UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Impact Factor Oracle
Documentación Técnica

Presentado por Gadea Lucas Pérez
en Universidad de Burgos — 3 de febrero de 2023
Tutores: Virginia Ahedo y Álvar Arnaiz

Índice general

Índice general i

Índice de figuras iii

Índice de tablas iv

Apéndice A Plan de Proyecto Software 1
A.1. Introducción 1
A.2. Planificación temporal 1
A.3. Estudio de viabilidad 3

Apéndice B Especificación de Requisitos 5
B.1. Introducción 5
B.2. Objetivos generales 5
B.3. Catálogo de requisitos 5
B.4. Especificación de requisitos 5

Apéndice C Especificación de diseño 7
C.1. Introducción 7
C.2. Diseño de datos 7
C.3. Diseño procedimental 7
C.4. Diseño arquitectónico 7
C.5. Prototipo inicial 7
C.6. API 13

Apéndice D Documentación técnica de programación 15
D.1. Introducción 15

D.2. Estructura de directorios 15
D.3. Manual del programador 15
D.4. Compilación, instalación y ejecución del proyecto 16
D.5. Pruebas del sistema 16

Apéndice E Documentación de usuario **17**
E.1. Introducción 17
E.2. Requisitos de usuarios 17
E.3. Instalación 17
E.4. Manual del usuario 17

Bibliografía **19**

Índice de figuras

C.1. Ejemplo de búsqueda por palabra clave 8
C.2. Localización de la revista en Google Scholar 9
C.3. Salida por consola 9
C.4. Captura de reCAPTCHA de Google 10

Índice de tablas

A.1. Planificación de Sprints 2

B.1. CU-1 Nombre del caso de uso. 6

Apéndice A

Plan de Proyecto Software

A.1. Introducción

La planificación temporal es esencial para el éxito de cualquier proyecto, especialmente en el desarrollo de *software*. En esta sección se detallará cómo se llevará a cabo el cronograma siguiendo una metodología ágil tipo Scrum, mediante el uso de *sprints*. Se describirán los pasos necesarios para planificar y gestionar de manera eficiente el tiempo y los recursos disponibles, asegurando así el cumplimiento de los objetivos del proyecto en el plazo establecido.

A.2. Planificación temporal

El presente proyecto comienza en septiembre y se extiende hasta junio. Durante este período de tiempo, es importante asegurarnos de que todas las tareas y hitos estén claramente definidos.

Para lograr esto, se han realizado reuniones periódicas para discutir el progreso del proyecto y asegurarnos de que estamos en el camino correcto. Además, hemos implementado *sprints* regulares para asegurarnos de que estamos avanzando de manera constante y cumpliendo con nuestras metas a tiempo.

Con esta planificación temporal sólida, estamos seguros de que podremos completar el proyecto a tiempo y cumplir con los objetivos establecidos. Sin embargo, es importante ser flexibles y estar preparados para hacer ajustes según sea necesario a medida que avanzamos en el proyecto, puesto que se realiza al mismo tiempo que trascurre el curso académico.

En la siguiente tabla se recogen los distintos *sprints* junto con su duración, objetivos y tareas.

Tabla A.1: Planificación de Sprints

Sprint	**Duración**	**Objetivos**	**Tareas**
1	19/09/2022 03/10/2022	Comenzar el desarrollo del proyecto	Elegir un modelo de referencias bibliográficas. Definición de conceptos.
2	03/10/2022 17/10/2022	Tareas de investigación y documentación	Búsqueda de antecedentes. Familiarizarse con la memoria en LaTeX.
3	17/10/2022 14/11/2022	Investigación y creación de un prototipo inicial	Búsqueda de información sobre MIAR Documentación sobre el JCR y el índice de impacto. Prototipo que permita la búsqueda de artículos en GS. Extracción de datos: ¿qué datos se pueden extraer? Reflexión sobre el diseño de la BBDD. Prueba del prototipo.
4	14/11/2022 28/11/2022	Base de datos y prototipo	Creación de la base de datos en MariaDB. Prueba de la BBDD. Mejoras del prototipo.
5	28/11/2022 12/12/2022	Base de datos y prototipo	Se muda la BBDD a Postgres. Pruebas e investigación para extraer el DOI.
6	09/01/2023 23/01/2023	Documentación y obtención del DOI	Obtención del DOI a través de Crossref. Avence de la memoria y los anexos. Mejora de la BBDD

A.3. Estudio de viabilidad

Viabilidad económica

Viabilidad legal

Apéndice B

Especificación de Requisitos

B.1. Introducción

Una muestra de cómo podría ser una tabla de casos de uso:

B.2. Objetivos generales

B.3. Catálogo de requisitos

B.4. Especificación de requisitos

CU-1	**Ejemplo de caso de uso**
Versión	1.0
Autor	Alumno
Requisitos asociados	RF-xx, RF-xx
Descripción	La descripción del CU
Precondición	Precondiciones (podría haber más de una)
Acciones	1. Pasos del CU 2. Pasos del CU (añadir tantos como sean necesarios)
Postcondición	Postcondiciones (podría haber más de una)
Excepciones	Excepciones
Importancia	Alta o Media o Baja...

