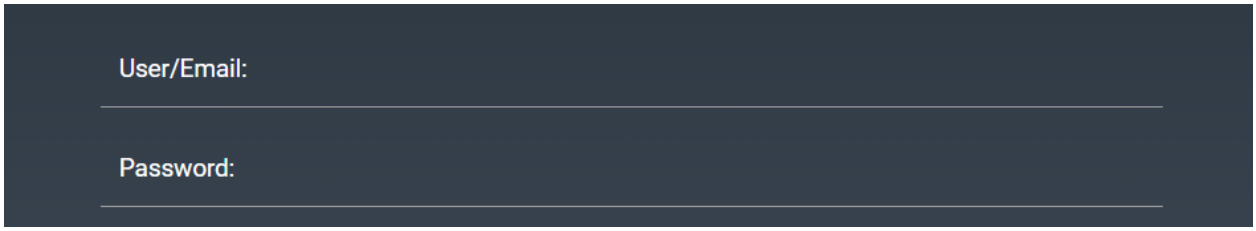


Página inicial de la aplicación web figura 4.2

Como podemos ver el formulario está dividido en varias partes que explicaremos a continuación:

- User/Password para *JCR* y/o *Scopus* (figura 4.3): en esta sección se pide la autenticación necesaria para poder buscar información dentro de *JCR* y *Scopus*, ya que son páginas en las que necesariamente debes haber iniciado sesión. Dicha información no se almacena en ningún momento, una vez que se accede a la página

es desechada.

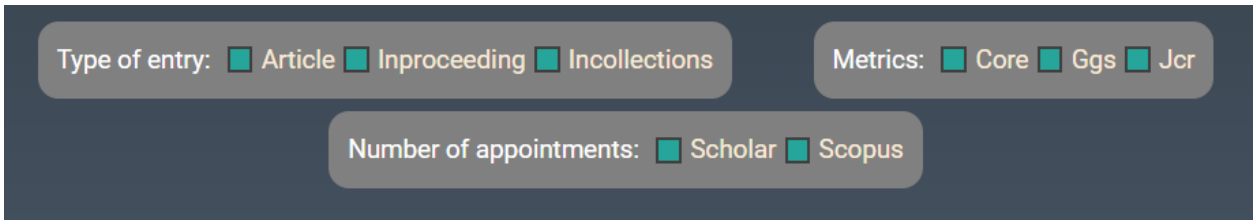


Formulario autenticación figura 4.3

- Start Year/ End Year (figura 4.4): En esta sección se filtran los resultados por años.

Filtros por años figura 4.4

-Filtros (figura 4.5): En esta sección se pueden seleccionar los filtros para evitar buscar información que no sea del interés del usuario. Para facilidad y comodidad del usuario, todos los filtros vienen activados por defecto.



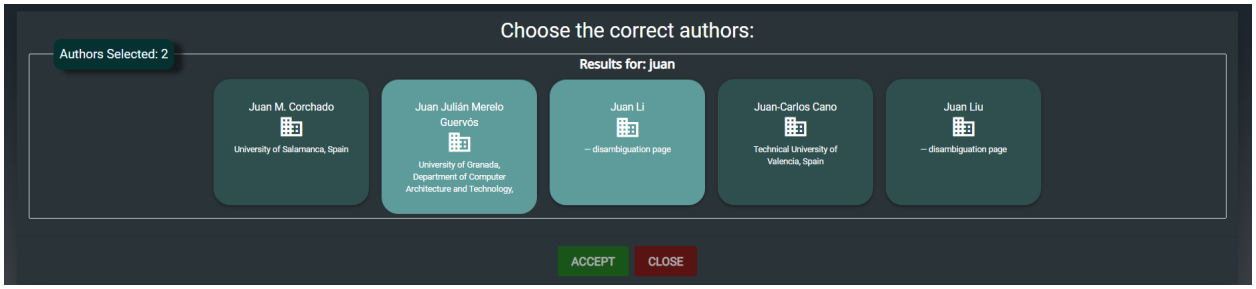
Filtros de selección de información figura 4.5

-Botón Add Author: Con este botón podemos generar tantas filas para realizar la búsqueda de tantos autores como queramos (ver figura 4.6)

