Ampliación de un complemento web para la gestión de bibliografías en Google Docs

David Hervás Rodríguez

Directores: Enrique Martín Martín y Adrián Riesco Rodríguez

Trabajo de Fin de Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Curso 2017 / 2018

RESUMEN

El presente trabajo tiene como objetivo modificar un complemento de Google Docs existente, cuya función consistía en realizar citas y añadir referencias mediante el uso de archivos de extensión BibTeX (.bib), para permitir más opciones de cara a la gestión bibliográfica. Dicho complemento está creado en el lenguaje propio de las aplicaciones de Google, Google Apps Script (una variante de JavaScript) y se utiliza de manera conjunta con lenguaje de marcas de hipertexto o, por sus siglas, HTML. El trabajo consta de tres modificaciones principales sobre el complemento existente, así como una investigación de diferentes complementos que poseen funcionalidades similares.

Dichas modificaciones estarán ampliamente relacionadas con la gestión bibliográfica de tal forma que la herramienta permita generar informes detallados, modificar los ficheros de extensión BibTeX para poder añadir nuevas entradas, así como procesar un número mayor de ficheros BibTeX de forma simultánea. Estas modificaciones tienen como objetivo dotar a la herramienta de un mayor atractivo de cara al uso por parte de futuros usuarios.

PALABRAS CLAVE

BibTeX, Google Apps Script, LaTeX, bibliografía, citas, Chrome Web Store

ABSTRACT

The aim of this project is to modify an existing Google Docs add-on, which worked as a citation and reference creator by using BibTeX files (.bib), in order to have a wider variety regarding the bibliography manager functionality. This add-on is created on the Google applicative language, Google Apps Script (a variant of JavaScript) and it is used along with hypertext markup language or by its acronym, HTML. The project consists of three main modifications on the existing add-on as well as an investigation of different add-ons, which possess similar functionalities.

These modifications will be widely related to the bibliographic management in such a way that the tool allows to generate detailed reports, modify BibTeX extension files to be able to add new entries as well as process a greater number of BibTeX files simultaneously. These modifications aim to provide the tool with greater attractiveness for use by future users.

KEYWORDS

BibTex, Google Apps Script, LaTeX, bibliography, cites, Chrome Web Store

ÍNDICE

1. INTRODUCCIÓN

En esta sección se abordarán los conceptos más importantes para comprender el presente trabajo. Para ello en la sección 1.1. se presentan las causas que empujaron a realizar la modificación de la herramienta. En la sección 1.2. se exponen los objetivos que se propusieron cumplir y por último en la sección 1.3. se describirá la línea de trabajo y la planificación seguidas para llevar a cabo el proyecto.

1.1. Motivación

Al realizar trabajos de investigación y análisis de temas con una gran densidad de contenido hay que consultar muchas y diversas fuentes de información de las cuales se extrae una gran cantidad de contenidos que posteriormente se reutilizarán en dichos trabajos.

Las nuevas tecnologías forman una parte esencial para contribuir a la inclusión de citas, pues permiten acceder a muchas fuentes originales para agilizar el proceso de añadir citas y referencias, haciéndolo menos costoso para el autor, pero hay que tener en cuenta todos los posibles casos que puedan representar para él una molestia y opte por no citar su trabajo.

Tecnologías como procesadores de texto que permiten una completa gestión de bibliografías como, por ejemplo, LaTeX así como los servicios de Google, entre los que destacaremos Google Docs que permite editar texto de manera colaborativa añadiendo comentarios, notas y permitiendo mantener un chat con los participantes.

Para permitir al autor incluir citas existen complementos suficientes, como BibDesk [1] o BibMe [2] de tal forma que sean presentados como herramientas amigables y fáciles de utilizar, pero, aunque suelen ser suficientes para que se incluya en el trabajo una referencia a la fuente de la que se ha obtenido información, en otras ocasiones los usuarios no quieren aprender a utilizar un complemento solamente para realizar un trabajo de investigación.

Por ello, este trabajo se ha centrado en extender la funcionalidad de un complemento existente, el cual combina las herramientas de procesado de texto junto con el uso de Google Docs además de buscar una solución a las limitaciones ya existentes como han sido el uso inicial de un único fichero BibTeX [3] o la falta de definición de algunos tipos de entrada.

De esa manera se han podido ampliar las posibilidades que ofrecer a un usuario cuando decida instalar un complemento dirigido, en gran medida, a la gestión bibliográfica mediante archivos BibTeX de un documento de Google Docs, así como a documentar herramientas diferentes que también se centran en la citación y la gestión bibliográfica.

Este trabajo tiene aplicados los conocimientos de diversas asignaturas que se han estudiado como son: *Diseño de sistemas interactivos* para enfocar al desarrollo al

punto de vista del usuario, *Fundamentos de la programación* la cual ha influido en la forma de realizar la implementación, *Tecnologías de la programación* que me ha aportado los conocimientos necesarios para realizar este proyecto, *Ingeniería del software* con la que he podido adoptar una jerarquía de trabajo organizada y *Estructuras de datos y algoritmos* para utilizar algoritmos iterativos y recursivos en la ejecución del programa.

1.2. Objetivos

Se buscaba implementar nuevas funcionalidades con el fin de ofrecer una mayor variedad de opciones y herramientas para poder gestionar las citas y referencias bibliográficas que permitan a un usuario encontrar un motivo por el que instalar el complemento.

Para ello se fijaron las siguientes modificaciones:

- Permitir utilizar de forma simultánea varios archivos bibliográficos (BibTeX).
- Generar un informe o reporte sobre las diferentes entradas que contienen los archivos bibliográficos que se han seleccionado.
- Permitir al usuario realizar un filtrado exhaustivo de cara a permitir una personalización del reporte generado.

Adicionalmente, se decidió incluir una investigación detallada de las funcionalidades y cualidades de complementos ya existentes que permitiesen realizar citaciones y añadir referencias.

1.3. Plan de trabajo

Para realizar cada una de las modificaciones detalladas anteriormente, primero fue necesario realizar un estudio sobre las diversas posibilidades que ofrecía el lenguaje Apps Script. Dicho estudio se simplificó debido a que Google ofrece una página *web*[1] dedicada a esta tecnología.

Dichas modificaciones, y por tanto el proyecto, se llevaron a cabo entre los meses de abril y julio de 2018, bajo la tutela de los profesores Adrián Riesco Rodríguez y Enrique Martín Martín. A lo largo de esos meses se mantuvo el contacto mediante correo electrónico, concretando reuniones en su despacho, si era necesario, para observar el progreso realizado y solventar dudas de conceptos.

Posteriormente, previa reunión con los tutores, se establecieron tres hitos de rigurosa implementación para llevar a cabo la realización del proyecto. Estos tres hitos se describirán brevemente a fin de dar una idea general del trabajo realizado. Más adelante se analizarán con un enfoque más riguroso para dar información precisa de su implementación y funcionamiento.
La comunicación mediante correo electrónico sirvió para apuntalar modificaciones, discutir sobre la necesidad de optimizar las mejoras a medida que se iban

[1] *https://developers.google.com/apps-script/*

implementando y para documentar errores en los procedimientos y proceder a su resolución.

Se discutió también sobre la necesidad de profundizar en los conceptos del lenguaje Apps Script para poder implementar de manera correcta apartados del proyecto tales como el servicio HTML o la gestión de errores. Esta necesidad supuso dedicar una parte del tiempo a estudiar la documentación específica de Apps Script en lo referente al servicio HTML o al control de excepciones, entre otras documentaciones estudiadas.

1.3.1. Fases del proyecto

Estudio previo: Se extendió durante la segunda mitad del mes de abril y el mes de mayo, tiempo que el alumno empleó para familiarizarse con el lenguaje Google Apps Script, realizar tutoriales[2] proporcionados por Google y también para empezar a configurar el entorno de desarrollo y conocer el complemento existente. El estudio previo no supuso mucha dificultad gracias, en gran medida, a la claridad y el detalle de las guías de la página para desarrolladores de Apps Script.

Hito 1: Una vez que se entendían los conceptos del lenguaje Apps Script se procedió a la implementación del primer hito, el cual consistía en permitir una selección múltiple de ficheros .bib en combinación con la generación de un reporte de citas bibliográficas. Dicho proceso permite al usuario seleccionar los ficheros .bib que desee para generar un informe bibliográfico que muestre información detallada de todas las citas que se encuentren en los documentos seleccionados.

Hito 2: A partir de los resultados del primer hito, se implementaron unos filtros para el reporte generado, de cara a permitir al usuario mostrar únicamente las citas deseadas. Para realizar dicho filtrado, el usuario debe introducir los valores que desee, manteniendo un formato que se puede consultar pasando el puntero sobre el icono ℹ, de forma que el usuario pueda ser capaz de entender la forma de utilizar los filtros.

Hito 3: Una vez se concluyó la implementación de los hitos 1 y 2, se creó un nuevo botón para generar una ventana que permite añadir a un fichero .bib una entrada bibliográfica. Dicha ventana contendrá un desplegable para indicar el fichero .bib en el que añadir la nueva entrada, un selector de tipos de entrada y, en función del tipo de entrada seleccionado, se mostrarán los campos a rellenar por el usuario para añadirla.

Analizando los hitos, se puede observar una clara influencia de diversas asignaturas cursadas durante la carrera en la implementación del *proyecto*[3] como por ejemplo, Tecnologías de la Programación para implementar cada uno de los hitos, Diseño de Sistemas Interactivos sobre todo para estilizar una aplicación de cara a los usuarios, y otras más que se detallarán más adelante.

El resto de la memoria tiene la siguiente estructura:

[2] *https://developers.google.com/apps-script/overview*
[3] *https://github.com/dahervas/TFG*

- En el capítulo 2 se tratarán los aspectos relacionados con el desarrollo y que han influido en la implementación del proyecto bien por ser tecnologías necesarias para ello o bien por estar relacionadas con el ámbito de gestión bibliográfica.

- En el capítulo 3 se expondrá el desarrollo completo del proyecto empezando por explicar la estructura que lo define para seguir con el detalle de la implementación de cada hito que lo compone.

- En el capítulo 4 se detallará paso a paso mediante capturas de pantalla el proceso de generación de un reporte utilizando varios ficheros BibTeX así como la inserción de nuevas entradas.

- En el capítulo 5 se expondrán las conclusiones del trabajo y se dará a conocer el trabajo futuro pensado para extender aún más la funcionalidad de la herramienta.

- En el capítulo 6 se incluyen las referencias a la información utilizada en esta memoria.

1. INTRODUCTION

In this section the most important concepts to understand the present work will be focused. For that in section 1.1. the causes that led to the modification of the tool are presented. In section 1.2. the objectives that were set out to fulfill are exposed and finally in section 1.3. the workline and the schedule followed to carry out the project will be described.

1.1. Motivation

It is not a secret that, when writing research essays about a subject which has a large amount of content, we have to read different and as many as possible sources of information, in order to introduce said knowledge on our paper.
It is common to find many scientific works in which a large number of citations can be observed, in many cases unnecessary, believing that they provide more quality to the work or that they generate a greater impact in the scientific community.

New technologies play an important role in helping researchers to include these bibliographic references; they are able to access the original source speeding up this tedious task, automatically registering the different fields required on an entry, but it is necessary to bear in mind all the possible cases that could represent for him an inconvenience and choose not to cite his work.

Technologies such as text processors that allow a complete management of bibliographies, such as LaTeX as well as Google services, among which we will highlight Google Docs that allows editing text collaboratively adding comments, notes and allowing to keep a chat with the participants.

There are plenty of computer complements that automatically include references such as BibDesk [1] or BibMe [2], which are presented like intuitive for the user. Nevertheless, even if these programs help, most of the people won't learn how to use them for the time it could take to be fully operational using the complement.

Therefore, this work has focused on extending the functionality of an existing add-on, which combines the tools of text processing along with the use of Google Docs in addition to finding a solution to existing limitations such as the initial use of a single BibTeX [3] file or the lack of definition of some types of input.

This is why this project is focused on broadening the possibilities offered to the user when deciding to install a complement aimed at, to a large extent, bibliographic citation by BibTeX files taken from a Google document and/or keep record of other tools also directed towards citations and bibliographic management.

The knowledge applied on this essay comes from different subjects studied like: *Interactive systems development* to focus the development on the user's point of view, *Fundamentals of Programming* that influenced on the way the project was implemented, *Computer Programming Technology* that gave me the necessary knowledge to be able to do this project, *Software engineering* which allowed me to

adopt an organized working hierarchy and *Data Structures and algorithms* to use iterative and recursive algorithms on the program execution.

1.2. Objectives

The aim of this project was to implement new features in order to offer a much higher variety of options and tools to allow the user to organize the bibliographic references and, at the same time, give him or her a compelling reason to install the add-on.

For this, the changes below were set:

- To allow using simultaneously several bibliographic files (BibTeX or .bib).
- To generate a report referring to the different entries contained in the selected bibliographic files.
- To allow the user to exhaustively filter the data and therefore customizing the generated report.

Additionally, it was decided to include a detailed research regarding the existing functionalities and features of bibliographic add-ons that already exist in the market.

1.3. Work plan

In order to achieve the modifications already mentioned, first of all it was necessary to determine the possibilities offered by the scripting language Apps Script. This study was very simple thanks to Google, which has a web *page*[4] dedicated to this technology.

The said changes, and therefore the whole project, were pursued between the months of April and June 2018, under the tutoring of professors Adrián Riesco Rodríguez and Enrique Martín Martín. Throughout these months communication was made via e-mail and setting meetings in their respective offices when needed, to analyze the progress made and solve questions regarding key concepts.

After a prior meeting with both professors, three milestones of rigorous implementation were set to carry out this project. These three milestones would be briefly described later to give the reader an overview of the process. Afterwards, they will be object of a more detailed approach to give precise information regarding its implementation and functioning.

Communication via e-mail was useful to include changes, discuss about the need of optimization of the different upgrades implemented and to document mistakes in the procedures, as well as the solution.

The need to go into detail about the Apps Script language's concepts was also addressed, in order to include some sections of the project such as the HTML service or the error management. This demand implied some time spent studying specific documentation about Apps Script referring to HTML service and exceptions control, among others.

[4] *https://developers.google.com/apps-script/*

1.3.1. Project stages

Previous study: It was carried on the second half of the month of April and May, time spent by the student to familiarize with Google Apps Script language, follow *tutorials*[5] provided by Google and also to start with the configuration of the development environment and to get used to the existing add-on. The previous study was not difficult at all thanks, to a large extent, to the clarity and the detail of the guides from the Apps Script developers web page.

Milestone 1: Once the concepts about Apps Script language were understood, the implementation of the first milestone started, which was a multiple file selection combined with the creation of a report filled with bibliographical entries. This milestone will allow the user to select as many .bib files as he wants to generate a bibliographical report showing all the information about the entries contained in the selected files.

Milestone 2: Regarding the first milestone's results, a list of filters for the generated report were built so that the user will be able to choose the entries he wants to show in the report. To make that filter, the user must introduce the values he wants to filter, maintaining a defined format that can be consulted by hovering the mouse pointer over the ℹ icon so a message containing an explanation will display.

Milestone 3: Once the milestones 1 and 2 were implemented, a new button was created to show up a window that allows the user to add a new bibliographic entry to a .bib file that he selects. This window will contain a dropdown menu on which the user must select the .bib file he wants to modify, an entry type selector and, based on the entry selected, a list of fields that the user should fill will be shown.

Analyzing the milestones, it is possible to observe a clear influence of diverse subjects studied during the Degree in the implementation of this *project*[6] for example, Computer Programming Technology to implement each of the milestones, Interactive systems development especially to stylize an application regarding the users, and others that will be detailed hereinafter.

The rest of the memory has the following structure:

- Chapter 2 will deal with aspects related to the development that have influenced the implementation of the project, either because they are necessary technologies or because they are related to the field of bibliographic management.

- In chapter 3 the full development of the project will be shown beginning with an explanation of the structure that defines it followed by the detail of the implementation of each milestone that composes it.

- Chapter 4 will show step by step through screenshots the process of generating a report using several BibTeX files as well as the insertion of new entries.

[5] *https://developers.google.com/apps-script/overview*
[6] *https://github.com/dahervas/TFG*

- In chapter 5, the conclusions of the project will be presented along with the future work based on extending even more the functionality of the tool.

- Chapter 6 will contain the references to the information used on this memory.

2. PRELIMINARES

En este capítulo se tratarán los aspectos y nociones más relevantes de las metodologías y tecnologías que se van a utilizar. En primer lugar, en la sección 2.1. se explicará la herramienta y el formato BibTeX y sus principales aplicaciones, muy relacionadas con el contenido de la sección 2.2, enfocado a la tecnología LaTeX. En la sección 2.3 se dará una breve explicación de la herramienta de almacenamiento online Drive de Google. A continuación, en la sección 2.4 se dará una explicación del servicio web Google Docs el cual es una base del presente proyecto. En la sección 2.5. se introduce el lenguaje que se va a utilizar para desarrollar el trabajo, Google Apps Script. Finalmente, en la sección 2.6 se explican las bases de la creación y publicación de complementos o *add-ons* en la Chrome Web Store. A continuación, en la sección 2.7 se hará una descripción del punto de partida, es decir, la herramienta en un primer contacto y se finalizará con una descripción de otras herramientas similares recogidas en la sección 2.8.

2.1. BibTeX

La herramienta BibTeX es una aplicación, utilizada en conjunto con LaTeX, cuya función es ayudar al usuario a organizar sus referencias y crear contenido bibliográfico. Actualmente, se puede obtener de manera gratuita, además de una variedad de herramientas auxiliares, gracias a la página *web*[7] de BibTeX.

El formato de ficheros BibTeX o .bib, se utiliza para describir y procesar listas de referencias y es comúnmente utilizado con documentos LaTeX. Dichos ficheros pueden contener diversos tipos de entradas:

- **@string** utilizada para definir una abreviación que se quiere mantener fija en todo el documento.
- **@preamble** se utiliza para definir comandos que serán incluidos en el fichero que se genere, en combinación con la herramienta BibTeX.
- **@comment** sirve para indicar que una entrada completa sea considerada como un comentario y, por ello, no se tenga en cuenta a la hora de generar el documento. Además, BibTeX también considerará como un comentario todo aquello que no pertenezca a una entrada (cuyo inicio se indica mediante el carácter @).
- **Entradas:** Cada una de ellas sirve para declarar una referencia única a un tipo de publicación, por ejemplo: @*article*, @*book*, @*booklet* etc.