Tabla B.1: CU-1 Nombre del caso de uso.

Apéndice C

Especificación de diseño

C.1. Introducción

C.2. Diseño de datos

C.3. Diseño procedimental

C.4. Diseño arquitectónico

C.5. Prototipo inicial

Al revisar la viabilidad del proyecto se plantearon posibles dificultades que podían surgir. Tras comprender que las complicaciones eran numerosas, se decidió crear un **prototipo sencillo**, consistente en un *script* en Python, que trataría de lanzar mil peticiones de búsqueda. Esto nos permitiría establecer los límites de realizar "web-scrapping" sobre Google Scholar.

Así pues, manos a la obra, se desarrolló un *script* sencillo, que solicita acceso a la página principal de Google Scholar y, mediante métodos HTTP, realiza una búsqueda (a partir de parámetros solicitados por pantalla). Finalmente, extrae el título de la página resultante tras hacer la búsqueda.

Todo esto se logra haciendo uso de la biblioteca de Python **BeautifulSoup** (ver documentación en el siguiente enlace). Esta biblioteca contiene métodos centrados en la extracción de datos de archivos HTML y XML para su posterior análisis. Es fácil advertir cuán idónea resulta esta biblioteca para nuestro propósito.

El primer obstáculo encontrado es que Google Scholar no permite hacer búsquedas en función de la revista. Si se trata de escribir el nombre de una revista en el buscador, aparecerán artículos relacionados que mencionan esa revista, pero no siempre artículos de la revista en cuestión. En cambio, permite hacer búsquedas a partir de palabras clave. Sin embargo, aunque no se puede buscar directamente por revista, sí se puede extraer de los resultados encontrados tras buscar una temática concreta.

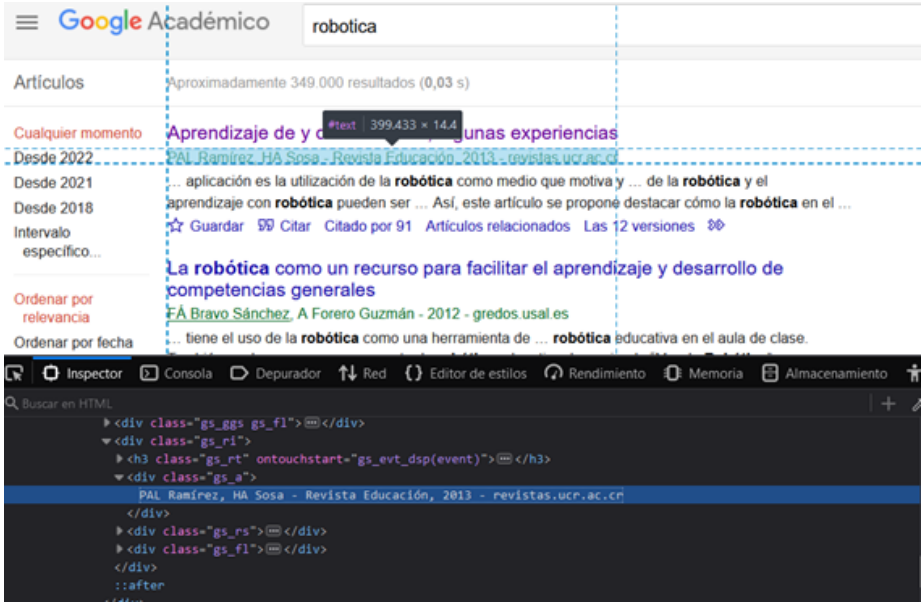

Figura C.1: Ejemplo de búsqueda por palabra clave

Sin embargo, existe una problemática: en cada artículo se hace referencia a la revista que lo publica de forma distinta. Por ejemplo, como se puede apreciar en la siguiente imagen, la revista se encuentra ubicada en una "etiqueta" HTML diferente.