Botones para añadir autor Figura 4.6

- **Homónimos**

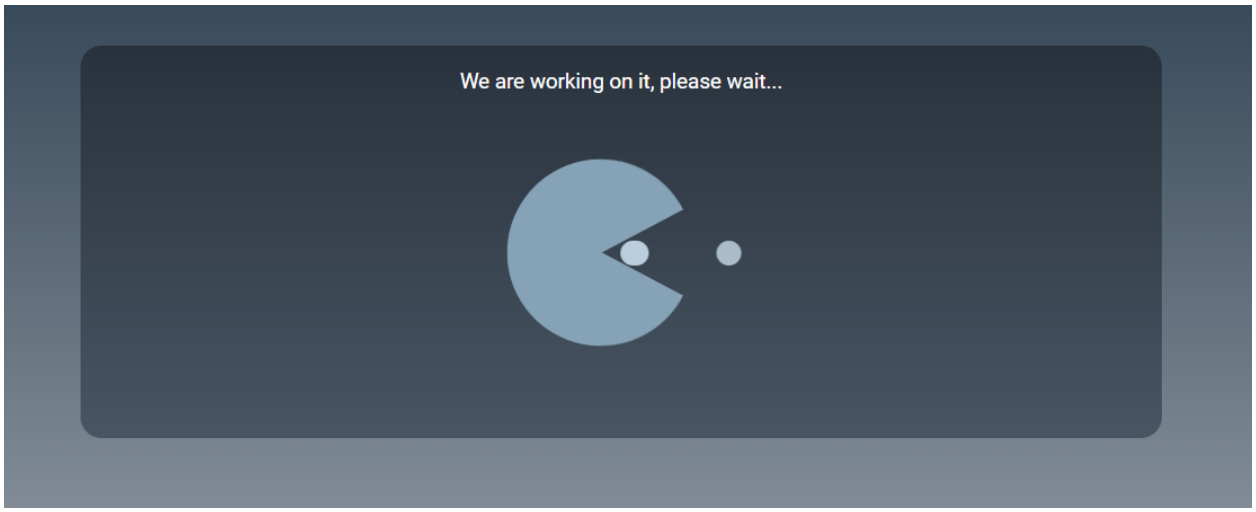
Cuando hay varios autores con el mismo nombre, el usuario podrá elegir el que desee, como podemos ver en la figura 4.7.



Ventana de selección homónimos figura 4.7

- **Proceso de carga**

Para mostrar que estamos realizando la consulta solicitada por el usuario, se ha decidido mostrar un *spinner* de carga (figura 4.8)

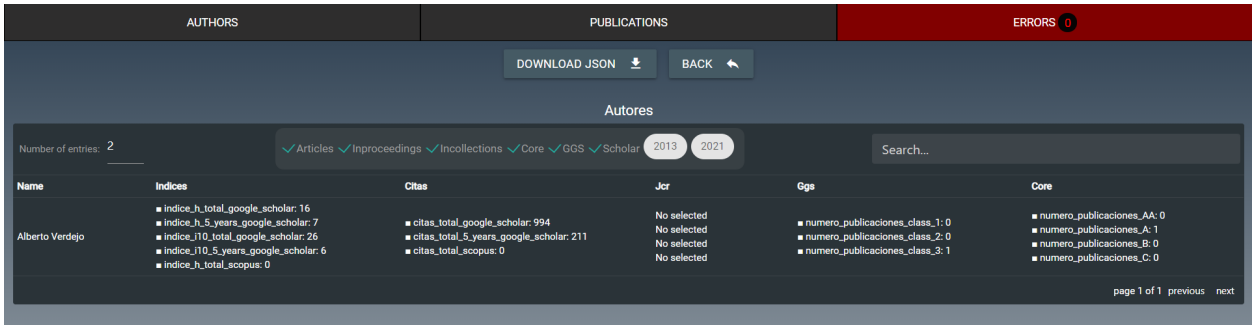


Spinner de carga figura 4.8

- **Resultado de la búsqueda**

Como podemos observar en la figura 4.9, los datos obtenidos son mostrados en una tabla y separados en dos secciones, la sección de datos que están relacionados con el autor en cuestión, y la sección de las publicaciones obtenidas de dicho autor.

En la tabla tendremos una sección que para facilidad del usuario indicará los filtros que se han seleccionado. Además tendremos un buscador, y por último, podremos cambiar el número de resultados que se muestran en la tabla.