2.2. LaTeX

LaTeX [4] se identifica como un sistema de composición de textos enfocado a la creación de documentos que requieran de una alta calidad tipográfica. Se utiliza comúnmente para la generación de artículos y textos científicos, los cuales incluyen determinadas expresiones como, por ejemplo, fórmulas matemáticas.

[7] *http://www.bibtex.org/Links/*

2.3. Google Drive

Google Drive [5] es un servicio de almacenamiento *online* que permite guardar cualquier tipo de archivo en la nube, por ello, se puede acceder en cualquier lugar mediante un dispositivo con conexión a Internet. Además, Google proporciona 15 GB de almacenamiento de manera gratuita y permite la opción de compartir los ficheros con otros usuarios de tal forma que ellos también puedan editarlos, siempre y cuando su propietario les haya dado permisos para ello.

2.4. Google Docs

Pese a que Google Docs se unificó[8] junto con otros servicios web de Google cuando se lanzó el servicio Google Drive, el funcionamiento es el mismo. Este servicio permite editar textos de manera colaborativa además de controlar los accesos a los documentos y manteniendo los menús y los atajos en el teclado para no ser concebido muy diferente a los procesadores de texto ya conocidos por los usuarios como OpenOffice[9] o Microsoft Office[10].

2.5. Google Apps Script

Google Apps Script [6] o Apps Script (de manera simplificada) es un lenguaje de codificación web basado en JavaScript que permite crear complementos y herramientas compatibles con los diferentes servicios y productos de Google y con servicios de terceros y construir aplicaciones web.

Adicionalmente, Google pone a disposición de aquellos usuarios interesados en utilizar Apps Script, una gran cantidad de *tutoriales y guías*[11] para que resulte más sencillo comenzar a utilizar este servicio, así como un gran *glosario*[12] que permite conocer detalladamente las diferentes clases, utilidades y componentes de Apps Script.

2.6. Complementos y Chrome Web Store

Chrome Web Store[13] es un lugar web desde el cual se pueden instalar una amplia gama de complementos y extensiones para utilizarse junto a los servicios proporcionados por Google como Google Docs, Google Sheets, entre otros, y también para instalar temas para el buscador Chrome. Todas estas extensiones son, en su mayoría, gratis y ofrecen muchas posibilidades y funciones.

Los complementos son publicados y añadidos a la Chrome Web Store por los autores y puede realizarse de forma sencilla y guiada gracias a la página *web*[14] que Google ofrece a sus usuarios, detallando los pasos a realizar.

[8] *https://es.wikipedia.org/wiki/Google_Drive*
[9] *https://www.openoffice.org/es/*
[10] *https://products.office.com/es-es/home*
[11] *https://www.google.com/script/start/*
[12] *https://developers.google.com/apps-script/reference/*
[13] *https://chrome.google.com/webstore/category/extensions*
[14] *https://developer.chrome.com/webstore/publish*

2.7. Primer contacto con la herramienta

En la figura 1, se observa que la herramienta consiste en una barra lateral que se crea al lanzar el complemento. En ella se encuentra un desplegable que muestra todos los ficheros encontrados en el servicio Google Drive del usuario conectado, un segundo desplegable para seleccionar el estilo con el que se quiere realizar las citas y dos botones: el de color azul situado en la izquierda es el botón que lanza la combinación de los documentos (el seleccionado en el desplegable mencionado anteriormente y el actual documento abierto) y un segundo botón que sirve para reiniciar los valores de la barra lateral a los valores por defecto.

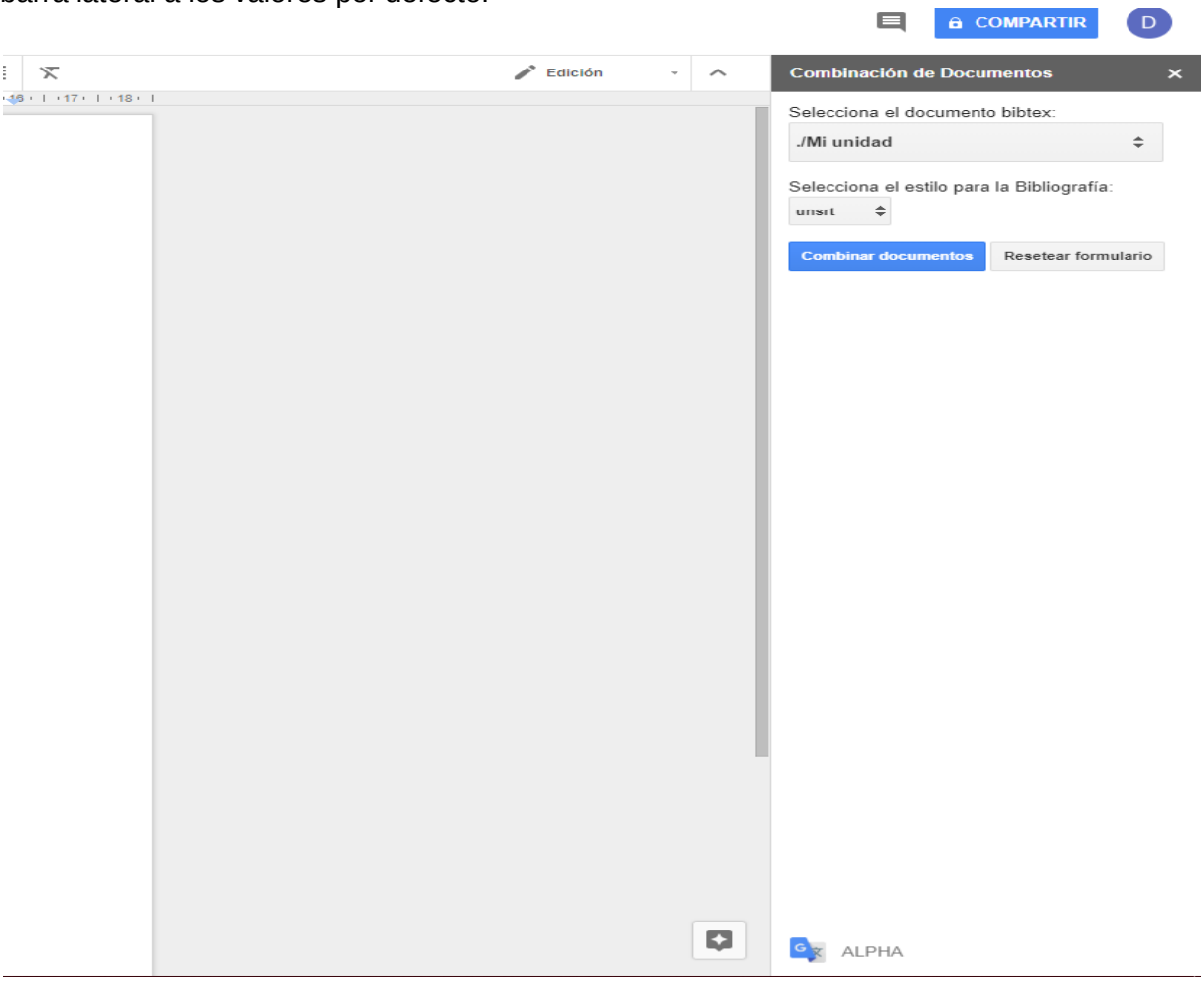


Figura 1: Complemento inicial

El objetivo del complemento inicial es permitir insertar citas en el documento abierto y crear un apartado final de referencias en función de las citas que se han añadido. Para ello se debe seleccionar (ver figura 2) un archivo compatible en el desplegable (extensión .bib) y tener, como mínimo, una cita en el documento abierto como se representa en la figura 3.

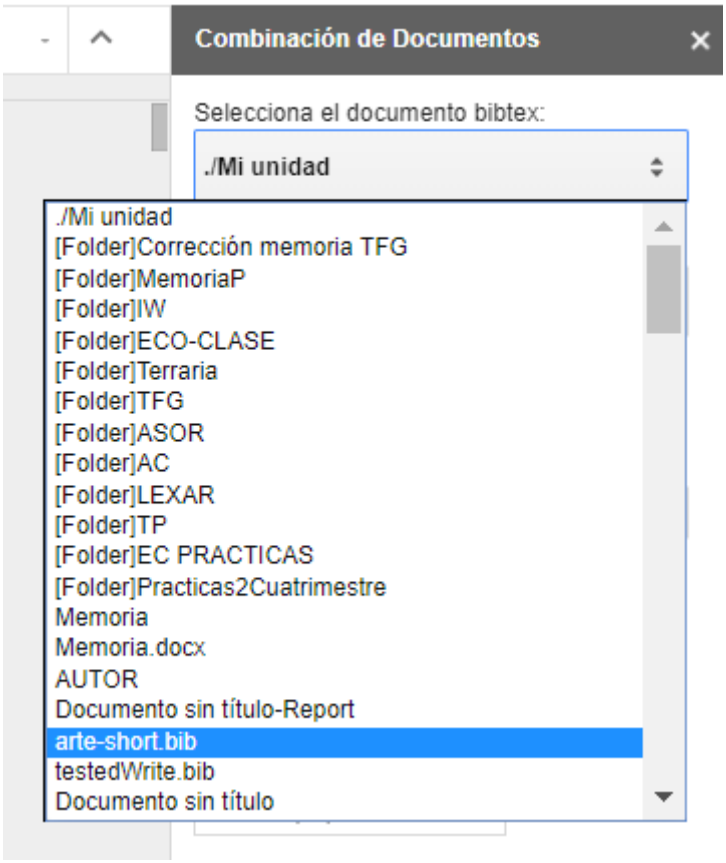


Figura 2: Selección de documento

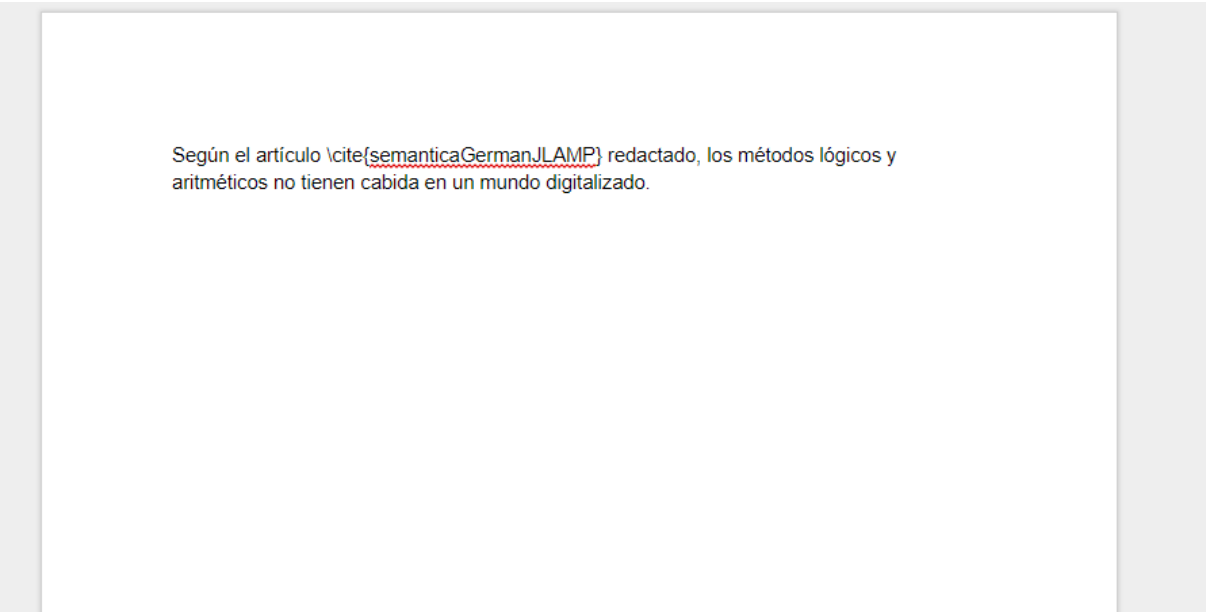


Figura 3: Texto con cita

Una vez seleccionado el archivo de extensión .bib y añadida una cita en el documento, procedemos a pulsar el botón azul con el texto *Combinar documentos*. Se nos mostrará un mensaje en color verde indicando el nuevo archivo creado en nuestro Drive como el que se puede ver en la figura 4. Finalmente, podremos ver en nuestro espacio de almacenamiento Drive, el documento generado (ver figura 5).

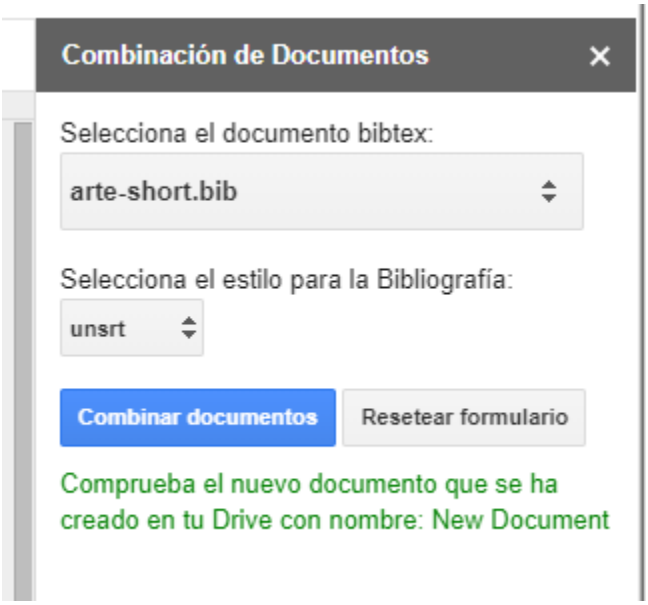
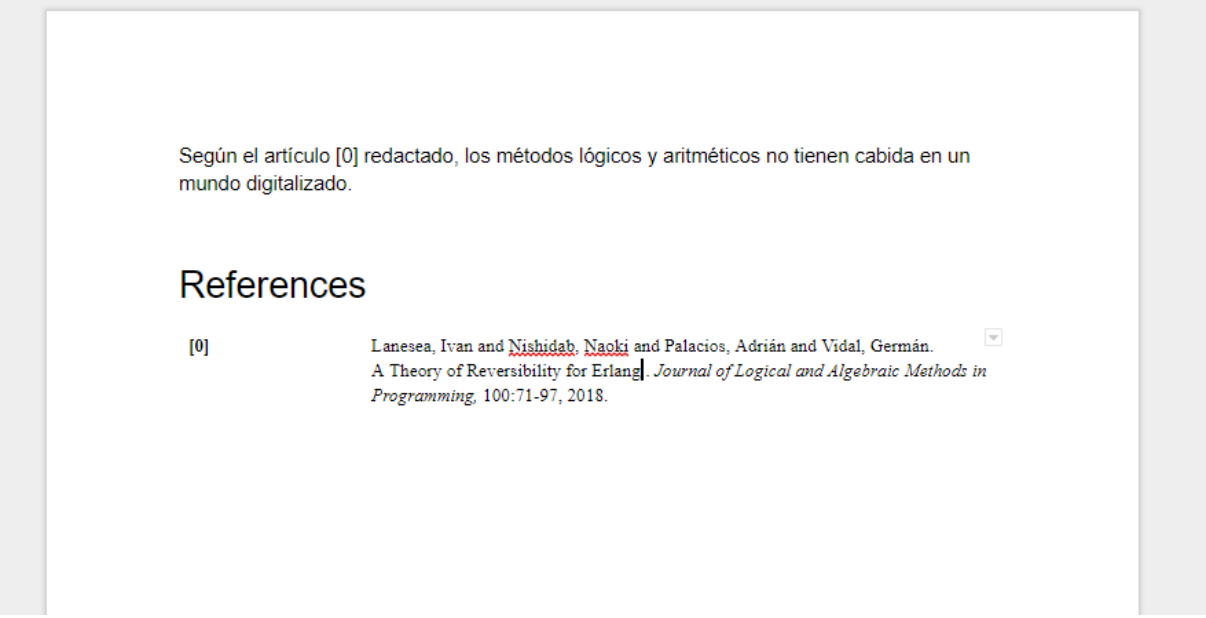


Figura 4: Mensaje de éxito


Según el artículo [0] redactado, los métodos lógicos y aritméticos no tienen cabida en un mundo digitalizado.

References

[0] Lanesea, Ivan and Nishidab, Naoki and Palacios, Adrián and Vidal, Germán. A Theory of Reversibility for Erlang. *Journal of Logical and Algebraic Methods in Programming*, 100:71-97, 2018.

Figura 5: Documento generado

2.8. Herramientas similares

- **RefWorks**[15]

Permite seleccionar entre diversos formatos bibliográficos, así como editarlos o añadir nuevos formatos o previsualizarlos.

En el caso del documento creado para la bibliografía, RefWorks soporta Word para Windows 2000 o posterior, Word para Mac 98 o posterior, OpenOffice.org Writer (.odt), HTML, Rich Text Format (RTF) y Texto.

A partir de ahí se crea la bibliografía en el formato elegido en unos segundos.

RefWorks también contiene una herramienta para citar llamada Write-N-Cite III la cual se puede descargar desde el menú proporcionado por RefWorks, dentro del apartado de 'Herramientas'.

Se escribe un documento de texto normal en el cual se pueden ir añadiendo citas de entre más de 430 disponibles. Además, simplifica la adición de citas mediante un botón *Citar* y permite previsualizarlas mediante el botón *Ver*. También permite generar una bibliografía a partir de este documento generado y escoger su formato bibliográfico.

RefWorks es gratuito y solo se necesita crear una cuenta para poder empezar a utilizarlo. Una vez que se crea una cuenta (o que se asocia a una entidad u organización) se puede, desde crear una base de datos, hasta añadir referencias, que luego se pueden utilizar en la generación de documentos propios. Dichas referencias se pueden organizar, gestionar y, en caso de tener una amplia variedad, se pueden realizar búsquedas detalladas en la base de datos.