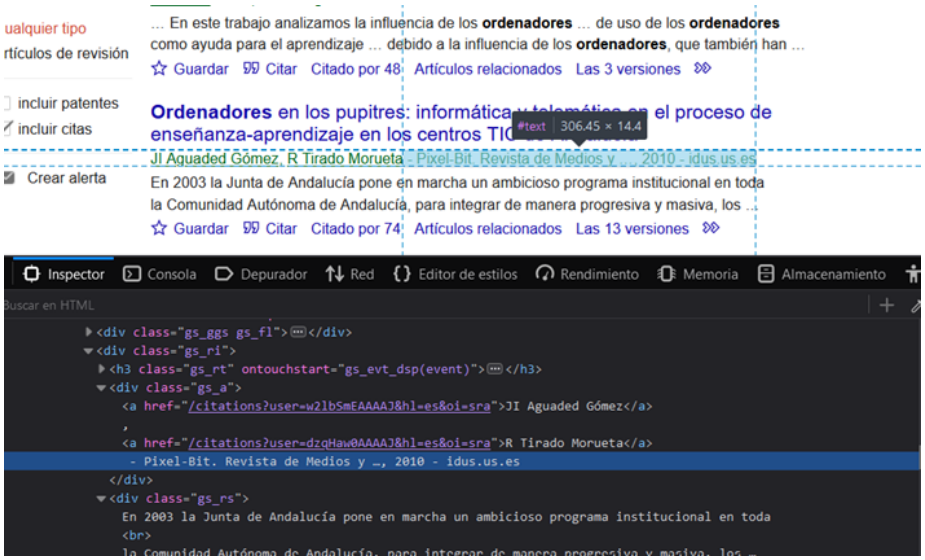

Figura C.2: Localización de la revista en Google Scholar

Por ello se ha buscado una solución alternativa. Existe una etiqueta (...) que incluye aquellas palabras resaltadas en negrita en la página web. Google Scholar resalta en una búsqueda aquellas palabras que coinciden con alguno de los parámetros buscados. Así pues, si se realiza una búsqueda en Google Scholar pasando como parámetro el nombre de la revista que nos interesa, bastará con buscar en la clase "gs_a" (donde se almacenan los detalles del artículo) alguna etiqueta "b". Se ha escrito un pequeño código de prueba y, en principio, parece que funciona sin problemas. Se adjunta dicho código (extraerRevistas.py) en el repositorio de GitHub. Este código hace dos búsquedas: una a Revista "Alas Peruanas" y otra a Revista "The Lancet" (elegidas de forma aleatoria).

La salida obtenida (solo de la primera página de resultados) es la siguiente:

Figura C.3: Salida por consola

La salida por pantalla mostrada en la imagen anterior sigue la siguiente estructura: ['nombre artículo', 'nombre de la revista', número_citas, fecha, id]

De igual forma se recogen los resultados en los CSVs (codificados en UTF-8) con nombre: BBDD1.csv y BBDD2.csv.

Otra de las mejoras que se implementa en el prototipo en la capacidad de navegar a través de las distintas páginas de resultados de Google Scholar. Para ello, se incluye un nuevo campo en la *query* de búsqueda que irá aumentando de 10 en 10 por cada página de resultados nueva que se consulte. Ahora ya se obtiene un número de resultados considerable (aproximadamente 100). Estos resultados se guardarán también en los CSVs (también codificados en UTF-8) con los siguientes nom-bres: BBDD3.csv y BBDD4.csv

Tras diseñar y programar el código mencionado, se procede a su prueba. La primera ejecución del *script* resultó desaletadora, ya que, a partir de la solicitud número **726**, Google Scholar detecta que un ***bot*** está realizando búsquedas. A partir de ese momento, nuestra dirección IP queda bloqueada y las solicitudes fallan sin excepción. Google Schoolar nos redirecciona a una página (figura C.4) donde se solicita al usuario resolver un *captcha*.