Ventana de procesamiento de datos Figura 4.9

5.2 El despliegue local

Consiste en ejecutar cada parte de la aplicación (cliente y servidor) en puertos locales, en este caso hemos configurado el puerto 3000 para la parte del cliente y el puerto 4000 para la parte del back-end, el proceso es bastante simple siguiendo estos pasos:

1. Descargar el código del cliente y servidor desde sus respectivos repositorios.
2. Desde la consola con la dependencia de Node.js instalada (https://nodejs.org/es/), procedemos a abrir el proyecto del cliente y ejecutamos `npm install` para descargar todas las dependencias (este paso solo se tiene que ejecutar la primera vez que estemos lanzando la aplicación, para futuras ocasiones las dependencias ya se encontraran descargadas).
3. Usamos el comando `npm start` que lanzará la aplicación web sobre el puerto 3000.

4. Mismo procedimiento que en el paso 2 pero esta vez con el código del back-end. Sin cerrar la ejecución del cliente, abrimos otra ventana en la terminal de npm y procedemos a hacer npm install y bajamos las dependencias, siendo solo necesario ejecutarlo la primera vez.
5. Ahora en la terminal usamos el comando `npm run dev` y lanzamos el servidor sobre el puerto 4000.
6. Accediendo a http://localhost:3000/ tienes la aplicación desplegada en local lista para usar.

Una alternativa es usarla como *API REST* sin capa de cliente desde, por ejemplo, postman (extensión del navegador), realizamos solo los pasos 4-5, una vez eso, sobre el end-point (URL la cual recibe la petición y envía una respuesta) lanzaremos una petición POST con un objeto *JSON* con una composición como la que aparece en el ejemplo, pudiendo alterar los filtros con valores booleanos a conveniencia, de igual manera para el número de autores a extraer información. Para realizar peticiones a la API REST se deberá utilizar el enlace DBLP para cada usuario en lugar del nombre.

En este objeto JSON tendremos un primer campo "authors" que será una lista con los autores sobre los que se quiere extraer información. Cada autor se enviará con un formato de objeto dentro del propio array con dos campos, el autor en el campo "author" y el link a su perfil de DBLP con el campo "link".

Tras la lista de autores aparecerán los filtros (el nombre del campo a usar en el *JSON* es "filters") donde podrás elegir con booleanos los siguientes campos:

Campos referidos al tipo de artículo:

- "checkInproceedings": permite obtener las entradas de tipo *inproceedings* del autor
- "checkArticles": permite obtener los articles del autor.
- "checkIncollections": permite obtener las incollections del autor.

Campos sobre las páginas web que se usarán para a extraer información del autor y de sus publicaciones

- "checkSchoolar": permite obtener datos de Google Scholar del autor.
- "checkGGS": permite obtener datos de la clasificación GGS del autor.
- "checkCore": permite obtener datos de la clasificación Core sobre el autor.
- "checkJRC": permite obtener datos de la clasificación JRC sobre el autor.
- "checkScopus": permite obtener datos de Scopus sobre el autor.

El usuario institucional y la contraseña (en nuestro caso particular usamos los de la UCM) solo son necesarios si tienes el filtro de Scopus o JCR activados, de ser así tendrás que rellenarlos en los siguientes campos con cadenas de texto:

- "mail": email que uses para loguearte en tu institución.
- "pass": contraseña de esta.

También podrás filtrar por periodos de tiempo, esta vez se definirán mediante enteros en los siguientes campos:

- "initYear": año por el que quieres comenzar a buscar artículos.

- "endYear": hasta que año quieres buscar.

A continuación mostramos un ejemplo de documento *JSON* para realizar una petición a la *API REST* y de 4 autores, con un filtro temporal de 1900 a 2050 y utilizando todas las fuentes de información:

```
{
  "authors": [
     {"author": "Adrian Riesco 001", "link": "https://dblp.org/pid/35/4359.html"},
    {"author": "Alberto Verdejo", "link": "https://dblp.org/pid/70/253.html"},
    {"author": "Isabel Pita", "link": "https://dblp.org/pid/35/2414.html"},
    {"author": "Narciso Martí-Oliet", "link": "https://dblp.org/pid/34/4176.html"}
 ],
  "filters": {
  "checkInproceedings": true,
  "checkArticles": true,
  "checkIncollections": true,
  "checkSchoolar": true,
  "checkGGS": true,
  "checkCore": true,
  "checkScopus": true,
  "checkJRC": true,
```

```
"mail": "tuEmailUCM",
"pass":"TuPassUCM",
"initYear": 1900,
"endYear": 2050
}
```

Capítulo 6 - Contribuciones

Este proyecto de una duración cercana a 9 meses ha sido realizado de forma equitativa por cada uno de los participantes de este proyecto. Desglosamos al detalle las aportaciones realizadas por cada uno de los miembros:

6.1 Francisco Rafael García Rofes

Investigación

Antes de empezar a desarrollar el proyecto tuvimos una reunión con los tutores los cuales nos comentaron lo que se iba a realizar y la técnica que lo llevaría a cabo, el scraping. Comencé a hacer un análisis de las diferentes herramientas de scraping que podíamos utilizar y después de una semana decidimos que utilizaremos Puppeteer, una biblioteca (framework) desarrollada por *Google* en lenguaje Javascript la cual nos sería más familiar a la hora de utilizar Javascript y ser una biblioteca de *Node.js*.