- **Paperpile**[16]

Complemento instalable mediante la pestaña de *complementos* de Google Docs.
Permite obtener una versión de prueba gratuita durante 30 días y después se puede obtener la versión **Academic** por un precio de $2,99 al mes o la versión **Business** por $9,99 al mes. Estas dos opciones también brindan la posibilidad de obtener previamente una versión de prueba y sus versiones pagadas incluyen, tal y como indica la página, todas las funcionalidades. El pago, se indica que se factura anualmente, es decir, al año se pagan los 12 meses con un precio u otro dependiendo de la versión adquirida.
Una vez adquirida, el usuario deberá iniciar sesión en Google de manera segura para empezar a utilizar esta extensión. Dado que permite organizar los archivos PDF que encuentra en tu Drive y citar las referencias de tus documentos de Google, habrá que darle permisos de acceso a dicho servicio para que pueda trabajar.

La extensión creará una pestaña lateral a modo de biblioteca que incluirá los archivos que se encuentren en tu Drive y los ordenará mediante una jerarquía. Además, cuenta con la opción de poder modificar este agrupamiento y añadirle etiquetas de colores para mejorar la diferenciación de contenidos.

[15] *https://www.refworks.com/es/*
[16] *https://paperpile.com/*

También incluye un campo de búsqueda para, en caso de librerías muy extensas, poder encontrar un documento, archivo, o referencia de manera más rápida. Dicha búsqueda se realiza de forma dinámica, es decir, a medida que se teclean caracteres va mostrando los resultados más parecidos. Se puede buscar por palabras clave, autores, publicaciones o el año de publicación.

Una vez buscada una referencia, se puede editar su contenido de una manera muy simple ya que sólo hay que hacer *click* en el botón *Edit data*. Las referencias generadas abarcan más de 40 tipos y 86 subtipos para permitir un gran abanico de posibilidades.

Paperpile también permite crear atajos de teclado para poder explotar al máximo sus funcionalidades de una manera más rápida y eficaz.

Cuando se encuentran referencias incompletas, existe una opción que realiza una actualización automática mediante la cual, se realiza una búsqueda exhaustiva en internet para encontrar los datos que falten. Para las referencias duplicadas, se realiza una limpieza mediante un *click*. También se encarga de mantener las abreviaturas actualizadas y correctas.

Así mismo, Paperpile instalará una extensión en Google Chrome para poder añadir referencias de forma simple si se encuentran en una página web.

- **EasyBib**[17]

Este complemento en su versión de prueba (gratuita) permite crear de manera sencilla una bibliografía y citas de libros, artículos y páginas web de forma automática únicamente introduciendo el título o la URL. Soporta los formatos MLA, APA, Chicago, Harvard y otros 7000.

A medida que se va realizando el documento, se pueden añadir las citas necesarias a un menú lateral, el cual es generado por el propio complemento, y una vez que se han incluido todas, mediante un *click* se generará una bibliografía ordenada alfabéticamente y será añadida al final de documento.

Pese a todo, y según los comentarios y puntuaciones obtenidas, EasyBib sólo permite 7 citas en su versión de prueba, lo cual lo limita bastante, además de que los usuarios reportan algunos fallos después de varias utilizaciones. Algunos usuarios incluso recomiendan **Paperpile.**

La versión Pro, cuesta $9,99 al mes. EasyBib Pro además es multiplataforma, es decir, que se integra de buena manera tanto en ordenador, como en *tablet* y dispositivos móviles, únicamente es necesaria una conexión a Internet.

- **F1000**[18]

Extensión para Google Chrome gratuita que se identifica como la “mejor herramienta de citado en Google Docs”. Cuenta con más de 7000 estilos para crear bibliografías y dar formato a las citas. F1000 también proporciona sugerencias inteligentes y permite

[17] *http://www.easybib.com/*
[18] *https://f1000.com/*

realizar una búsqueda en PubMed y en una base de datos propia que contiene más de 170.000 recomendaciones expertas de artículos, todo dentro de Google Docs, sin tener que abandonar la pestaña.

- **Explore**

Pese a que la herramienta Explore se lanzó para Google Docs, Sheets y Slides como una utilidad para mejorar y agilizar el proceso de creación de dichos proyectos, no fue hasta tres meses después de su lanzamiento cuando se introdujo una mejora para permitir realizar citaciones a partir de las búsquedas realizadas mediante la propia herramienta. Sobre todo, para "estudiantes que tengan que escribir investigaciones o análisis, y también para aquellos que sólo quieran verificar sus fuentes de información" permitiendo, mediante un sólo *click*, añadir citaciones a modo de pie de página. Esta herramienta es una extensión del propio G Suite[19], una suite/agrupación de herramientas online completamente web y con muchos aspectos centrados en su uso empresarial.

Una vez analizadas dichas herramientas, se puede observar que algunas ventajas respecto a muchos complementos del mercado, pueden ser que esta herramienta es gratuita, sin ningún tipo de mejora a una cuenta *premium* que ofrezca más servicios, se ofrece todo y para todos sin necesidad de crear una cuenta (excepto en los servicios de Google, pero eso es necesario para tener acceso al propio servicio de Drive) y tampoco se utilizarán los datos personales ni se conectará con servicios externos.

Por otro lado, es verdad que al no haber sido desarrollada durante más tiempo hay muchas desventajas respecto a los complementos ya existentes como no permitir servicio multiplataforma, no posee una gran variedad de estilos, solamente funciona con el servicio Google Docs y no tiene una funcionalidad que permita buscar referencias en la web ni provee de sugerencias.

[19] *https://gsuite.google.com/apps-show/#/*

3. DESARROLLO

En este capítulo se comenzará explicando la estructura del proyecto y se presentarán los cambios realizados y las funciones nuevas que se han añadido acompañadas de una explicación e imágenes para su comprensión.

3.1. Estructura del proyecto

Antes de presentar la implementación del complemento se debe profundizar en la estructura que se ha utilizado. Como se indicó en la sección 1.3 (Plan de trabajo), la implementación de las mejoras se ha realizado mediante el entorno de desarrollo que proporciona Google y que se representa en la figura 6. Así mismo, la distribución del proyecto también se encuentra en esta ventana de desarrollo ampliada en la figura 7.

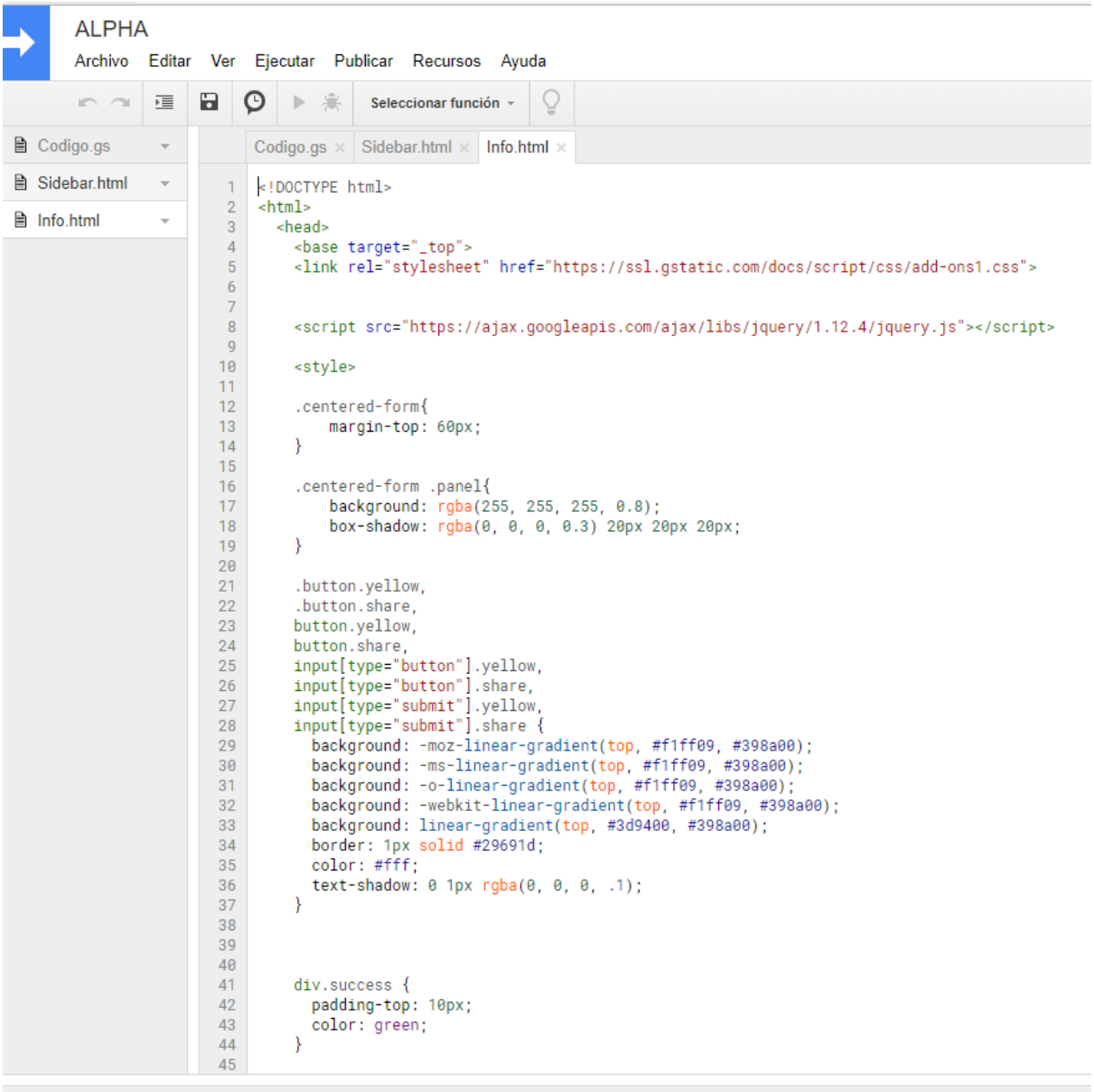


Figura 6: Ventana de desarrollo

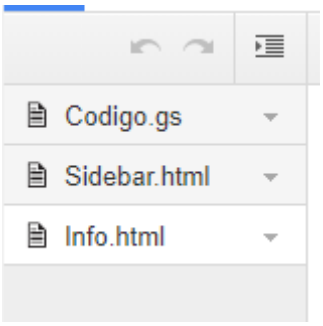


Figura 7: Distribución del proyecto

Dentro del proyecto hay que destacar como partes más relevantes los siguientes apartados:

- Codigo.gs, el archivo principal de código que contiene todas las funciones utilizadas en el proyecto, tanto las antiguas como las nuevas, además de la definición de clases y objetos necesarias para la correcta construcción del proyecto.

- Sidebar.html, contiene todos los recursos necesarios para generar la barra lateral que permite interactuar con el complemento y mostrar mensajes tanto de éxito como de error junto con la llamada de generación de la ventana *Añadir entrada*.

- Info.html, donde se especifican los elementos que se añadirán a la ventana *Añadir entrada*, así como los mensajes de éxito y de error.

Las modificaciones en las que se basa dicho proyecto, se realizaron sin atender a ningún método de trabajo organizativo específico. Dichas modificaciones se llevaron a cabo a medida que ampliaba mis conocimientos del lenguaje Google Apps Script y, al ser consideradas como modificaciones independientes unas de las otras, no hubo que pensar en casos de colisión entre dichas modificaciones.

Únicamente llevé a cabo una planificación consistente en realizar una nueva modificación cada 8 o 9 días, de tal forma que se empleaban 7 días para realizar una implementación de la funcionalidad, y los restantes servían para realizar pruebas en busca de fallos y aplicar correcciones. Adicionalmente, se disponía de unos días más, al finalizar el mes, para realizar una prueba exhaustiva del complemento y para comprobar que la funcionalidad inicial no había sido deshabilitada o modificada.

Así mismo, se concertaron reuniones con los directores del proyecto, para observar el avance realizado y proponer soluciones a los problemas que se fueran encontrando durante la implementación.

3.2. Hito 1

La idea general del primer hito era habilitar una selección múltiple de ficheros .bib e implementar la generación de un informe de referencias que, en una primera instancia, contuviese todas las citas especificadas en los ficheros .bib seleccionados.

A partir de ese punto, decidí empezar por implementar la selección de un segundo fichero .bib mediante un desplegable separado (ver figura 8) para comprobar si las funciones que se empleaban eran reutilizables. El desplegable introducido en la interfaz de la herramienta se observa en la figura 9.

```
<div class="block form-group">
  <label for="select_file">Selecciona otro documento bibtex:</label>
  <br>
  <select id="select_file2" name="select_file2"></select>
  <p><b id="tittle" ></b></p>
  <p id="id"></p>
</div>
```

Figura 8: Nuevo desplegable

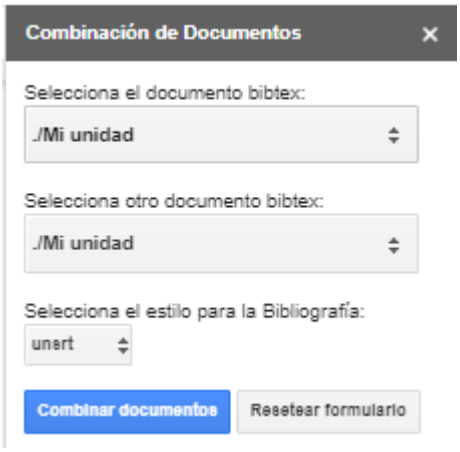


Figura 9: Nuevo desplegable en la barra lateral

Pese a que se comprobó que el código era reutilizable y no presentaba problemas a la hora de citar un documento utilizando un segundo fichero .bib, se consensuó con los directores del proyecto una mejora del apartado para poder utilizar un número indefinido de ficheros .bib. Para ello se implementó una selección múltiple (ver figura 10) que mostraría los documentos seleccionados a medida que se fuesen incluyendo de tal forma que el resultado se aprecia en la figura 11.

```
function selectFile(select){
  var $ul = $(select).prev('ul');
  var disp = document.getElementById('select_file');
  var text = disp.options[disp.selectedIndex].text;

  if(text.indexOf("Folder") > -1|| text.indexOf("../") > -1){return;}
  else{


    if ($ul.find('input[value=' + $(select).val() + ']').length == 0)
      $ul.append('<li onclick="$(this).remove();">' +
        '<input type="hidden" name="select_file[]" value="' +
        $(select).val() + '" /> ' +
        text + '<input type="equis">'+'</li>');
  }
}
```

Figura 10: Inclusión de una lista dinámica de ficheros seleccionados

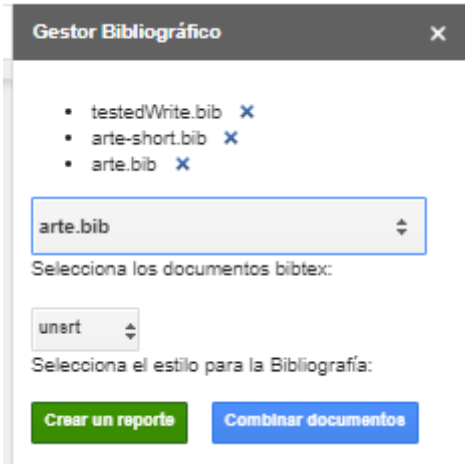


Figura 11: Apariencia de la lista dinámica

También se añadió un símbolo en forma de aspa que permite eliminar de la lista un fichero seleccionado, de tal forma que se contemplase el error de selección por parte del usuario.

Durante la implementación de la lista de selección múltiple, se implementó un nuevo botón que permitiese la generación del reporte de referencias, manteniendo las funcionalidades que poseía previamente el complemento como se aprecia en la figura 12.

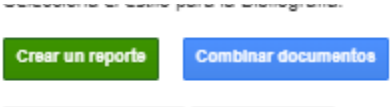


Figura 12: Botón para generar el reporte

Al pulsar dicho botón, se realiza una llamada desde el lado del cliente a la función *openDocReport()* la cual recopila los ficheros .bib seleccionados previamente así como los filtros que se hayan decidido aplicar (ver figura 13) y que se explicarán en el Hito 2.

```
function openDocReport(){
  $("#success").remove();
  this.disabled = true;
  $('#error').remove();

  var style = $('#select_style').val();



  var docsBib = [];

  var files = document.getElementsByName("select_file[]"), i;
  for(i=0; i<files.length; i++){
    docsBib[i] = files[i].value;
  }
  var option = 2; //Opción 2 = Mostrar un reporte
```

Figura 13: Extracto de código para generar un reporte

Una vez que se han recopilado los ficheros .bib y los filtros, se utiliza la API asíncrona de JavaScript *google.script.run* para realizar una llamada a la función *getBibtexAndDoc()* del lado del servidor pasando como parámetros los ficheros .bib, los filtros y los campos *style* y *option*, los cuales indican el estilo que tomará la bibliografía o el reporte y un valor numérico que indica si se ha seleccionado citar y añadir un apartado de referencias (*option = 1*) o crear un reporte (*option = 2*), respectivamente.

A la utilización de la API *google.script.run* se le suman los métodos *withSuccessHandler()*, el cual define una función a llevar a cabo cuando el servidor lleva a cabo el proceso correctamente, el método *withFailureHandler()* para definir una acción a realizar en el caso de que el proceso que se ejecuta en el lado del servidor falle y el método *withUserObject()* para establecer un segundo objeto que se utiliza como parámetro a los métodos anteriores como se puede ver en la figura 14.

```
  google.script.run
  .withSuccessHandler(
    function(exitoBibtex, element) {

      if(exitoBibtex == false ){
        var msg = "Hubo algun error durante el proceso. Es posible que en tu .bib falte algún campo obligatorio en algu
        showError(msg, $('#button-bar'));
      }
      else if(exitoBibtex.indexOf("-XÇXÜ-") > -1){
        exitoBibtex = exitoBibtex.slice(6);
        var msg = "Se encontró un error en la cita con ID: "+exitoBibtex;
        showError(msg, $('#button-bar'));
      }
      else{
        var test = "si";
        var msg = "Comprueba el nuevo documento que se ha creado en tu Drive con nombre: "+exitoBibtex+"-Report";
        showSuccess(msg, $('#button-bar'));
      }

      element.disabled = false;

  })
  .withFailureHandler(
    function(error, msg, element) {
    msg = "Hubo un problema durante el proceso.Comprueba que hayas seleccionado el documento correcto."+error;
    showError(msg, $('#button-bar'));
    element.disabled = false;
  })
  .withUserObject(this)
  .getBibtexAndDoc(docsBib,style,filtros,option);

}

function nombreDocumento(name){
  alert(name);
}
```

Figura 14: Llamada al lado del servidor y uso del valor devuelto

Una vez que el servidor comienza la ejecución de la función (ver figura 15), se crea un diccionario que contendrá la información de todas las citas. Los documentos que se han seleccionado, son convertidos a objetos de la clase BibTex, cuya definición se obtuvo *aquí*[20].