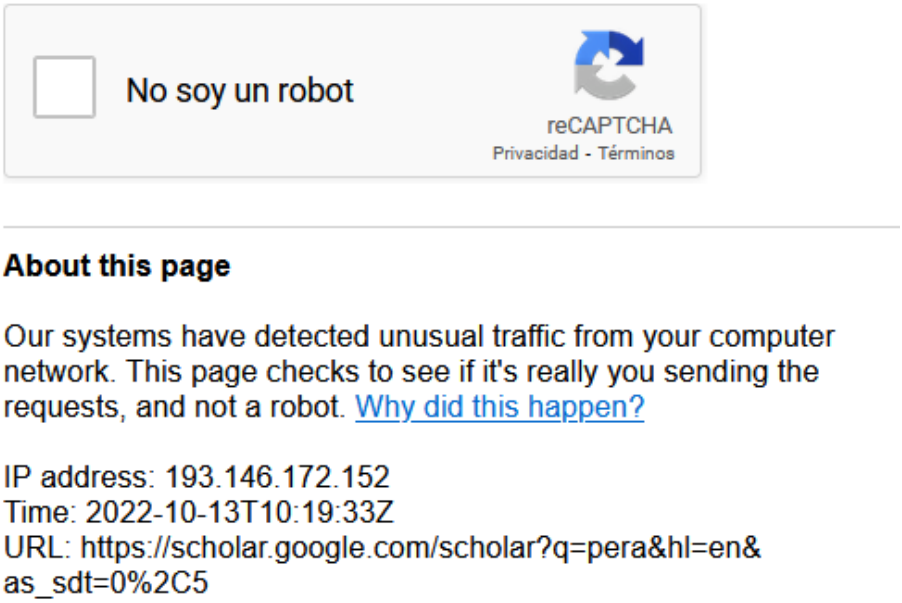

Figura C.4: Captura de reCAPTCHA de Google

El número de solicitudes exitosas es demasiado bajo para cumplir su función en nuestro proyecto, por lo que se procede a buscar una solución alternativa.

Tras distintas pruebas, se encontró una forma de superar la barrera del *captcha*. A saber: añadiendo a la *url* una sección de texto extra que permite suprimir esta excepción durante un periodo concreto de tiempo. Así pues, logramos ejecutar con éxito el *script* tantas veces como fuese necesario. Sin embargo, esta solución tampoco es válida a largo plazo. Si se intenta ejecutar de nuevo el programa, en otra sesión, la dirección vuelve a ser inválida. Además, es un remedio poco práctico, ya que se debe concatenar distintos parámetros y cadenas de texto que, dependiendo del momento y el contexto en que se ejecute, pueden no servir.

Como la propuesta anterior no fue satisfactoria, se siguió buscando opciones. La siguiente propuesta consistía en usar un agente de usuario distinto para cada solicitud[1] De esta forma "enmascaramos" nuestra dirección IP. La biblioteca **Request** de Python ofrece métodos para lograrlo. Dicho esto, se implementó un método que extrae *proxies* de listas públicas y gratuitas de Internet (v.g.: proxyscrape.com). Se prueba la ejecución del prototipo una vez más y, finalmente, funciona sin inconvenientes. Ahora ya se puede decir que el proyecto es **viable**.

La siguiente meta de nuestro prototipo es la obtención del DOI, que es un campo fundamental que funcionará a modo de clave primaria en la base de datos. De esta forma se busca evitar futuros errores a la hora de identificar un artículo o a la hora de comparar artículos para eliminar duplicados. Sin embargo, se plantea una problemática al respecto: Google Scholar no comparte el DOI de los artículos en ninguna división de su página web (si bien es cierto que algunos artículos lo incluyen al final de su URL, pero no siempre ocurre esto). Puesto que por el momento no existe una forma de obtenerlo directamente, se han propuesto varias soluciones, a saber:

- Comprobar el porcentaje de éxito de extraer el DOI de la URL de los artículos (en aquellos casos en los que aparezca).
- Acceder a la página del artículo en cuestión y extraerlo de dicha página.
- Introducir el nombre del artículo en la página de Crossref o similares, que te permiten obtener el DOI del artículo cuyo nombre pases como parámetro.

[1] Un agente de usuario es cualquier software, que actúa en nombre de un usuario, que «recupera, presenta y facilita la interacción del usuario final con el contenido web». Algunos ejemplos destacados de agentes de usuario son los navegadores web. La cadena User-Agent es uno de los criterios por los cuales los rastreadores web pueden ser excluidos del acceso a ciertas partes de un sitio web utilizando el Estándar de exclusión de robots (archivo robots.txt).

Como conclusión, se descarta la primera opción tras hacer una breve prueba, ya que falla en seguida. Se descarta también la segunda opción, ya que cada página sitúa el DOI en un sitio haciendo muy difícil la búsqueda del mismo a través de un algoritmo. Por lo tanto, la opción más adecuada en este caso es la tercera. Aunque supone una búsqueda extra en la complejidad algorítmica del prototipo, es la única forma segura de calcular el DOI sin equivocaciones. Por lo tanto, se incluye en el bucle del prototipo un nivel extra de profundidad de búsqueda.