Después de varias reuniones con los tutores marcando los objetivos y funcionalidades que debía contener el proyecto se dio el visto bueno a la opción de Puppeteer, además se decidió que el proyecto encajaba como aplicación web con su interfaz gráfica para un uso más intuitivo.

Por consiguiente dividimos el trabajo y en mi caso se me asignó el scraping y actualización en 3 páginas de las que debía sacar datos de un autor en concreto automatizando todo el proceso, estas páginas son JCR, *Scopus* y *Google Scholar*.

Implementación

A la hora de hacer scraping en una página lo primero que tuve que hacer fue un análisis exhaustivo de la propia página y su comportamiento, recolectando información y haciéndome una idea de cuáles iban a ser los elementos que necesitaba y cuáles iba a desechar. Esto me llevó varias semanas, y decidí empezar por *Google Scholar* ya que, de las tres, era la única de todas en la cual no hacía falta un inicio de sesión previo. Era un proceso nuevo para mi y la forma de pensar era muy diferente a la programación tradicional.

Después de acabar con *Google Scholar*, continué tanto con *Scopus* como con *JCR* y, aunque mi conocimiento era mucho más amplio pasadas las semanas, tardé unos 3 meses en conseguir el objetivo final en ambas páginas, debido que las propias páginas no tenían bien definidas las clases e identificadores de elementos.

Pruebas

En cuanto a las pruebas que hice, estuve testeando todas las funcionalidades, tanto las creada por mi como por mis compañeros, con distintos autores y docentes, comprobando y solventando los errores que ocurrían debido a lo ya comentado antes sobre la mala implementación de las páginas en las que estábamos haciendo scraping.

Estas páginas fueron las que ocuparon la mayor parte de mi tiempo en el proyecto, pero también tuve reuniones con Luis para resolver distintos problemas que nos sucedían a la hora de mandar los datos del servidor al cliente, especialmente los errores que daba la página *JCR* en distintos artículos, además de la ayuda en la maquetación y elección de colores y tablas.

Memoria

Para la realización de la memoria decidimos dividir los puntos comunes y además cada uno escribir la parte realizada en código. Realicé la redacción de los objetivos que pusimos en común al principio del proyecto, la imagen y explicación de la arquitectura, además en ese mismo capítulo realice la parte de servidor (*Scholar*, *JCR*, *Scopus*). También he contribuido en los preliminares y las conclusiones y trabajo futuro.

Por último, realicé un repaso general de todos los capítulos para corregir errores ortográficos y de comprensión.

6.2 Rubén Martín Acebedo

Investigación

En primer lugar, para poder llevar a cabo este proyecto era necesario estudiar en profundidad las tecnologías a utilizar, para ello todos los miembros del equipo tuvimos que analizar las diferentes herramientas que permitieran scraping y automatización, así que estuve dedicando las dos primeras semanas a experimentar los *framework* de scraping disponibles en distintos lenguajes (*Javascript*, *Python*) y con numerosas variantes por tecnología (*Cheerio* y *Puppeteer* para *Javascript*).

Tras esto, al dividir las tareas se me fui el encargado de la modelización y configuración de la conexión cliente vía *API REST*, la cual realicé en colaboración con Luis, dándome las pautas que necesitaba y por mi parte definiendo los formatos que me tenía que enviar, llegando al final a un resultado bastante positivo usando apenas dos *endpoints* sin necesidad de utilizar bases de datos, lo cual aligera bastante el trabajo.

Implementación

En términos de desarrollo real de *scraping*, fui el encargado de extraer información y automatizar el flujo iniciando desde la web del *DBLP*, donde me encargué del procedimiento completo de iniciar con las correspondientes optimizaciones del procedimiento de scraping con *Puppeteer* (quitarle *CSS* a la web, imágenes y elementos Javacript para optimizar el proceso de carga). Tras este paso, buscaba los autores en el *DBLP*, via automatización lograba abrir sus principales datos en formato *XML* y aquí procedía a extraerlos. Fue una tarea muy compleja y ardua debido a que no tiene selectores simples y fáciles de identificar al ser una página autogenerada, forzar la elección de selectores muy precisos y evaluar los patrones que seguía este *XML*.