Después se comprueba que las entradas de cada documento contienen los campos obligatorios y que no hay fallos en ellos mediante la función *checkErrors()* que, a su vez, utiliza métodos propios de la clase BibTex para realizar las comprobaciones correspondientes como se observa en la figura 16.

En el caso de que haya error, *checkErrors()* devolverá un número entero mayor que 0, el cual indicará la posición de la cita en la que se ha encontrado un error para que el servidor pueda obtener el ID de la cita en función de su posición en el diccionario y devolverlo para generar el mensaje de error correspondiente a dicha cita. Además, antes de volver al lado del cliente, al ID de la cita que se ha obtenido, se le añade una cadena identificativa de error, en este caso *"-XÇXÜ-"* compuesto por valores que Google Drive no admite para los nombres de archivos.

Este proceso se lleva a cabo para que el cliente identifique el parámetro devuelto como un error ya que cuando hay éxito, se utiliza el mismo canal para enviar el nombre del documento actual, que se utilizará en el mensaje de éxito y sin la cadena identificativa, no reconocería un error si no un nombre de documento y mostraría un mensaje de éxito en lugar de el de fallo.

[20] https://github.com/select/google-docs-bibtex-cite/blob/master/bibtex-cite.js

```
function getBibtexAndDoc(docsBib, estilo, filtros, option){

  var bibtex_dict = [];
  var documentos = []; //array de docs de Drive
  for(i=0; i<docsBib.length; i++){
    documentos[i] = DriveApp.getFileById(docsBib[i]);
  }

  var objetosBib = []; //array de objetos de la clase BibTex
  for(i=0; i<documentos.length; i++){
    objetosBib[i] = new BibTex();
    objetosBib[i].content = documentos[i].getBlob().getDataAsString();
    objetosBib[i].parse();
  }

  var exito = true;
  for(i=0; i<objetosBib.length; i++){
    var errorEncontrado = checkErrors(objetosBib[i]);
    if(errorEncontrado >= 0){

      exito = errorEncontrado;
      var codError = "-XÇXÜ- ";
      var falla = codError.concat(objetosBib[i].data[exito].cite);
      return falla;

    }
  }

  if(exito == true){
    var biblioExist = checkBibliography();
    var reportExist = checkReport();
  }
  if((exito == true && biblioExist) || (exito == true && reportExist)){

    var aux = 0;
    var allIdCitas = [];
    var index = 0;
```

Figura 15: Extracción de código del lado del servidor

```
//COMPRUEBA SI EN TODOS LOS DOCUMENTOS ENCONTRADOS HUBO EXITO EN SU CREACIÓN (NO FALTA NINGÚN CAMPO OBLIGATORIO)

function checkErrors(bibtex){
  var i = 0;
  var errorEncontrado = false;
  while(i<bibtex.data.length && !errorEncontrado){
    var outobj = bibtex.google(i);
    if(outobj['exito'] === false){
      errorEncontrado = true;
      return i;
    }


    i++;
  }
  return -1;
}
```

Figura 16: Llamada a la función *checkErrors()*

Una vez que se ha comprobado que no hay fallos en las entradas de ningún documento, se procesa de manera individual de tal forma que se añadan al *array* (ver figura 1) que compondrá el reporte únicamente si cumplen las condiciones de los filtros (en caso de que las haya) que se analizarán más adelante en el Hito 2.

```
for(i=0; i<objetosBib.length; i++){
  //Para documento, lo añadimos al diccionario
  var Objeto = new BibTex();
  Objeto = objetosBib[i]; //Objeto contiene todas las entradas del documento .bib correspondiente


  for(x=0; x<Objeto.data.length; x++){ //Para cada entrada del Objeto

      var autores = [];
      autores = Objeto.data[x].author;


      if(compruebaYear(Objeto.data[x].year, filtros[1]) && compruebaTipo(Objeto.data[x].entryType, filtros[0]) &&
      compruebaAutor(autores, filtros[2]) && compruebaSeries(Objeto.data[x].series, filtros[3]) && compruebaEditorial(Objeto.data[x].publisher, filtros[4])){

        var outobj = Objeto.google(x);
        bibtex_dict[Objeto.data[x].cite] = outobj;
        allIdCitas[index] = Objeto.data[x].cite;

        index++;
      }

  }

}
```

Figura 17: Procesado de cada entrada

A continuación, y como se observa en la figura 18, se obtiene el ID del documento abierto para copiarlo y renombrarlo y se invoca al método *sustitute()* o al método *report()* en función del valor de si se solicita crear un apartado de referencias o generar un reporte, indicado por el valor del parámetro *option.*

```
    if(option == 1){
      var arrayCites = getCites();
      var arrayCitesId = getId(arrayCites);
    }

    var idDoc = DocumentApp.getActiveDocument().getId(); //id del documento abierto

    var file = DriveApp.getFileById(idDoc); //archivo que representa el doc abierto

    if(option == 1){
      var newIdDoc = file.makeCopy("New Document").getId(); //copia del documento, renombrandolo y obteniendo su id

      var doc = DocumentApp.openById(newIdDoc);
      var body = doc.getBody();
      var texto = body.getText();
      exito = sustitute(arrayCites, arrayCitesId, body, bibtex_dict, doc, estilo);
    }
    else if(option == 2){
      var nombre = file.getName();
      var newIdDoc = file.makeCopy(nombre+"-Report").getId(); //copia del documento, renombrandolo y obteniendo su id

      var doc = DocumentApp.openById(newIdDoc);
      var body = doc.getBody();
      var texto = body.getText();

      exito = report(allIdCitas, bibtex_dict, doc, estilo, body);

      if(exito){
        var joker = DocumentApp.getActiveDocument();
        var harley = joker.getName();
        return harley;
      }
    }

  return exito;
}
```

Figura 18: Llamada a la creación del reporte.

La función *report()* recibirá los parámetros:

- **allIdCitas**, un *array* de todas las citas que han superado los filtros y, por tanto, deben ser incluídas en el reporte generado.
- **bibtex_dict** es el diccionario con la información de todas las citas.
- **doc** representa el nuevo documento (una copia renombrada del que el usuario tiene abierto).
- **estilo** que indica el formato de las citas en el reporte generado.
- **body2** que se identifica como el cuerpo del nuevo documento descrito anteriormente y sobre el cual se añadirá el reporte generado.

Dentro de la función *report()* se definirá el *array listaTuplasReporte* cuyos valores serán objetos (ver figura 19), con los atributos:

- **cite** cuyo valor indicará el ID de una entrada.
- **info** la cual tendrá asociado como valor la información de la entrada correspondiente del diccionario que se ha pasado como parámetro a la función.

```
function report(allIdCitas, bibtex_dict, doc, estilo, body2){
  /* lista para el reporte */
   var listaTuplasReporte = []; //Información de todas las citas que hayan pasado los filtros
   var exito = true;

   for(i=0; i<allIdCitas.length; i++){
     var ClaveUnitaria = allIdCitas[i];
     listaTuplasReporte[i] = {};
     listaTuplasReporte[i].cite = ClaveUnitaria;
     listaTuplasReporte[i].info = bibtex_dict[ClaveUnitaria];

   }

   /* SECCIÓN PARA EL REPORTE DE LAS CITAS */

  var exitoReporte = setReport(body2, listaTuplasReporte, estilo);

  doc.saveAndClose();

  return exito;
}
```

Figura 19: Función *report()*

Seguidamente se llamará a la función *setReport()* la cual buscará la cadena *\report* en el documento nuevo, devolviendo error si no la encuentra e insertando un párrafo en blanco en el lugar donde se encuentra la cadena eliminándola del texto, en caso de que la encuentre como se representa en la figura 20.

```
//Insertamos el informe en donde aparezca el \report

function setReport(body, listaTuplasReporte, estilo){

  /* PÁRRAFO PARA EL INFORME/REPORTE */

  var rangeElem = body.findText("\\\\report");
  var exito;

  if(rangeElem === null){ //No se encontró \report en el texto
    exito = false;
  }else{
    exito = true;
    var elem = rangeElem.getElement(); //Cogemos el elemento
    var parent = elem.getParent(); //Cogemos al padre
    var index = parent.getParent().getChildIndex(parent); //Índice del hijo

    var style = {};
    style[DocumentApp.Attribute.FONT_SIZE] = 10;
    style[DocumentApp.Attribute.BOLD] = true;

    var miReport = body.insertParagraph(index, "Report");

    index++;
    miReport.setHeading(DocumentApp.ParagraphHeading.HEADING1);


    var arrayCitesInserted = constructReporte(listaTuplasReporte, estilo, body, index);

    elem.removeFromParent();
  }
  return exito;
}
```

Figura 20: Función *setReport()*

A su vez, llamará a la función *constructReporte()* que se encargará de definir los estilos de texto en cursiva, en formato normal y en negrita, así como de definir la tabla y rellenar las filas con los datos correspondientes a cada una de las entradas válidas (ver figura 21), teniendo en cuenta el estilo que se ha seleccionado, a la hora de incluir los datos.

```
function constructReporte(listaTuplasReporte, estilo, body, index){

  /* --------------------- Estilos ------------------------ */

  var boldStyle = {};
  boldStyle[DocumentApp.Attribute.FONT_SIZE] = 10;
  boldStyle[DocumentApp.Attribute.BOLD] = true;
  boldStyle[DocumentApp.Attribute.ITALIC] = false;
  boldStyle[DocumentApp.Attribute.FONT_FAMILY] = "Times New Roman";

  var italicStyle = {};
  italicStyle[DocumentApp.Attribute.FONT_SIZE] = 10;
  italicStyle[DocumentApp.Attribute.BOLD] = false;
  italicStyle[DocumentApp.Attribute.ITALIC] = true;
  italicStyle[DocumentApp.Attribute.FONT_FAMILY] = "Times New Roman";

  var normalStyle = {};
  normalStyle[DocumentApp.Attribute.FONT_SIZE] = 10;
  normalStyle[DocumentApp.Attribute.BOLD] = false;
  normalStyle[DocumentApp.Attribute.ITALIC] = false;
  normalStyle[DocumentApp.Attribute.FONT_FAMILY] = "Times New Roman";


  /* ----------------- Construcción de la Tabla ---------------------- */

  var table = body.insertTable(index);
  table.setBorderWidth(0);
```

Figura 21: Extracción de la función *constructReport().*

Durante el proceso de definición de la tabla, se llama a la función *contructName()* y a la función *constructObj()*, observado en la figura 22, que se encargarán de generar el nombre identificativo para el reporte y de ordenar los campos de la entrada (autor, titulo, año, volumen, páginas, etc.) y de aplicar los correspondientes formatos de acuerdo al estilo que se utiliza, respectivamente.

El resultado de las funciones *contructName()* y *constructObj()* se introducirán en la tabla, la cual había sido dividida durante su definición en dos columnas.

```
obj.name = constructName(contador,listaTuplas[i],estilo);
obj.cite = listaTuplas[i].cite;

contador++;

var posArray = arrayCitesInserted.length;
arrayCitesInserted[posArray] = obj;

var tr = table.appendTableRow();

for(var j=0; j<2; j++){

  var td = tr.appendTableCell();
  var paraInCell = td.getChild(0).asParagraph();   //párrafo del la celda

  if(j === 0){                                              //celda1
    paraInCell.appendText("[" + obj.name + "]");
    paraInCell.setAttributes(boldStyle);
  }else if(j == 1){                                        //celda2

    var finalObj = constructObj(listaTuplas[i].info);

    for(var key in finalObj){
      if((key == "title" && listaTuplas[i].info.entryType != "article") || (key == "journal") || (key == "booktitle")){
        var text = paraInCell.appendText(finalObj[key]);
        text.setAttributes(italicStyle);
      }else{
        var text = paraInCell.appendText(finalObj[key]);
        text.setAttributes(normalStyle);
      }
    }
  }
}
```

Figura 22: Llamadas a las funciones *constructName()* y *constructObj()*.

Finalmente, en la primera columna se introducirá el nombre asociado a la entrada y que es el resultado devuelto por la función *constructName(),* representada en la figura 23, mientras que en la segunda columna se introducirá la información ordenada y con el formato adecuado devuelto por *constructObj()*, mostrada en la figura 24. Este proceso se realizará de forma iterativa dando lugar a una tabla con tantas filas como entradas válidas hubiese.

```
function constructName(pos,tupla,estilo){
  var name = "[";
  switch(estilo){
    case "abbrv":

      break;
    case "acm":

      break;
    case "alpha":

      break;
    case "plain":

      break;
    case "apalike":

      break;
    case "unsrt":
      name = pos;
      break;
  }

  return name;
}
```

Figura 23: Función *constructName()*.

Como también se aprecia en la Figura 23, actualmente hay soporte para los estilos *abbrv, acm, alpha, plain* y *apalike* pero únicamente está implementado el último de la propia figura, el estilo *unsrt.*

```
function constructObj(info){
  switch(info.entryType){
    case 'manual':
      /*
      Required fields: title.
      Optional fields: author, organization, address, edition, month, year, note.
      */
      var obj = {authors: "", title: "", organization: "", address: "", edition: "", month: "", year: "", note: "", entryType: "manual"};

      break;
    case 'unpublished':
      /*
      Required fields: author, title, note.
      Optional fields: month, year.
      */
      var obj = {authors: "", title: "", note: "", month: "", year: "", entryType: "unpublished"};

      break;
    case 'booklet':
      /*
      Required fields: title.
      Optional fields: author, howpublished, address, month, year, note.
      */
      var obj = {authors: "", title: "", howpublished: "", address: "", month: "", year: "", note: "", entryType: "booklet"};

      break;
    case 'conference':
      /*
      Required fields: author, title, booktitle, year.
```

Figura 24: Extracción de la función *constructObj()*.

Antes de finalizar la ejecución del método *constructObj()*, se realiza una llamada a la función *finishInfo()* y a la función *addSeparators()*, como se muestra en la figura 25, encargadas de añadir la definición del tipo de entrada y comprobar si hay valores no permitidos, como también de eliminar los campos que no tengan un valor, añadir los separadores adecuados entre campos así como revisar las tildes y símbolos, respectivamente. En la figura 26 se observa el funcionamiento del método *finishInfo()* y en la figura 27 el funcionamiento de *addSeparators()*.

```
    }else{
      if(key in obj){
        obj[key] = info[key];
      }
    }
  }

  var nextObj = finishInfo(obj);

  var finalObj = addSeparators(nextObj);
  return finalObj;
```

Figura 25: Extracción de la función *constructObj()* con llamada a *finishInfo()* y *addSeparators()*

```
function finishInfo(obj){
  switch(obj.entryType){

    case "mastersthesis":

      if(obj.type == ""){
        obj.type = "Master´s thesis";
      }

    break;
    case "phdthesis":

      if(obj.type == ""){
        obj.type = "PhD thesis";
      }

    break;
    case "techreport":

      if(obj.type == ""){
        if(obj.number == ""){
          obj.type = "Technical Report";
        }else{
          obj.type = "Technical Report " + obj.number;
          delete obj['number'];
        }
      }

    break;
    case "proceedings":
```

Figura 26: Extracción de la función *finishInfo()*

```
function addSeparators(obj){
  for(var key in obj){
    if(obj[key] == ""){
      delete obj[key];   //elimino aquellas propiedades que no tienen un valor
    }
  }

  delete obj["entryType"];

  var size = Object.keys(obj).length;
  var cont = 1;

  for(key in obj){

    if((key == "authors") || (key == "title") || (key == "year")){
      obj[key] = obj[key] + ". "; //autores y titulos separados por un punto
      cont++;
    }else if(cont == size){
      obj[key] = obj[key] + ".";  //es la ultima propiedad -> añadir el punto final
    }else if(key == "month"){
      obj[key] = obj[key] + " ";  //el mes va seguido del año separado por un espacio
      cont++;
    }else{
      obj[key] = obj[key] + ", "; //el resto separados por una coma
      cont++;
    }

    obj[key] = checkTildes(obj[key]);
    obj[key] = checkSimbols(obj[key]);

  }

  return obj;
}
```

Figura 27: Función *addSeparators()*

Todo el proceso se observa en el diagrama de secuencia de la figura 28A y 28B, en el cual se aprecian los métodos y pasos realizados durante el proceso del hito 1, consistente en la generación del reporte, en el cual también se indica que el proceso de filtrado (se explicará en el hito 2) se ejecuta durante el proceso de generación del reporte, así como los casos en los que se devuelve error o no dependiendo del resultado final de la ejecución.