Para hacer la búsqueda en Crossref es necesario pasar tanto el título como el año de publicación para asegurarnos de que se obtiene exactamente el artículo que deseamos y no otros similares. Además, es preciso tener en cuenta que Crossref no permite a los programas hacer búsquedas desde la misma interfaz que los usuarios (para evitar que los programas bloqueen las búsquedas de los usuarios). Por lo tanto, se realizarán las búsquedas en la dirección donde se ubica la API creada por Crossref especialmente para la extracción de datos por parte de programas (TDM).

Con el prototipo listo, se prueba a extraer la información de los artículos de las primeras 20 páginas de resultados de Google Scholar de dos revistas. Los resultados no son muy esperanzadores: se obtienen 180 artículos de cada revista en 15 minutos. Estos resultados no son eficientes, por lo que se tratará de optimizar el prototipo siguiendo dos pautas. Primero, se tratará de extraer las llamadas a Crossref de forma que solo se tenga que realizar una única llamada una vez que se han extraído el resto de detalles sobre los artículos. La segunda pauta es emplear programación concurrente para ejecutar varios hilos al mismo tiempo.

Así, cada revista será procesada por un hilo distinto. Para ello, será necesario recurrir a la librería de Python **multiprocessing**.Tras actualizar estos cambios, el prototipo obtiene resultados mucho más óptimos: obtiene los 180 artículos de ambas revistas en tan solo 8 minutos.

El próximo objetivo será determinar el número máximo de hilos que se pueden lanzar sin impactar negativamente el rendimiento del prototipo. Para ello, se llevará a cabo una prueba de estrés en el dispositivo del usuario. El resultado se presentará a través de un gráfico que ilustra la relación entre el número de hilos y el desempeño del programa.

[INSERTAR GRÁFICO]

C.6. API

Se pretende desarrollar una API REST sencilla que utilizará una arquitectura cliente-servidor de dos capas.

La funcionalidad principal de la API será predecir el índice de impacto de una lista de revistas elegida por el usuario. Además, se proporcionarán gráficos e información adicional para ayudar a los usuarios a entender mejor los resultados. Por otro lado, se podrán distinguir dos perfiles de usuario: el usuario normal y el administrador. El administrador tendrá permisos para "reentrenar" la red que predice los índices y gestionar al resto de usuarios.

El *backend* se programará en Python y el *frontend* con HTML, CSS y JavaScript. Para ello, se utilizará Flask, que se trata de un framework de Python para el desarrollo de aplicaciones web.

Para la conexión con la base de datos, se hará uso de *psycopg2*, que es un módulo de Python que proporciona una interfaz para conectarse y interactuar con bases de datos PostgreSQL. Es una de las librerías más populares y ampliamente utilizadas para trabajar con PostgreSQL en Python. Además, es compatible con la mayoría de las versiones de Python y es muy fácil de utilizar.

Por otro lado, para la autenticación, se han evaluado varias opciones que ofrece Flask, a saber: Flask-Login, Flask-Security y Flask-JWT. Finalmente, se ha elegido Flask-Login debido a su sencillez y facilidad de uso. Esta opción permite al usuario iniciar sesión con un nombre de usuario y una contraseña y almacena la información de la sesión en las *cookies* del navegador. Este paquete incluye la protección de rutas con decoradores.

Apéndice D

Documentación técnica de programación

D.1. Introducción

D.2. Estructura de directorios

D.3. Manual del programador

Esta sección está destinada a proporcionar información detallada sobre cómo utilizar el programa desarrollado en el proyecto. Aquí se describen las funciones y características clave de dicho software, así como los detalles sobre la configuración y la integración del programa con otros sistemas.

Base de datos

Para aquellos usuarios de sistemas operativos Windows, es importante tener en cuenta un detalle en el momento de obtener información de la base de datos en PostgresSQL. Cuando se recolectan los datos en formato CSV, es necesario cambiar la ruta de generación de estos archivos a la carpeta "Public"[1]. Esto se debe a que el usuario por defecto que genera PostgresSQL (usuario "postgres"), no tiene los permisos suficientes para leer e importar datos de los CSV en cuestión ya que, para ello, se utiliza el comando *copy* (que solo puede ser ejecutado por "Superusers")[1]. Sin embargo, la carpeta

[1]Directorio que permite compartir archivos entre distintos usuarios en un mismo sistema Windows. Microsoft ha mantenido este directorio desde la versión de WindowsXP.

"Public" cuenta con los permisos necesarios para que el usuario que crea Postgres pueda acceder y leer los datos. Por lo tanto, es importante seguir este paso para garantizar que se pueda obtener correctamente la información recogida previamente.

D.4. Compilación, instalación y ejecución del proyecto

D.5. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

Bibliografía

[1] Jorge Domínguez Chávez. *CLIENTE PSQL DE POSTGRESQL*. 04 2020.