Su estructura cambiaba cuando el autor tenía homónimos, por lo que tuve que aparte hacer ajustes en el anterior punto de configuración con ayuda de Luis para crear un end-point extra y pedir una confirmación de que el autor es el correcto de los ofrecidos por el *DBLP*.

Tras la extracción de estos datos y guardarlos dinámicamente en una estructura de datos en memoria, procedí a extraer la información y automatizar tanto la clasificación *CORE* como *GGS*, no obstante, nos dimos cuenta que el *GGS* era proporcionado en varios archivos *CSV* por la propia web
http://scie.lcc.uma.es/gii-grin-scie-rating/ratingSearch.jsf
Por lo tanto, se decidió junto a los tutores usar esos recursos optimizando tiempo y esfuerzo. Mi trabajo aquí consistió en limpiar los *CSV* dejando solo los datos que nos

interesaban, cambiar el formato a *JSON* y utilizar los filtros pertinentes para recolectar eficientemente la información, aquí Francisco me ayudó a optimizar su consulta.

A nivel de extracción de los datos requeridos para el ranking *CORE* fue otro de los procesos más tediosos del programa:, tiene muchos selectores complejos, poco específicos y una cantidad enorme de filtros con patrones a veces sin correlación; lo que requirió una gran cantidad de tiempo Para optimizar el tiempo de ejecución tuve que emplear varias estructuras de datos que almacenarán en memoria los datos pertinentes de cada entrada para evitar la duplicidad de consultas en el caso de que otro autor futuro en esa misma petición sea coautor del mismo artículo. Esto redujo notablemente el tiempo de ejecución en este punto, que junto a la puesta a punto inicial de borrado de *CSS* e imágenes se ha logrado un resultado más que satisfactorio.

Desde aquí hice pequeñas aportaciones al scraping de Google Scholar, como el deslizador web el cual baje hasta el punto más inferior de la página con una función asíncrona dentro de una función propia de Puppeteer que emula la propia consola del navegador, tarea que fue realmente compleja de desarrollar sin variables externas de control.

Tras todo esto se nos requirió que la aplicación pudiese filtrar por tipos de publicación y tipo de datos, de esto me encargué yo en la parte de servidor y Luis en la de cliente, tuvimos una reunión para detallar el formato de los datos que debería recibir endpoint.

Despliegue

En este apartado de configuración, también fui en encargado del despliegue, ya que no lo hicimos de forma convencional, me encargue de realizar su despliegue en cada entrega, así los tutores eran capaces de testear nuestras aportaciones con absoluta sencillez, simplemente tenían que acceder a la URL que les enviamos y esta era inmutable.

En ese punto hubo problemas importantes, ya que probé numerosas plataformas, sin éxito. La primera a testear *Heroku*, la cual pese a tener una configuración relativamente sencilla, solo dejaba un máximo de 30 segundos como duración máxima de una petición HTTP. Esto no era válido para los requisitos de la aplicación, donde para este proyecto en concreto, es muy raro que una petición completa baje de los 3 minutos. Tras investigar descubrí que las *Google Functions*, un servicio *serverless* de *Google*, permitían su despliegue y hasta 9 minutos. Esto no era suficiente en caso de usar todas las capacidades de la herramienta con muchos investigadores simultáneamente, pero normalmente para invocaciones con 3-4 personas es suficiente.

Memoria

La parte de la memoria fue repartida de forma equitativa, hicimos en reuniones conjuntas los puntos comunes que necesitaban debate previo como el índice, las conclusiones o la bibliografía.

Me encargué de la sección "motivación" del proyecto en la introducción, después expliqué en la sección del servidor todos los puntos donde participé, desde el proceso de datos hasta cada uno de los siguientes puntos: *DBLP*, *Core* y *GGS*.

Por último me encargué de la guía de uso y de hacer una revisión global de la memoria para aportar aquellas cosas que no hayan podido ver mis compañeros desde su punto de vista.

6.3 Luis Antonio Rojas Ramírez

Investigación

Antes de comenzar con el desarrollo de este trabajo de fin de grado, lo primero que había que hacer era dialogar y dejar claro los puntos sobre los que iba a tratar el proyecto. Con dicho objetivo hicimos varias reuniones, donde los tutores nos expusieron los distintos enfoques que querían abordar con este trabajo, y entre los integrantes del equipo llegamos a la conclusión de que el trabajo debía consistir en una aplicación web.