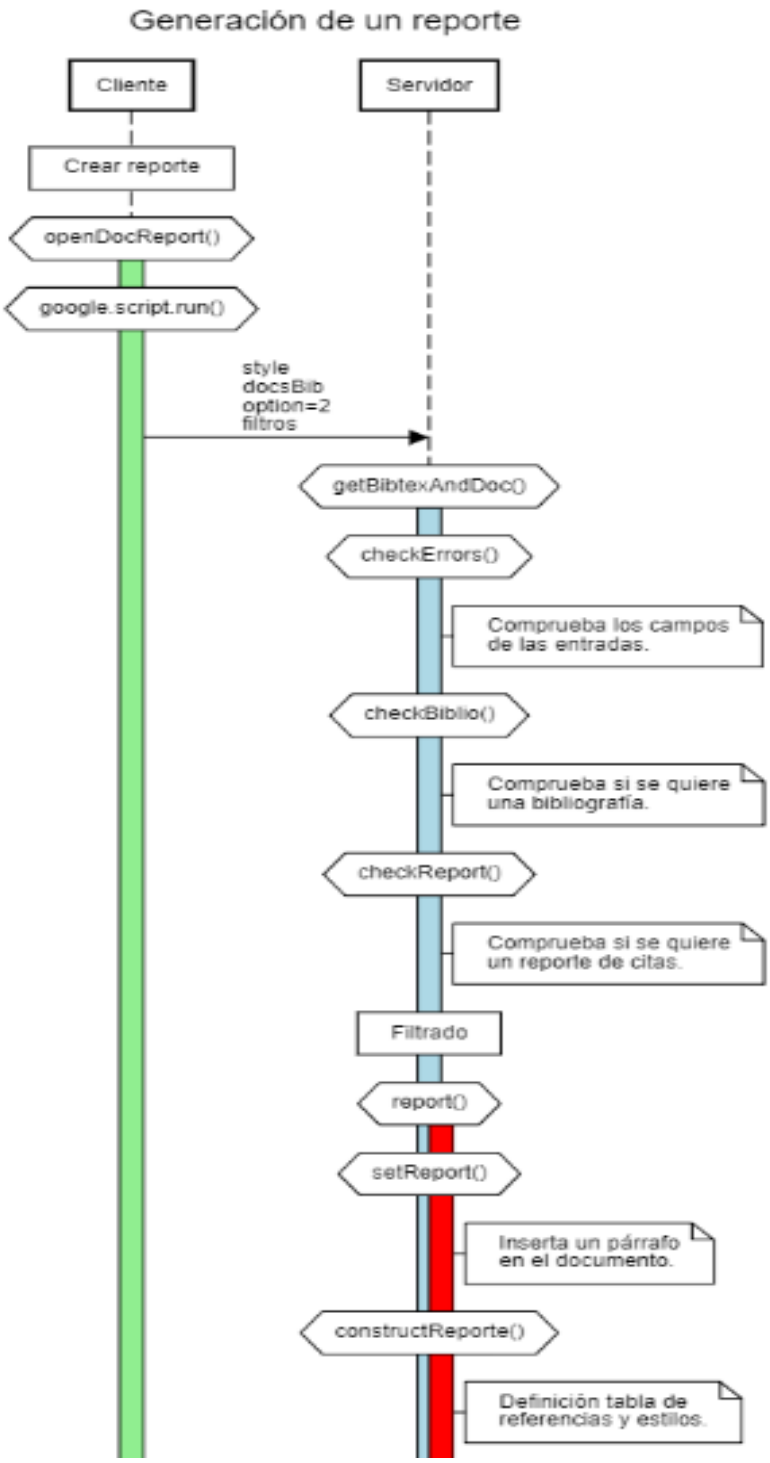


Figura 28A: Generación de un reporte

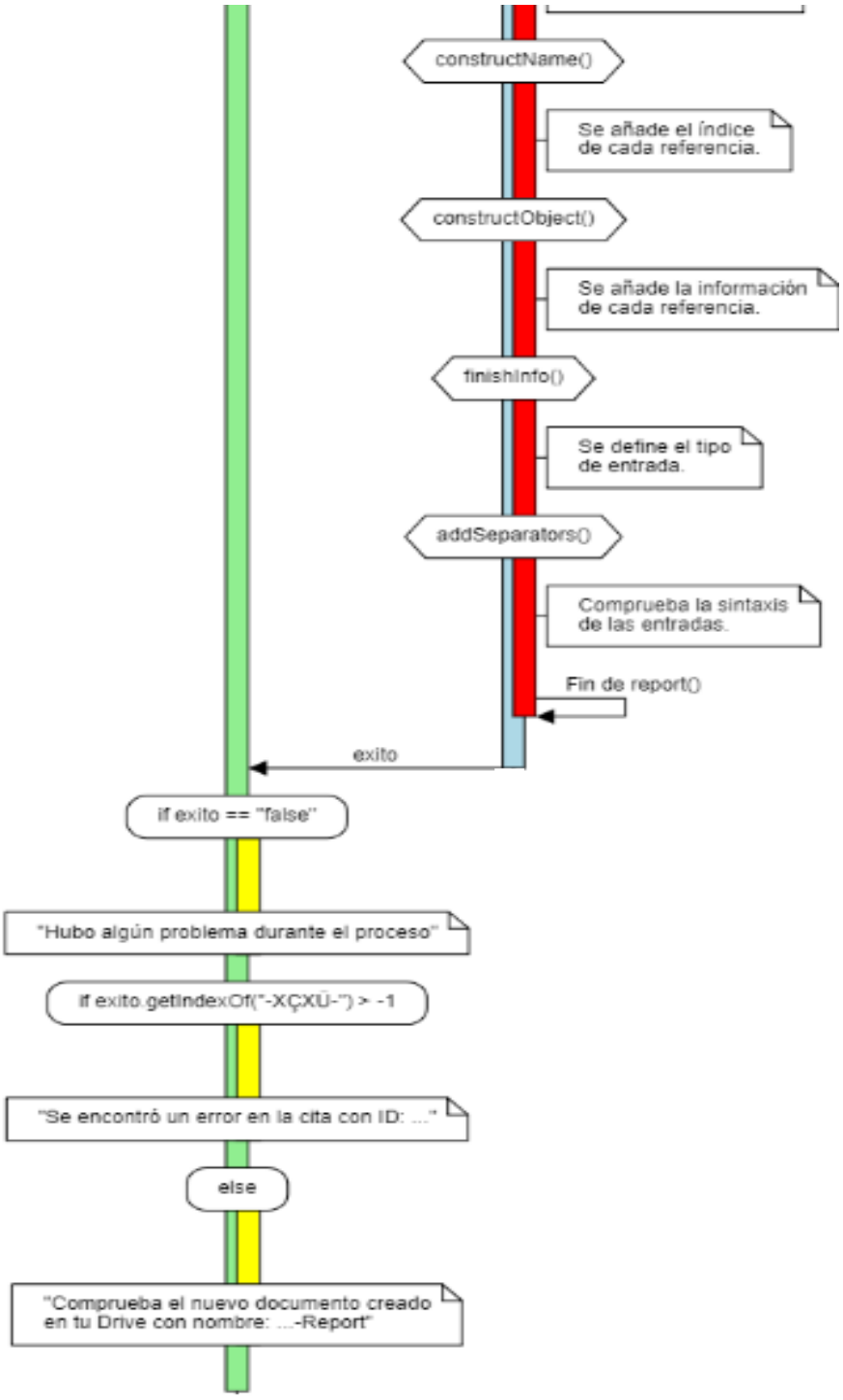


Figura 28B: Generación de un reporte

3.3. Hito 2

Como segundo hito, y valiéndonos de la nueva funcionalidad implementada con el primero, se decidió proveer al usuario de la capacidad de filtrar las citas que aparezcan en el reporte utilizando para ello campos de texto (ver figura 29) donde se incluyen los filtros que se desean aplicar y realizando el mismo proceso para generar un reporte.

Cuando el usuario escribe en los campos proporcionados una cadena para ser utilizada como filtro, dicha cadena sigue el mismo proceso que los documentos elegidos para generar el reporte.

Figura 29: Campos de filtrado.

Los filtros se aplican a los siguientes campos:

- **Tipo de entrada** de tal forma que los tipos que se escriban serán los únicos que se muestren, por ejemplo, **article** o **book** pero también se permite una concatenación de valores mediante comas, por ejemplo, **phdthesis, mastersthesis, booklet**. Este filtro se aplica mediante la función *compruebaTipo()* representada en la figura 30.

```
function compruebaTipo(tipoBib, dato){
  if(dato && tipoBib){
    if(dato.indexOf(",") > -1){ //Varios tipos
      var tipos = dato.split(",");
      for(i=0; i<tipos.length; i++){
        tipos[i] = tipos[i].trim();
        if(tipoBib == tipos[i]){
          return true;
        }
      }
      return false;
    }
    else{ //Sólo un tipo
      dato = dato.trim();
      if(tipoBib == dato){
        return true;
      }
      return false;
    }
  }
  return true;
}
```

Figura 30: Filtro de tipo de entrada

- **Año o rango de años,** el usuario puede especificar un único año, por lo que solamente se incluirán en el reporte aquellas citas cuyo campo *year* sea igual al dato proporcionado, un rango de años utilizando un guión "-" como separador, por ejemplo, **1998-2000**, de manera que sólo se incluirán las citas cuyos campos *year* estén incluidos en ese intervalo. También se pueden especificar ambas, por ejemplo, **2018, 1998-2015**, por lo que sólo se incluirán las citas cuyos campos *year* sean iguales a **2018** o esté comprendido en el intervalo de **1998** a **2015**. Para la aplicación de este filtro, se implementó la función *compruebaYear()* que se puede apreciar en la figura 31.

```
function compruebaYear(yearBib, dato){

  if(dato && yearBib){
    if(dato.indexOf(",") > -1){ //Hay más de un año/franja
      var subFiltro = dato.split(",");
      for(i=0; i<subFiltro.length; i++){
        if(subFiltro[i].indexOf("-") > -1){ //tenemos una franja
          var rango = subFiltro[i].split("-"); //separamos la franja
          rango[0] = rango[0].trim();
          rango[1] = rango[1].trim();
          if(yearBib >= rango[0] && yearBib <= rango[1]){
            return true;
          }
        }
        else{ //tenemos un año
          subFiltro[i] = subFiltro[i].trim();
          if(subFiltro[i] == yearBib){
            return true;
          }
        }
      }
      return false;
    }
    else{ //Sólo hay un año/franja
      if(dato.indexOf("-") > -1){ //Franja única
        var rango = dato.split("-");
        rango[0] = rango[0].trim();
        rango[1] = rango[1].trim();
        if(yearBib >= rango[0] && yearBib <= rango[1]){
            return true;
        }
        return false;
      }
      else{ //Año único
        dato = dato.trim();
        if(dato == yearBib){
          return true;
        }
        return false;
      }
    }
   }
   return true;
}
```

Figura 31: Filtro de año o rango de años

- **Autores** siendo este el filtro más complejo. Se permiten nombres de autores, por ejemplo, **David**, así como nombres y apellidos, **David Hervás Rodríguez**. A partir de este punto, el usuario puede filtrar mediante operadores lógicos: citas compuestas por dos autores, por lo que se utilizará el operador *AND*, de la forma **David AND Felipe** o de la forma **David AND Felipe AND Carlos**, incluyendo únicamente las citas escritas por todos los autores; citas compuestas por alguno de los autores que ha incluido mediante el uso del operador *OR*, siendo de la forma **David OR Felipe** o también **David OR Felipe OR Carlos**, mostrando una cita cuando pertenezca a uno de los autores; citas que no incluyan a un autor determinado mediante la utilización del operador *NOT*, de la forma **NOT David**, para que no se incluyan las citas en las que haya participado el autor **David.**

  También se pueden utilizar dichos operadores lógicos combinados entre sí y con la utilización de paréntesis "()" para priorizar operadores de la tal forma que un filtro válido fuese, **David AND (Felipe OR Carlos)** por lo que, primero se realizará el filtrado de aquellas citas que pertenezcan al autor **Felipe** o al autor **Carlos** y después, en una segunda iteración, de las citas resultantes del primer filtrado se incluirán en el reporte aquellas en las que también haya participado **David**. También es válido un filtro de la forma **David OR (Felipe AND Carlos)** para incluir aquellas entradas que cumplan la condición de que pertenezca al autor **David** o la condición de pertenecer a los autores **Felipe** y **Carlos**, o un filtro de la forma **NOT(David AND Felipe)** para excluir aquellas citas escritas por ambos autores y su variante **NOT(David OR Felipe)** de tal forma que la exclusión se dará siempre y cuando alguno de los autores sea partícipe de la cita. Este proceso se lleva a cabo mediante la función *compruebaAutor()*, de la cual se puede ver una extracción en la figura 32.

  Finalmente, cabe destacar que los operadores lógicos pueden escribirse en minúscula y los paréntesis pueden estar al principio de la operación lógica o al final, por lo que los filtros **David and Felipe**, **not(David OR Carlos)** o **(David and Carlos) or Felipe**, siguen siendo filtros válidos.

  Este filtro, únicamente soporta expresiones con la sintaxis indicada anteriormente y no cualquier fórmula booleana que emplee los operadores *AND*, *OR* y *NOT*. Para que se pudiese realizar el filtrado con cualquier fórmula booleana, habría sido necesario implementar un análisis sintáctico que generase un árbol sintáctico para la fórmula indicada. Los conocimientos para aplicar ese proceso no estaban a mi alcance debido a no haber cursado una asignatura de procesadores de lenguaje y, por ello, no se pudo implementar.

```
function compruebaAutor(autores, dato){
  if(dato && autores){

    if(dato.indexOf("(") > -1 && dato.indexOf(")") > -1){//Si existen paréntesis
      var regExp = /\(([^(][^)]+)\)/; //Coger lo que haya entre los paréntesis
      var matches = regExp.exec(dato);

      if(dato.indexOf("NOT ") > -1 || dato.indexOf("not ") > -1){ //NOT (a AND b)
        if(compruebaAutor(autores, matches[1])){
          return false;
        }
        else{
          return true;
        }
      }

      if(matches[1].indexOf(" OR ") > -1 || matches[1].indexOf(" or ") > -1){ //(b OR c)
        if(dato.indexOf(" AND ") > -1){
          var test = dato.split(" AND "); //a;(b OR c)
        }
        else if(dato.indexOf(" and ") > -1){
          var test = dato.split(" and "); //a;(b or c)
        }

        if(test[0].indexOf(" OR ") > -1 || test[0].indexOf(" or ") > -1){//(b OR c);a
           if(compruebaAutor(autores, test[1]) && compruebaAutor(autores, matches[1])){
             return true;
           }
           else{
             return false;
           }
        }
        else{
          if(compruebaAutor(autores, test[0]) && compruebaAutor(autores, matches[1])){
             return true;
           }
           else{
             return false;
           }
        }
```

Figura 32: Extracción del filtro de autores

- **Series** representa el nombre de una colección de libros y se pueden especificar varias cadenas, mediante el uso de las comas. Al ser un campo que no está presente en todos los tipos de entradas, se decidió excluir aquellas entradas que no contengan un campo **series**. La función implementada para realizar este filtrado es *compruebaSeries()* y se muestra en la figura 33.

```
function compruebaSeries(serieBib, dato){ //Aquellos que no tengan series, no se muestran
  if(dato){
    if(dato.indexOf(",") > -1){
      var series = dato.split(",");
      for(i=0; i<series.length; i++){
        series[i] = series[i].trim();
        if(serieBib == series[i]){
          return true;
        }
      }
      return false;
    }
    else{
      dato = dato.trim();
      if(serieBib == dato){
        return true;
      }
      return false;
    }
  }
  return true;
}
```

Figura 33: Filtro de series

- **Editorial** que expresa el nombre de la editorial encargada de la publicación, indicada en el campo *publisher* de una entrada. Al igual que en el caso anterior, al no ser un campo presente en todos los tipos de entradas, aquellas entradas que no contengan la clave *publisher* son excluidas del reporte. La función *compruebaEditorial()* es la que lleva a cabo el proceso de filtrado en esta ocasión y se representa en la figura 34.

```
function compruebaEditorial(editorialBib, dato){ //Aquellos que no tengan editorial, no se muestran
  if(dato){
    if(dato.indexOf(",") > -1){
      var editoriales = dato.split(",");
      for(i=0; i<editoriales.length; i++){
        editoriales[i] = editoriales[i].trim();
        if(editorialBib == editoriales[i]){
          return true;
        }
      }
      return false;
    }
    else{
      dato = dato.trim();
      if(editorialBib == dato){
        return true;
      }
      return false;
    }
  }
  return true;
}
```

Figura 34: Filtro de editoriales

Para terminar, al lado de cada uno de los campos de filtrado, se añadió un icono con un símbolo de información ℹ el cual sirve para mostrar un mensaje de ayuda al usuario en caso de querer saber cómo funcionan los filtros. Cada filtro tiene asociado un mensaje de ayuda para que no se proporcione información innecesaria al usuario y se muestra únicamente cuando se pasa el puntero por encima del icono, como se aprecia en la figura 35.

Figura 35: Información del filtro de tipo de entrada

3.4. Hito 3

Una vez que se alcanzaron los objetivos establecidos en el primer y segundo hito, este último se centró en implementar una propiedad de adición, es decir, facilitar al usuario una manera de añadir nuevas entradas a su fichero .bib sin tener que descargarlo del servicio Drive a su ordenador, modificarlo, subir el nuevo fichero .bib a Drive de nuevo y borrar el fichero antiguo de Drive para evitar confusiones al seleccionarlo en el complemento.

Para ello se implementó un nuevo botón (ver figura 36) en la barra lateral con el texto *Añadir entrada* el cual hace una petición al servidor para generar una nueva ventana que contendrá los datos especificados en el fichero *Info.html*. El botón se observa en la interfaz en la figura 37.

```
<div class="block" id="button-bar">
  <input type="reset" id="reset" value="Resetear formulario">
  <input type="reset" id="info" value="Añadir entrada" onclick="showAlert()">
</div>
```

Figura 36: Código del botón para añadir nuevas entradas

Figura 37: Botón para añadir entrada en la barra lateral

En el lado del servidor, se utiliza el servicio HTML para hacer una llamada a la creación de una nueva ventana modificando la interfaz existente. En esa parte, se especifica el fichero del cual se obtienen los datos de la ventana, así como el nombre que tendrá. Mediante la función *showInfo()*, mostrada en la figura 38, se hace uso de dicho servicio HTML.

```
function showInfo() {

  var html = HtmlService.createHtmlOutputFromFile('Info')
  .setWidth(600)
  .setHeight(425)


  DocumentApp.getUi().showModalDialog(html, 'Añadir entrada');


}
```

Figura 38: Llamada a *HTMLService*, utilizando un fichero .html

Una vez que el servidor hace la petición de generar un nuevo elemento en el interfaz, se recogen los datos de *Info.html* y se muestra la nueva ventana con los elementos propios del fichero, como se aprecia en la figura 39.

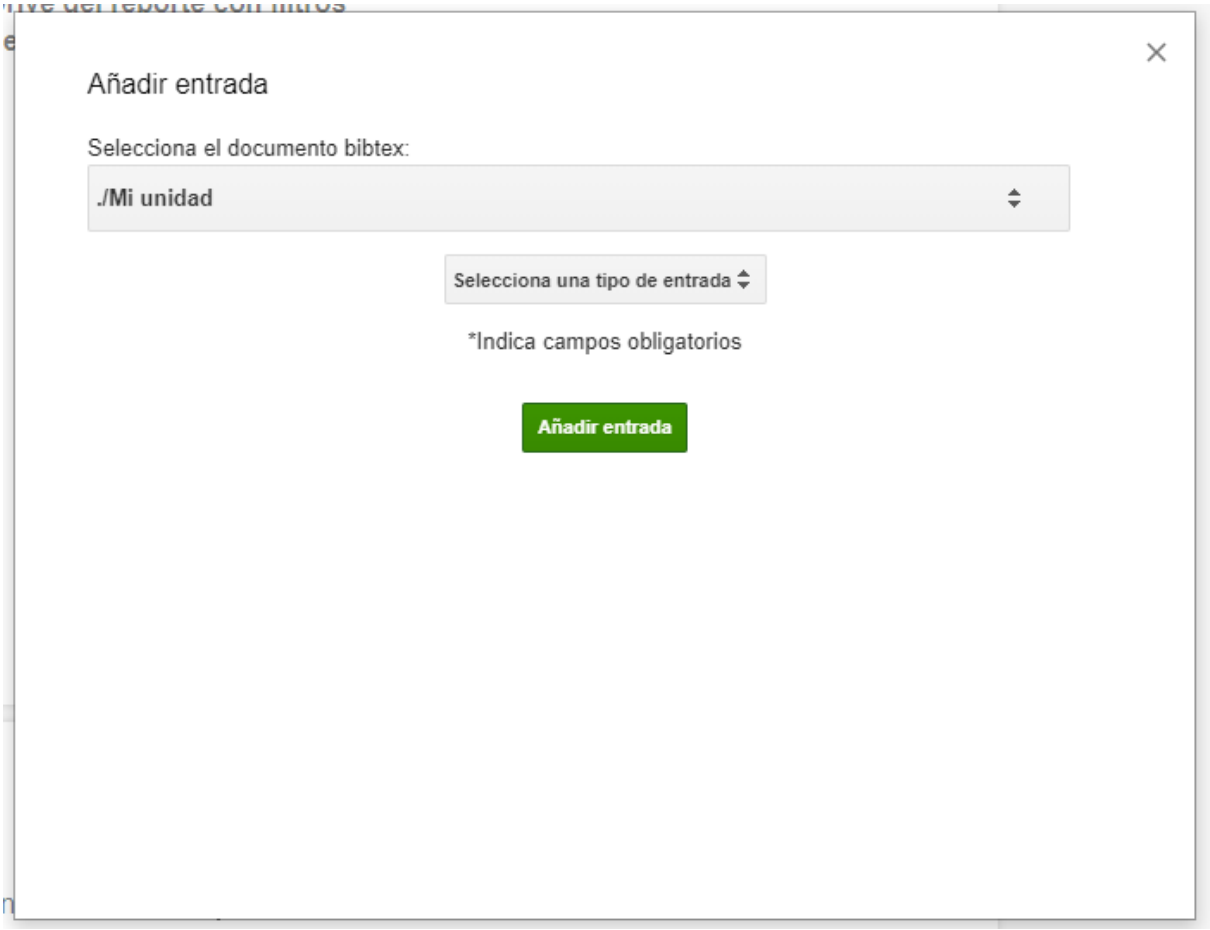


Figura 39: Ventana para añadir entradas

Esta nueva ventana consta de un desplegable en el cual se indicará el fichero .bib en el que se desea añadir la nueva entrada, un desplegable con el cual se debe elegir el tipo de la nueva entrada para que, a continuación, aparezcan los campos asociados a dicho tipo de entrada. Dichos campos serán tanto los obligatorios para ese tipo de entrada y que se indican con un asterisco * antes del nombre del campo, como los opcionales que únicamente contienen el nombre del campo. Además, hay un texto que indica que la presencia de un asterisco en un campo denota obligatoriedad.

Todo ello se observa en la figura 40, para la cual se ha indicado que se quiere añadir una entrada de tipo *@techreport.*

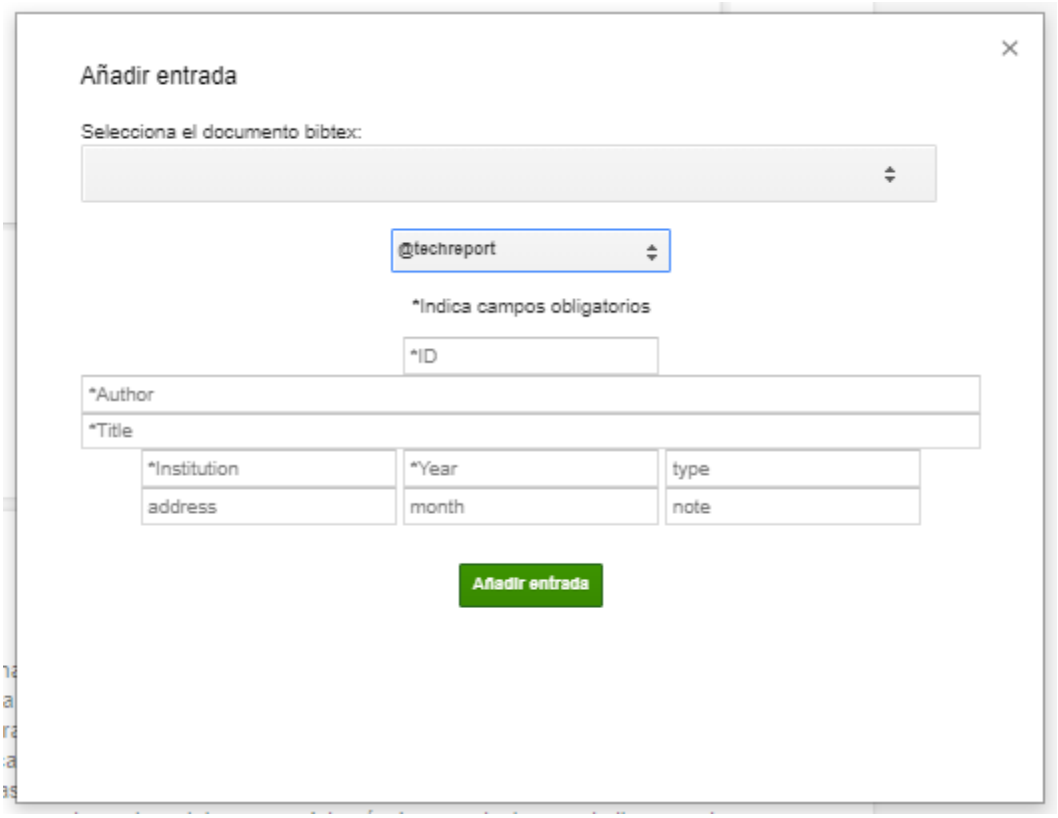


Figura 40: Campos obligatorios y opcionales para una entrada de tipo *techreport*

Una vez que los campos se han rellenado, al pulsar el botón verde con el texto *Añadir entrada*, se realizará una llamada a la función *prueba()* en la que se capturarán los datos introducidos en los campos asignados, la opción del desplegable que se ha seleccionado y el ID del fichero .bib que se ha elegido para añadir la entrada.

Es entonces cuando, habiendo capturado los datos, se realizará una nueva petición al servidor mediante la API de JavaScript *google.script.run* (ver figura 41) con los métodos *withSuccessHandler()* para especificar la actuación al haberse completado la llamada al lado del servidor y habiendo recibido un parámetro de dicho lado, el método *withFailureHandler()* que sirve para especificar un mensaje de error en caso de que la llamada al lado del servidor no haya terminado correctamente y el método *withUserObject()* para definir un segundo objeto utilizado como parámetro para los métodos anteriores.

```
option = option.substr(1);
 google.script.run
 .withSuccessHandler(
   function(exito, element){
     if(exito == 99){
       var msg = "Ya existe una entrada con ese ID. No se ha modificado el fichero";
       showErrorE(msg, $('#infoBlock'));
     }else if(exito){
       var msg = "Comprueba el documento seleccionado en tu Drive";
       showSuccessE(msg, $('#infoBlock'));
     }else{
       var msg = "Falta algún campo obligatorio. No se ha modificado el fichero";
       showErrorE(msg, $('#infoBlock'));
     }
     element.disabled = false;
   })
 .withFailureHandler(
   function(msg, element) {
     msg = "Hubo un problema durante el proceso de inclusión.";
     showErrorE(msg, $('#infoBlock'));
     element.disabled = false;
   }
 )
 .withUserObject(this)
 .pruebaEscritura(idDoc,campos,option);
}
```

Figura 41: Llamada al lado del servidor para la escritura de la entrada

La llamada a la función del lado del servidor *pruebaEscritura()* está representada en la figura 42 y recibe los parámetros:

- **idDoc** que representa el ID del documento en el que se va a añadir la nueva entrada.
- **campos**, un *array* con los datos que ha introducido el usuario para cada atributo de la entrada.
- **option** que contiene el tipo de entrada que ha seleccionado el usuario y con el que se realizan comprobaciones.

```
function pruebaEscritura(idDoc, campos, option){

   var docBib = DriveApp.getFileById(idDoc);


   var existe = false;
   var exito = true;

   var error = false;
   var numberError = 99; //Error para ID repetido

   var objetivo = new BibTex();
   var objetivoComp = new BibTex();
   objetivo.content = docBib.getBlob().getDataAsString();
   objetivoComp.content = docBib.getBlob().getDataAsString();

   objetivoComp.parse();
   for(i=0; i<objetivoComp.data.length; i++){
     if(objetivoComp.data[i].cite == campos[0]){
       existe = true;
     }
   }

   if(existe == true){

     return numberError;
   }

     var fallo = false;
     switch(option){
       case "book":

         for(i=0; i<5; i++){
           if(campos[i] == "" || campos[i] === undefined){
             fallo = true;
           }
         }
```

Figura 42: Extracción de la función *pruebaEscritura()*

En la función *pruebaEscritura()* se creará un objeto de tipo BibTex() para comprobar si el primer dato introducido por el usuario, es decir, el ID de la nueva entrada, ya existe o no. En caso de que el ID se encuentre en el fichero .bib a modificar, se dará por finalizada la llamada al servidor y se devolverá el número 99, asociado al error por ID ya existente. La comprobación del código devuelto se realiza en el método *.withSuccessHandler()* que se muestra en la figura 43 y, en caso de que haya error, se mostrará en la ventana un mensaje de error como el de la figura 44.

```
.withSuccessHandler(
  function(exito, element){
    if(exito == 99){
      var msg = "Ya existe una entrada con ese ID. No se ha modificado el fichero";
      showErrorE(msg, $('#infoBlock'));
    }else if(exito){
      var msg = "Comprueba el documento seleccionado en tu Drive";
      showSuccessE(msg, $('#infoBlock'));
    }else{
      var msg = "Falta algún campo obligatorio. No se ha modificado el fichero";
      showErrorE(msg, $('#infoBlock'));
    }
    element.disabled = false;
  })
```

Figura 43: Captura de los errores ID repetido y campo obligatorio vacío

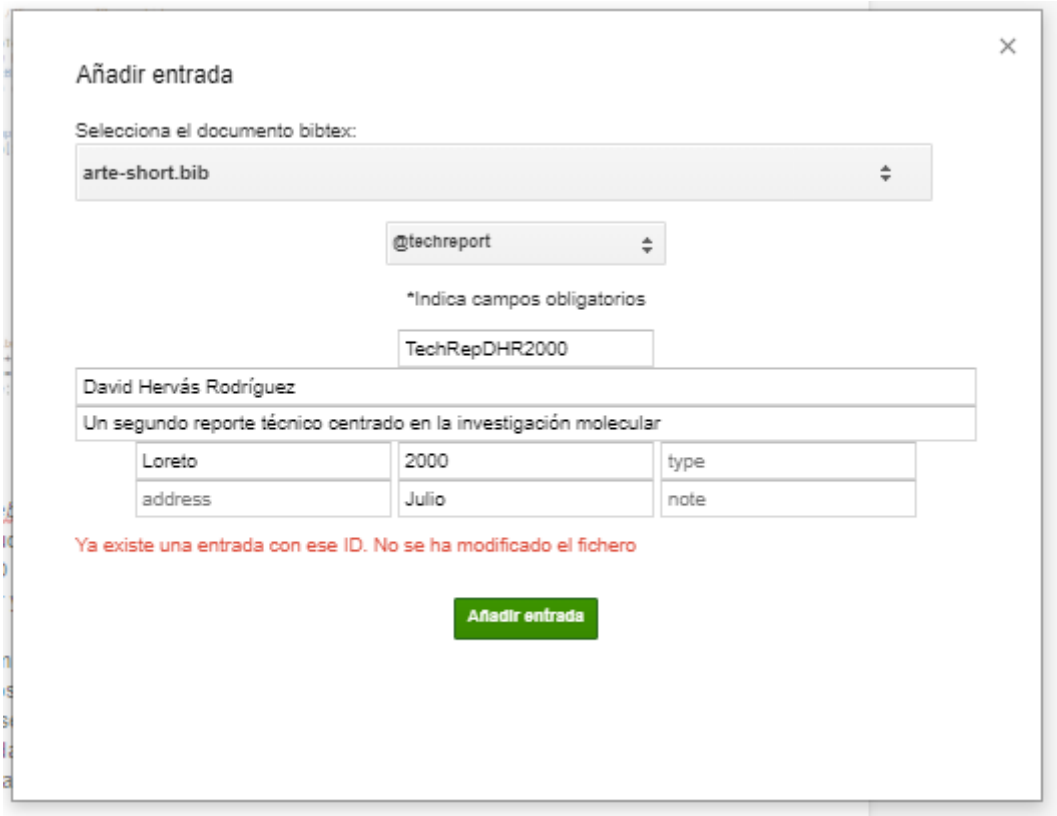


Figura 44: Mensaje de ID repetido al añadir una entrada

Si el ID no está repetido, se comprobará que los campos obligatorios para el tipo de entrada, representado por el parámetro *option*, existan y tengan un valor definido que no sea vacío. En el caso de que algún campo obligatorio no tenga un valor, la función devolverá el parámetro *exito* con el valor *false*, que está asociado al error por campo obligatorio vacío de forma que se mostrará un mensaje de error advirtiendo del problema. Además durante este proceso, se da formato a la cadena *datFin* que representa la nueva entrada y que se incluirá en el fichero .bib correspondiente.

En caso de que al realizar la comprobación se obtenga el valor *false* en el parámetro éxito, un mensaje de error se mostrará como en el caso de tener el ID repetido, descrito anteriormente, pero especificando que el problema es algún campo obligatorio (ver figura 45).

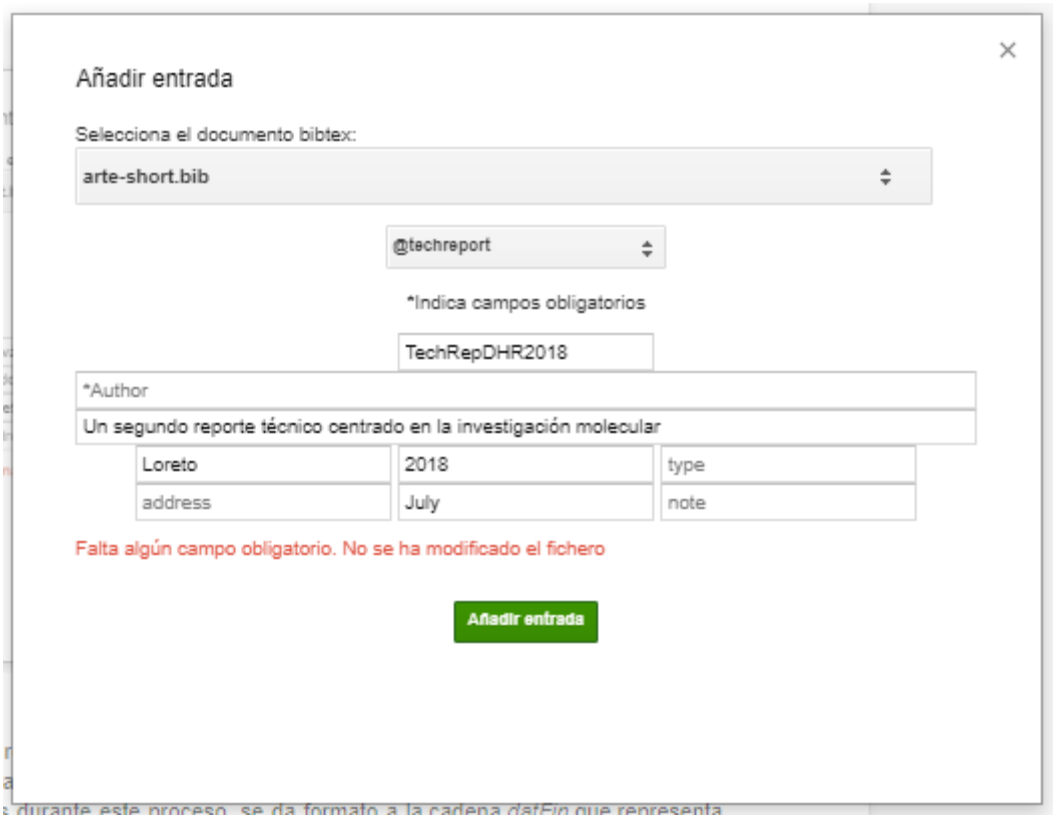


Figura 45: Mensaje de campo obligatorio vacío al añadir una entrada

Dicha cadena, estará compuesta en primer lugar por el contenido ya existente del fichero .bib para después añadirle la nueva cadena representando la entrada que se quiere insertar. Esto es necesario, debido a que para modificar el fichero .bib, se utiliza el método *setContent()* de la API de Drive incluida en Apps Script, el cual define el contenido del fichero en cuestión, reemplazando su contenido previo. Es por ello que en el proceso se inserta el contenido previo del fichero .bib y después la nueva cadena. En la figura 46 se aprecia que en la variable *datFin* se guarda el contenido actual del fichero representado como *objetivo.content* al que se le añadirá los campos en orden de la nueva entrada para, más adelante, reemplazar el contenido del fichero original mediante el método *.setContent()*.

```
case "techreport":
  //comprobar atributos obligatorios
  for(i=0; i<5; i++){
    if(campos[i] == "" || campos[i] === undefined){
      fallo = true;
    }
  }

  if(fallo == true){
    exito = false;
  }
  else{
    var datFin = objetivo.content + '\n\n' + '@' + option + '{' + campos[0] + ',\n   author={' + campos[1] + '},\n   title={'
    + campos[2] + '},\n   institution={' + campos[3] + '},\n   year={' + campos[4] + '}';
    //Campos opcionales
    if(campos[5]){
      datFin = datFin + ',\n   type={' + campos[5] + '}';
    }
    if(campos[6]){
      datFin = datFin + ',\n   address={' + campos[6] + '}';
    }
    if(campos[7]){
      datFin = datFin + ',\n   month={' + campos[7] + '}';
    }
    if(campos[8]){
      datFin = datFin + ',\n   note={' + campos[8] + '}';
    }
    datFin = datFin + '\n}';

    docBib.setContent(datFin);
  }

break;
```

Figura 46: Definición de una cadena de tipo *techreport* en la función *pruebaEscritura()*

Una vez que se ha llamado a la función *setContent()* y se ha modificado el fichero .bib, se devuelve el parámetro *exito* con el valor *true* que, a su vez, es recogido por el controlador en el lado del cliente y mostrará el mensaje de éxito que se aprecia en la figura 47 y el fichero .bib seleccionado habrá sido modificado como se observa en la figura 48.