Para poder empezar con el desarrollo dedicamos un par de semanas a investigar y aprender la sintaxis necesaria para la realización de la aplicación, así como los distintos lenguajes que íbamos a utilizar. A mí personalmente, aparte de hacer dicha investigación se me designó realizar un estudio de las diferentes tecnologías que podrían usarse para llevar a cabo la tarea del diseño gráfico de la aplicación.

Después de investigar durante unas semanas llegué a la conclusión de que la mejor herramienta para hacer este proyecto era *React*, ya que era una herramienta novedosa que no habíamos explicado en la carrera y que en un futuro podría tener mucha demanda en el mercado laboral.

Implementación

A continuación reuní información sobre distintos cursos de React y elaboré una estrategia para que todos pudiesen conocer este *framework*

Posteriormente empecé con el desarrollo de la interfaz, en principio consistía en un buscador donde el usuario tendría que insertar el nombre o los nombres de aquellos autores sobre los que quisiera obtener la información filtrada, esta información sería mostrada en un formato de tipo tabla para que el usuario pudiera acceder cómodamente a ella.

Después de construir un prototipo para que fuera validado por los tutores, y tras varias interacciones con ellos, decidieron que también querían que se pudiera descargar esa información en un JSON para su posterior manipulación. En un principio realicé esta acción guardando todos los datos en un JSON que estaba fijo como un archivo dentro de la aplicación, pero gracias a una investigación logré que ese *JSON* se creará de forma dinámica y nos ahorremos ese espacio en la aplicación.

Revisiones y mejoras

Cuando ya teníamos una aplicación funcionando y validada, había que hacer ciertas mejoras para ampliar la funcionalidad de la web, uno de esos arreglos fue el de añadir unos filtros para que el usuario pudiera elegir qué tipo de datos quería buscar. Para ello había que modificar la interfaz en varios sitios, tanto en el formulario inicial como en las tablas, ya que había que mostrar en estas qué filtros se habían seleccionado.

Antes de dar por terminado el proyecto surgieron unos problemas, ya que al realizar la búsqueda de un autor, había que contemplar un diseño para poder seleccionar entre sus homónimos, de forma que no se descuadre lo que ya teníamos hecho. Esta parte también la realicé yo, junto con la elaboración de una animación para mostrar que la búsqueda se está realizando.

Finalización

Una vez arreglados estos cambios, simplemente faltaba ayudar un poco a los compañeros con algún que otro error que sucedió en algunas páginas donde buscábamos la información y mostrar los errores en el formulario cuando el usuario no podía encontrarse.

Para finalizar, desarrollé una nueva pestaña en colaboración con Francisco en la cual se mostraban los errores que habían surgido durante el proceso de búsqueda de información.

Memoria

Para la realización de la memoria decidimos dividir los puntos comunes además de que cada uno escriba sobre la parte realizada en código. Mi parte consistió en la

redacción del plan de trabajo, explicar cómo hemos dividido la carga y el tiempo dedicado a cada una de las fases. Otra de las partes que realize fue toda la documentación del servidor (preliminares, guia de uso e implementación).

Junto con todo lo anterior también realicé de forma equitativa puntos de la memoria como pueden ser el trabajo futuro, las conclusiones, bibliografía y una revisión sobre todos los puntos en general.

Capítulo 7 - Conclusiones y trabajo futuro

El objetivo principal de nuestra elección de Trabajo Fin de Grado era aprender desde cero a sacar datos y automatizar tareas en una web (programar un algoritmo que permita extraer sus datos en el formato deseado) para futuros proyectos personales. Echando una mirada en retrospectiva nos parece que este objetivo principal está más que cumplido, el nivel exigido en el TFG en torno a la extracción de datos, la automatización del bot que realiza el flujo automático para llegar a estos datos y su procesamiento ha sido alto. Debido a la gran variedad de retos que nos ha supuesto cada etapa de automatización hemos aprendido a usar casi todas las herramientas proporcionadas por Puppeteer de forma eficiente, lo que nos ha ayudado a resolver cada problema que ha ido surgiendo. Hemos logrado hacer un uso correcto de selectores para intentar hacer mantenible el máximo tiempo posible el código (dentro de los límites que nos da el trabajar sobre una web de terceros y no tener su control) siendo muchos de ellos muy complejos y trabajosos de lograr encontrar para mil situaciones distintas que solucionar.