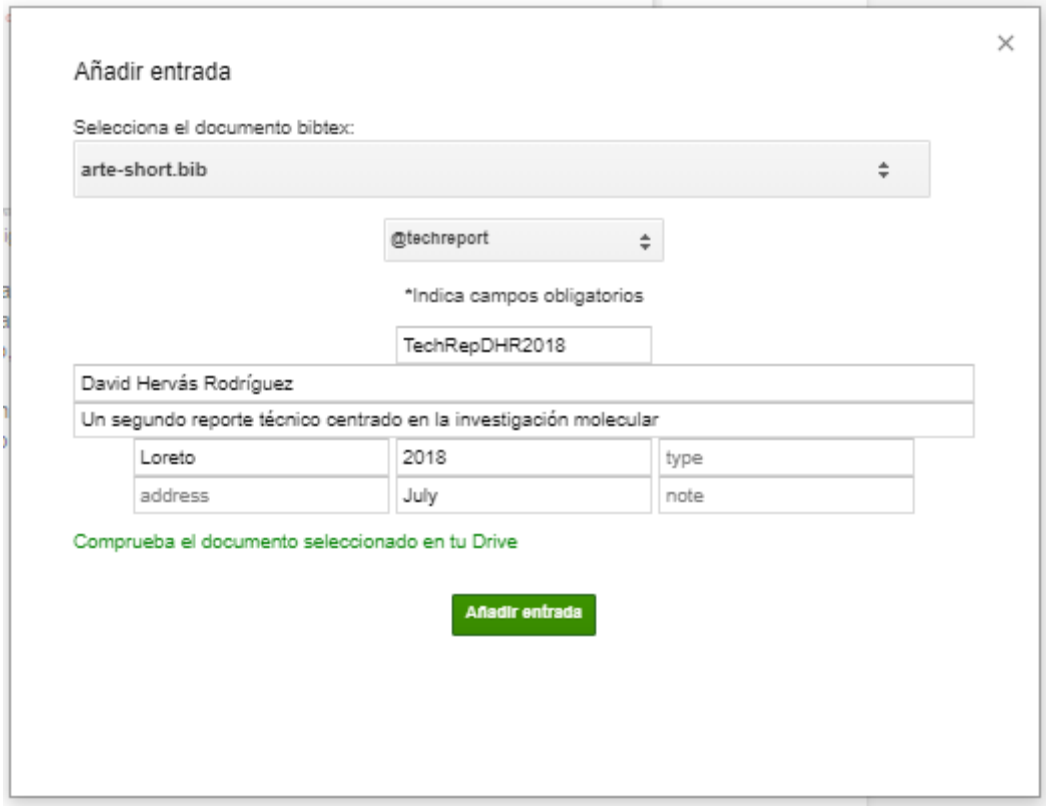
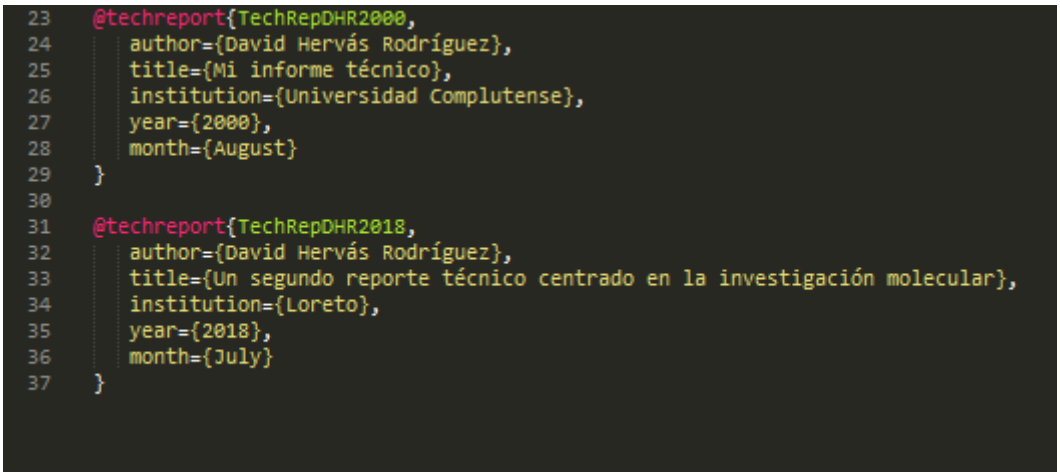


Figura 47: Mensaje de éxito al añadir una nueva entrada



```
@techreport{TechRepDHR2000,
    author={David Hervás Rodríguez},
    title={Mi informe técnico},
    institution={Universidad Complutense},
    year={2000},
    month={August}
}

@techreport{TechRepDHR2018,
    author={David Hervás Rodríguez},
    title={Un segundo reporte técnico centrado en la investigación molecular},
    institution={Loreto},
    year={2018},
    month={July}
}
```

Figura 48: Fichero .bib con la nueva entrada añadida al final

4. EJEMPLOS DE USO

En esta sección se muestra un ejemplo del funcionamiento de las modificaciones implementadas en el complemento. Se indicará el proceso a realizar para:

- Instalar el complemento
- Generar un reporte bibliográfico mediante varios ficheros .bib
- Aplicar filtros sobre el mismo reporte
- Añadir una entrada a un fichero .bib

Primero, hay que acceder a nuestro servicio de Google Drive. Para ello, al abrir una pestaña en Google Chrome, pinchamos sobre el símbolo ⋮⋮⋮ ubicado en la esquina superior derecha y después sobre el icono de Drive.

Si no tenemos iniciada nuestra sesión en Chrome, se nos solicitará iniciarla introduciendo nuestro correo electrónico y nuestra contraseña.

Una vez que hemos accedido a nuestro espacio en Drive, debemos crear un nuevo documento en blanco de Google. Esto se hace seleccionando en la esquina superior izquierda el botón Nuevo . A continuación, hay que seleccionar en el desplegable *Documentos de Google* y después *Documento en blanco*.

Una vez en el documento que hemos creado, pinchamos en el apartado *Herramientas* y seleccionamos la opción *Editor de secuencias de comandos* y se nos abrirá una ventana de desarrollo como la de la figura 49.

Figura 49: Nueva ventana de desarrollo

Una vez en la ventana de desarrollo, es conveniente que hagamos *click* en el texto *Proyecto sin título* e indiquemos un nombre con el que guardar nuestro proyecto (ver figura 50). De esta manera, más adelante nos saltaremos este paso.

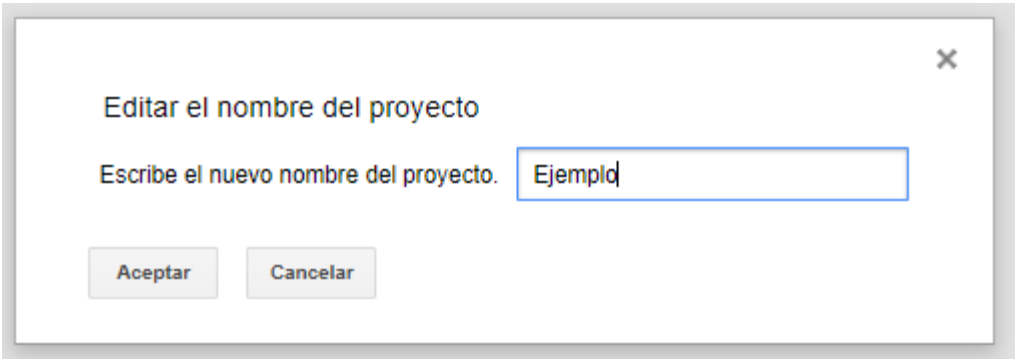


Figura 50: Nombre del proyecto

Ahora que hemos guardado nuestro proyecto de Google Apps Script, deberemos acceder a la dirección de GitHub[21] en la que se encuentran los archivos de código. Deberemos copiar el contenido del archivo *Codigo.gs* de GitHub en el archivo que tenemos abierto del editor de secuencias de comandos y que, por defecto, tiene el nombre *Código.gs* de tal forma que el resultado sea como el que se ve en la figura 51.

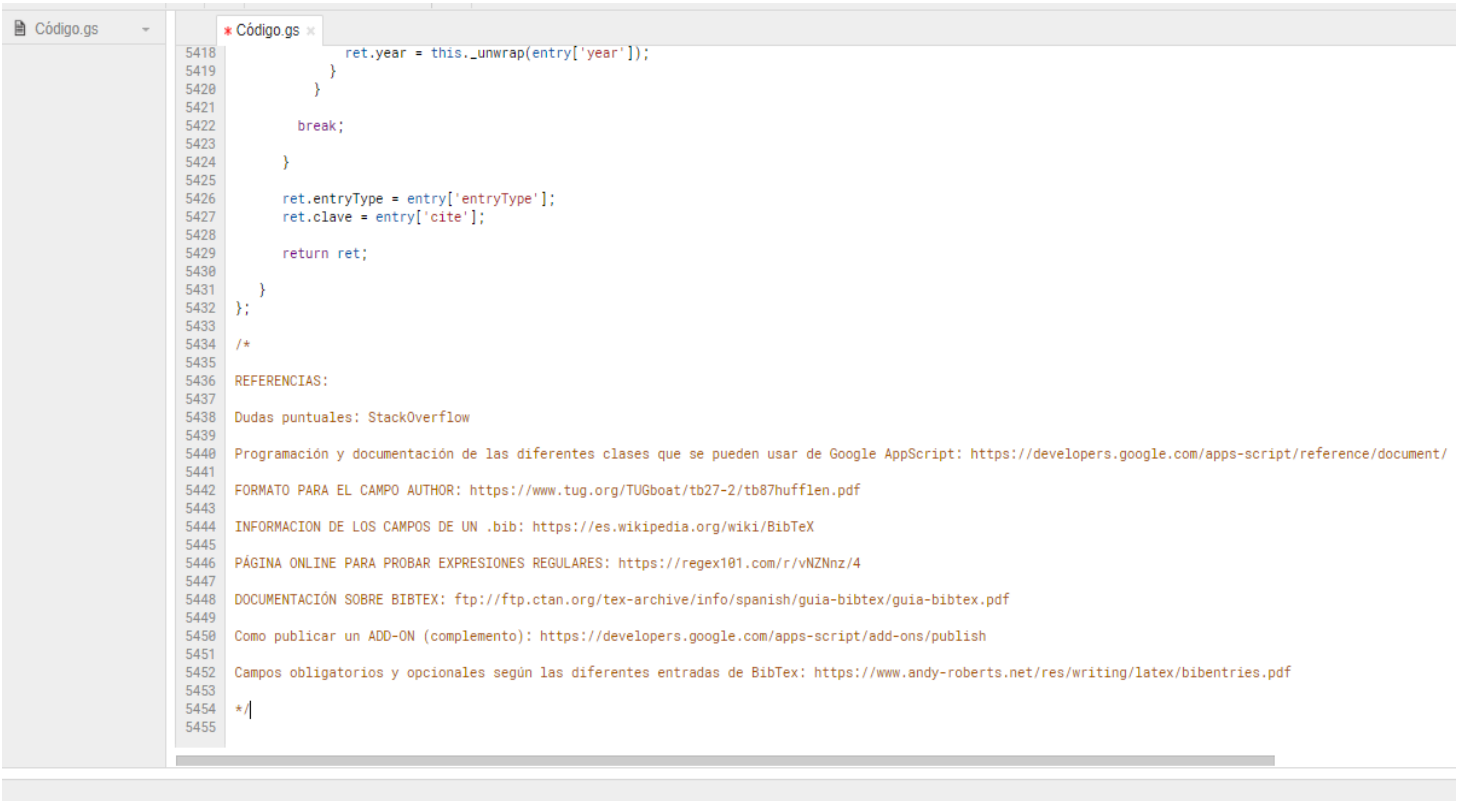


Figura 51: Añadido el fichero *Codigo.gs* de GitHub al proyecto

Una vez que hemos tenemos el fichero *Código.gs* con el contenido del fichero de GitHub, debemos seleccionar, en la misma ventana de desarrollo, *Archivo* y después en la subpestaña *Nuevo*, seleccionamos *Archivo HTML* y lo guardaremos con el nombre *Sidebar*. Debemos crear dos archivos HTML, por lo que repetiremos el proceso anterior, esta vez guardando el archivo HTML con el nombre *Info*, de tal forma que deberemos tener un menú lateral similar al de la figura 52.

[21] *https://github.com/dahervas/TFG*

Figura 52: Ventana de desarrollo con los ficheros HTML creados

Nuevamente, deberemos copiar el contenido de los ficheros del repositorio de GitHub *Sidebar.html* e *Info.html* en sus respectivos ficheros del proyecto de Google Apps Script. Una vez completado este proceso, volveremos al documento en blanco que hemos generado al principio y pulsamos F5 o el símbolo para cargar la página de nuevo

De esta manera, la pestaña del editor de secuencias de comandos se fusionará con el documento en blanco y se habrá terminado la instalación del complemento. Ahora en la pestaña *Complementos* aparecerá una opción con el nombre que le dimos al proyecto, en este caso el nombre *Ejemplo*. Al pasar el puntero por encima del nombre de nuestro proyecto se nos dará la opción *Comenzar* para iniciar el complemento como se muestra en la figura 53.

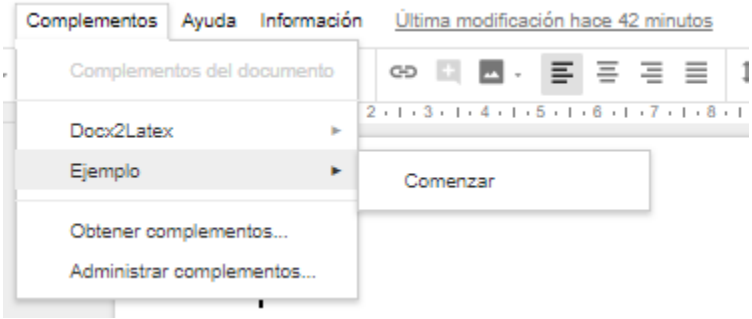


Figura 53: Complemento instalado en el documento

Al iniciarlo por primera vez se nos pedirá que concedamos permisos al complemento mediante un mensaje como el mostrado en la figura 54. Esto se debe a que el complemento leerá los ficheros de Drive para mostrarlos en el desplegable como también necesitará permisos para modificarlos.

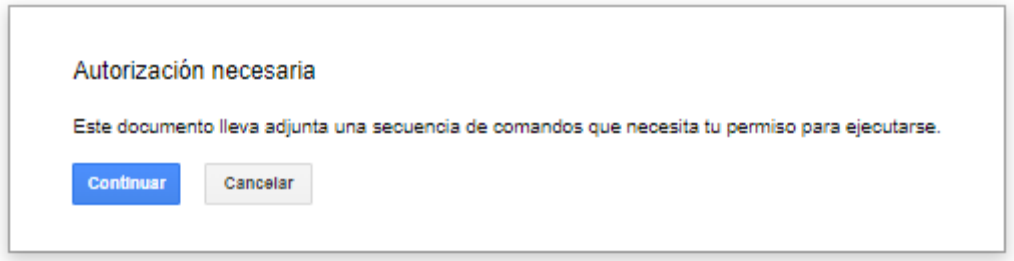


Figura 54: Mensaje de autorización al ejecutar por primera vez el complemento

Cuando se hayan concedido los permisos correspondientes, se mostrará la barra lateral en el documento abierto. A continuación, para generar un reporte tendremos que asegurarnos de tener en nuestro Drive al menos un fichero de extensión .bib el cual será el que utilizaremos para generar dicho reporte. De todas formas, en el repositorio de GitHub del cual se han extraído los ficheros *Codigo.gs*, *Sidebar.html* e *Info.html* hay tres ficheros de extensión .bib, *arte.bib*, *tested.bib* y *tested2.bib* los cuales pueden ser descargados y después añadidos al Drive para tener ficheros con los que comprobar el buen funcionamiento del complemento.

Llegados a este punto, podemos hacer *click* en la esquina superior izquierda del fichero, en el texto *Documento sin título* para asignar un título al documento, aunque no supondría un problema no darle un nombre.

Es entonces cuando podemos proceder a añadir en el documento abierto la cadena *\report* la cual es necesaria para incluir el reporte generado en ese lugar. Cuando hayamos incluido dicha cadena y hayamos seleccionado los ficheros que se quiera en el complemento, pulsaremos el botón *Crear un reporte* (ver figura 55).