También de cara al futuro nos ha resultado positivo el trabajar con entornos de despliegue reales e integraciones con git, herramienta la cual hemos podido dominar en este proyecto entendiendo un flujo de trabajo de dos partes distintas conectadas vía un *API REST*, entender cómo conectarlas en un entorno real ha sido muy enriquecedor.

En global creemos que el proyecto ha sido un éxito, ya que resolvemos el problema que motivó su creación: poder obtener información de páginas de información bibliométrica ordenada y de una forma sensiblemente veloz.

Además de cumplir todos los objetivos que nos propusimos al comenzar el proyecto: el tiempo final para obtener los datos es muy inferior comparado con la forma tradicional (uno a uno manualmente),por otro lado tanto la aplicación web como la versión local han sido un éxito ya que simplifica el uso de la aplicación.

El único punto agridulce es que, al depender de páginas de terceros, la extracción de la información puede dejar de funcionar si cambia notablemente la estructura de alguna web, obligando a rehacer total o parcialmente en función del tamaño de la modificación. Esto nos ocurrió durante el desarrollo del proyecto en el momento que Scopus añadió nuevas funcionalidades variando la estructura de la web y exigiéndonos adaptarnos a sus cambios.

Trabajo futuro

Aunque la aplicación está operativa y ofrece buenos resultados hemos decidido ampliar el proyecto para futuras iteraciones con los siguientes puntos:

- *Mejoras rendimiento*

Para mejorar los procesos de carga se podría crear una base de datos para almacenar los resultados de los autores ya buscados, de tal manera que se mejoren considerablemente los tiempos de búsqueda de los autores.

Otra de mejoras que proponemos se centra en la forma de realizar el scraping de datos, ya que en estos momentos se realiza de forma síncrona, actualmente se ejecutan de uno en uno los algoritmos necesarios para la obtención de datos en cambio, si dicho algoritmo se realizará de forma asíncrona, es decir, si se accediese a todas las páginas simultáneamente, se mejorarían mucho los tiempos de finalización.

- Mejoras visuales

Para la capa visual queremos que el spinner de carga (actualmente una imagen de Pacman) se sustituya por una barra de progreso que indique cada paso del scrapping, tanto el autor que se está procesando actualmente , como las diferentes etapas que va siguiendo el proceso de scraping.

Otra de las mejoras que nos gustaría implementar es el desarrollo de una nueva interfaz para solicitar la autenticación necesaria para la búsqueda de información en JCR y Scopus. Nos gustaría que se solicitara dicha autenticación en el momento justo de hacer inicio de sesión en las plataformas mencionadas anteriormente.

- Mejoras de accesibilidad

Para mejorar el acceso a la aplicación queremos realizar un programa para escritorio que pueda ser descargado desde la plataforma web.

Otro de los objetivos que tenemos en mente es buscar un hosting con mayor capacidad de procesamiento y conseguir aumentar el tiempo de procesamiento de autores.

Para finalizar, queremos extender la aplicación para que se devuelva la información obtenida en diferentes formatos (CSV, PDF, etc.) para su posterior procesamiento.

Capítulo 8 - Conclusions and future work

The main objective of our choice of work was to learn to scrape a web and automate tasks on a web (program a bot that allows to extract your data in the desired format) for future personal projects, looking back, it seems that this main objective is more than fulfilled, the level required in the TFG around data extraction, automation of the bot that performs the automatic flow to reach this data and its processing has been high, due to the great variety of challenges that each stage of automation has proposed us we have learned to use almost all the tools provided by Puppeteer in an efficient way that have helped us to solve each problem that has arisen, we have managed to make a correct use of selectors to try to make the code maintainable as long as possible (within the limits that are given when working on a third party website and not having the control) being many of them very complex and difficult to find for a thousand different situations to solve.

Also for the future it has been positive for us to work with real deployment environments and integrations with git, a tool which we have been able to master in this project understanding a workflow of two different parts connected via a REST API, understanding how to connect them in a real environment has been very enriching.

Overall we believe that the project has been a success, since we solve the problem that motivated its creation, to be able to obtain information from educational research pages in an orderly and sensibly fast way.

In addition to meeting all the objectives we set ourselves at the beginning of the project: the final time to obtain the data is much shorter compared to the traditional

way (one by one manually), on the other hand both the web application and the local version have been a success as it simplifies the use of the application.

The only bittersweet point is that by relying on third party sites to extract the information, we know that if they change significantly the structure of any web can stop working that section giving the possibility of having to redo all or part depending on the size of the modification, leaving the uncertainty always a possibility of change, as already happened to us during the development of the project where websites like scopus added new features varying the structure of the web and requiring us to adapt to their changes.

Future work

Although the application is operational and it delivers good results we have decided to extend the project for future iterations with the following items:

- Performance improvements

To improve the loading processes we have decided to create a DB in which the results of the authors already searched are stored, in such a way that the search times of the authors are considerably improved.

Another improvement we propose focuses on how to perform data scraping, since at the moment it is done synchronously, currently the algorithms necessary to obtain data are performed one by one, however, if this algorithm is performed asynchronously, that is, if all pages are accessed simultaneously, the completion times would be greatly improved.

- Visual improvements

For the visual layer we want the loading spinner (currently a pacman) to be replaced by a progress bar that indicates each step of the scraping, both the author that is currently being processed and the different stages of the scraping process.

Another improvement we would like to implement is the development of a new interface to request the authentication required for searching information in JCR and Scopus. We would like this authentication to be requested at the very moment of logging in to the platforms mentioned above.

- Accessibility improvements

To improve the accessibility of the application we want to make a desktop program that can be downloaded from the web platform.

Another goal we have in mind is to look for a hosting with more processing capacity, thus increasing the processing time of authors.

Finally we want to add a tool to the application to return the information obtained in different formats (CSV, PDF, etc) for further processing.

Capítulo 9 - Glosario de términos:

- **scraping:** es una técnica utilizada mediante programas de software para extraer información de sitios web.
- **Puppeteer**: framework desarrollado en Javascript y mantenido por Google, cuyo propósito es proporcionar las herramientas para realizar el scraping de datos.
- **React**: Biblioteca de Javascript de código abierto desarrollada por Facebook para facilitar el desarrollo de aplicaciones web "*Single Page*".
- **Node.js**: es un entorno Javascript de lado de servidor que utiliza un modelo asíncrono y dirigido por eventos.
- **DBLP**: Digital Bibliography & Library Project, se trata de una base de datos sobre los investigadores científicos en el campo de ciencias de la computación, la cual facilita todos sus artículos y proporciona información sobre ellos.
- **Core:** Computing Research and Education Association of Australasia, Core, es una asociación de departamentos universitarios de informática de Australia y Nueva Zelanda, con un ranking propio de referencia mundial.
- **GSS:** *GSS* Conference Rating es una herramienta de acceso libre para la evaluación de los congresos de informática que ha sido desarrollada por el GII (Group of Italian Professors of Computer Engineering), el GRIN

(Group of Italian Professors of Computer Science), y la SCIE (Spanish Computer-Science Society).

- **Google Scholar:** Buscador de *Google* enfocado y especializado en la búsqueda de contenido y bibliografía científico-académica.
- **Scopus**: Scopus es una base de datos bibliográfica de resúmenes y citas de artículos de revistas científicas y congresos.
- **JCR:** Journal Citation Reports es una publicación anual que realiza el Instituto para la Información Científica, que actualmente es parte de la empresa Clarivate. Esta publicación evalúa el impacto y relevancia de las principales revistas científicas del campo de las ciencias aplicadas y sociales. Originalmente era parte del Science Citation Index, y actualmente está realizado a partir de los datos que este contiene.
- **Journal Impact Factor:** Es una fórmula matemática la cual determina la relación entre el número de citas a la revista en el año X con la cantidad de elementos citables en los años X-1 y X-2.

Capítulo 10 - Bibliografía

[1] Node.js Design Patterns: Master best practices to build modular and scalable server-side web applications, Mario Casciaro & Luciano Mammino, 2016.

[2] Fullstack React: The Complete Guide to ReactJS and Friends, Anthony Accomazzo & Nate Murray, Ari Lerner, 2017.

[3] Introduction to React, Cory Gackenheimer, 2015.

[4] Scraping Javascript-dependent website with Puppeteer: Node.js asynchronous runs. to scrape JS-stuffed websites, Igor Savink, 2020

[5] Fullstack Vue: The Complete Guide to Vue.js by Hassan Djirdeh & Nate Murray & Ari Lerner, 2018

[6] Pro Angular 9: Build Powerful and Dynamic Web Apps ,Adam Freeman, 2020

[7] Firebase: trabajar en la nube: Vicente Carbonell & Jordi Bataller & Jesús Tomás & Jaime Lloret, 2019

[8] Go Web Scraping Quick Start Guide: Implement the power of Go to scrape and crawl data from the web,Vincent Smith , 2019