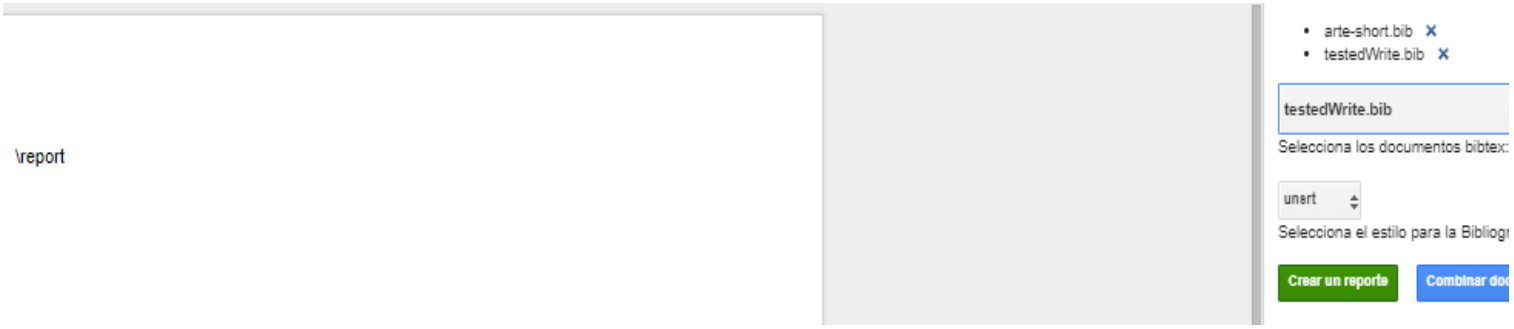


Figura 55: Cadena *\report* y ficheros .bib para la generación del reporte.

Como se aprecia en la figura 56, se mostrará un mensaje de color verde (asociado al buen funcionamiento) que nos indicará que el proceso ha sido un éxito y que se ha generado en nuestro Drive un reporte cuyo nombre será el nombre del documento en blanco que habíamos generado (si no le hemos puesto un nombre será *Documento sin título*) más la cadena *-Report*.

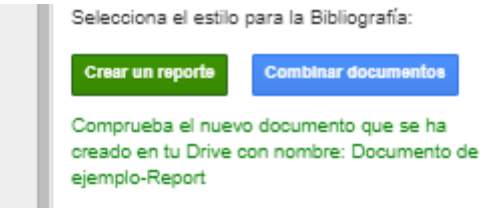


Figura 56: Mensaje de éxito en la generación del fichero

Por supuesto, en nuestro Drive podremos acceder al documento generado con el nombre indicado en el mensaje de éxito anterior mostrado en el apartado *Acceso rápido* (ver figura 57) y con el contenido mostrado en la figura 58.

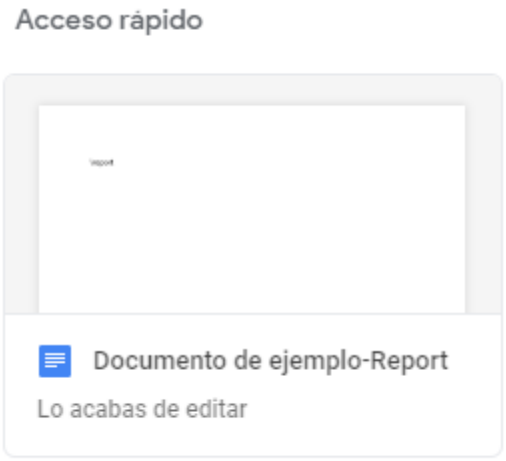
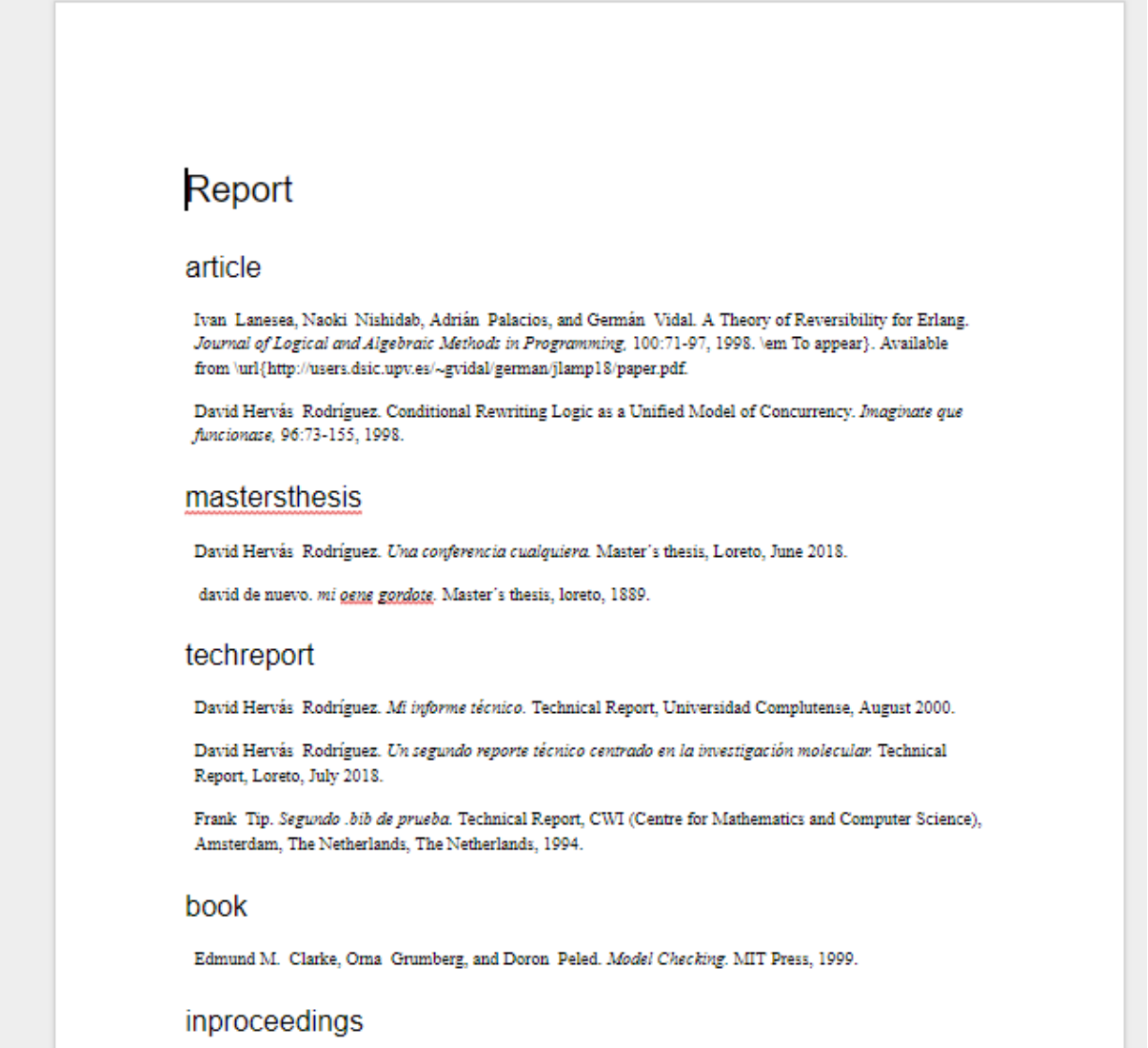


Figura 57: Vista predeterminada de Drive del reporte



Report

article

Ivan Lanesea, Naoki Nishidab, Adrián Palacios, and Germán Vidal. A Theory of Reversibility for Erlang. *Journal of Logical and Algebraic Methods in Programming*, 100:71-97, 1998. \em To appear}. Available from \url{http://users.dsic.upv.es/~gvidal/german/jlamp18/paper.pdf.

David Hervás Rodríguez. Conditional Rewriting Logic as a Unified Model of Concurrency. *Imaginate que funcionase*, 96:73-155, 1998.

mastersthesis

David Hervás Rodríguez. *Una conferencia cualquiera*. Master's thesis, Loreto, June 2018.

david de nuevo. *mi oeng gordote*. Master's thesis, loreto, 1889.

techreport

David Hervás Rodríguez. *Mi informe técnico*. Technical Report, Universidad Complutense, August 2000.

David Hervás Rodríguez. *Un segundo reporte técnico centrado en la investigación molecular*. Technical Report, Loreto, July 2018.

Frank Tip. *Segundo .bib de prueba*. Technical Report, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands, 1994.

book

Edmund M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

inproceedings

Figura 58: Reporte generado en Drive

En el caso de obtener un mensaje de error como el de la figura 59, indicando que hubo un problema con una entrada junto con el ID de dicha entrada, deberemos descargar los ficheros .bib implicados en la generación del reporte, y buscar en ellos el ID proporcionado (ver figura 60).

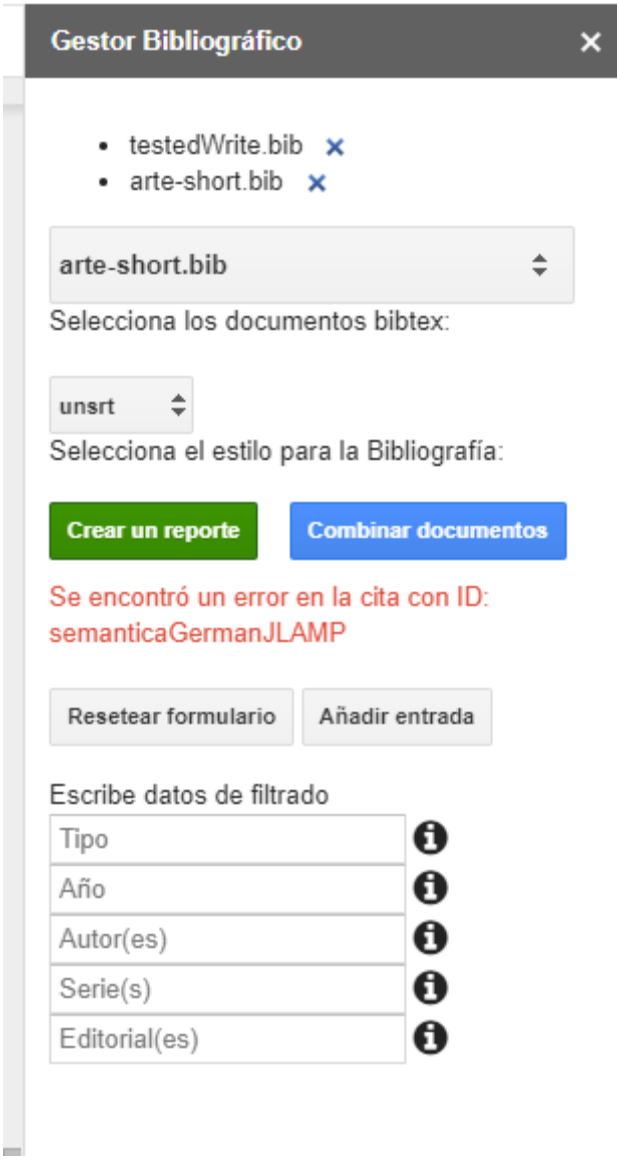


Figura 59: Mensaje de error en una entrada

```
@article{semanticaGermanJLAMP,
  title = "A Theory of Reversibility for {E}rlang",
  author = {Lanesea, Ivan and Nishidab, Naoki and Palacios, Adri\'an and Vidal, Germ\'an},
  journal = "Journal of Logical and Algebraic Methods in Programming",
  volume = 100,
  pages = "71-97",
  note = "{\em To appear}. Available from \url{http://users.dsic.upv.es/~gvidal/german/jlamp18/paper.pdf}",
}
```

Figura 60: Entrada errónea

Revisando los campos obligatorios y opcionales de cada entrada y que podemos consultar en [7], vemos que las entradas de tipo *article* deben incluir obligatoriamente los campos *author*, *title*, *journal* y *year*. Por ello, observamos que no se encuentra el campo *year* en la entrada, lo cual ha provocado el fallo. Corregimos el error añadiendo el campo perdido como se muestra en la figura 61 y subimos de nuevo el fichero .bib a Drive.

```
@article{semanticaGermanJLAMP,
  title = "A Theory of Reversibility for {E}rlang",
  author = {Lanesea, Ivan and Nishidab, Naoki and Palacios, Adri\'an and Vidal, Germ\'an},
  journal = "Journal of Logical and Algebraic Methods in Programming",
  volume = 100,
  pages = "71-97",
  note = "{\em To appear}. Available from \url{http://users.dsic.upv.es/~gvidal/german/jlamp18/paper.pdf}",
  year = {1998}
}
```

Figura 61: Entrada corregida con el campo obligatorio que faltaba

Una vez que hemos generado el reporte con éxito, procederemos a aplicar filtros a dicho reporte. Probaremos realizando un filtrado por tipos de entrada y rango de años, de tal forma que en los campos correspondientes indicaremos que sólo queremos que en el reporte se incluyan las citas cuyo tipo de entrada sea, por ejemplo, *inproceedings*, cuyos años comprendan el rango de *1997* a *1999* como se indica en la figura 62.

Figura 62: Campos para filtrar el reporte por tipo de entrada y años

Igual que en el paso anterior, obtendremos un mensaje de éxito que nos informará de la correcta generación del reporte con filtros tal y como se muestra en la figura 63, obteniendo un documento cuyo contenido será el mostrado en la figura 64.

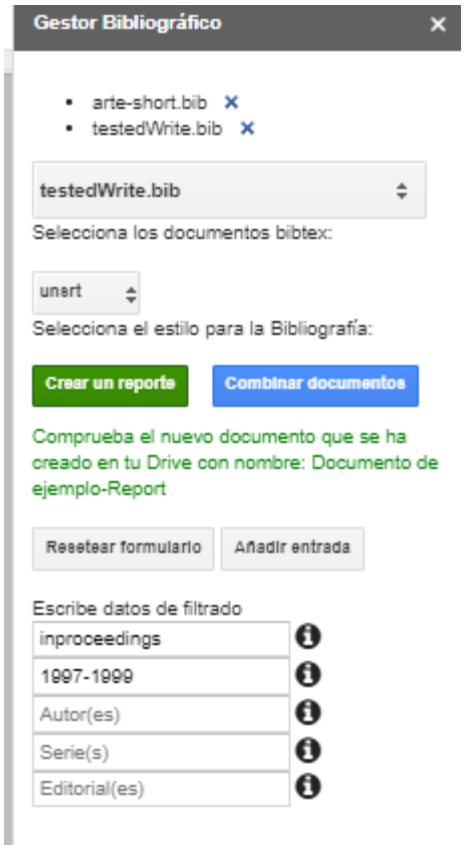
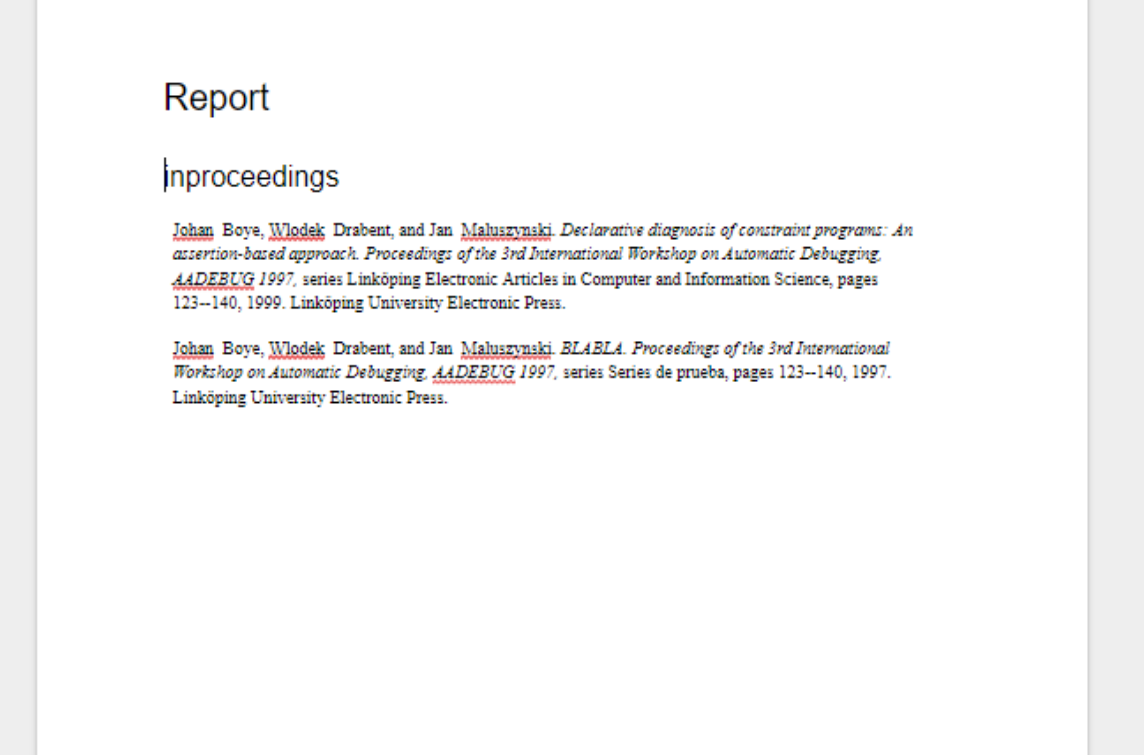


Figura 63: Mensaje de éxito al generar un reporte filtrado



Report

inproceedings

Johan Boye, Wlodek Drabent, and Jan Maluszynski. *Declarative diagnosis of constraint programs: An assertion-based approach. Proceedings of the 3rd International Workshop on Automatic Debugging, AADEBUG 1997*, series Linköping Electronic Articles in Computer and Information Science, pages 123--140, 1999. Linköping University Electronic Press.

Johan Boye, Wlodek Drabent, and Jan Maluszynski. *BLABLA. Proceedings of the 3rd International Workshop on Automatic Debugging. AADEBUG 1997*, series Series de prueba, pages 123--140, 1997. Linköping University Electronic Press.

Figura 64: Reporte generado con el filtrado

Si ahora queremos un reporte que contenga las entradas de tipo *inproceedings*, pero de los autores *Narciso* y *Alberto* habría que indicarlo en los filtros de la manera que se muestra en la figura 65.

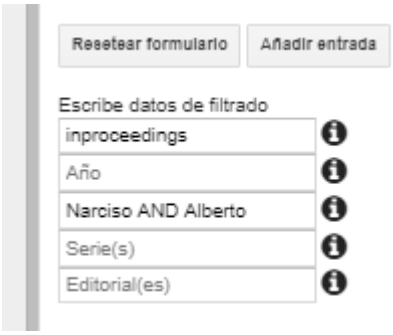


Figura 65: Campos para filtrar por tipo de entrada y autores

Como se ve en la figura 66 un mensaje de éxito nos confirmará la correcta generación del reporte con los filtros indicados en nuestro Drive el cual presentará un contenido como el de la figura 67.

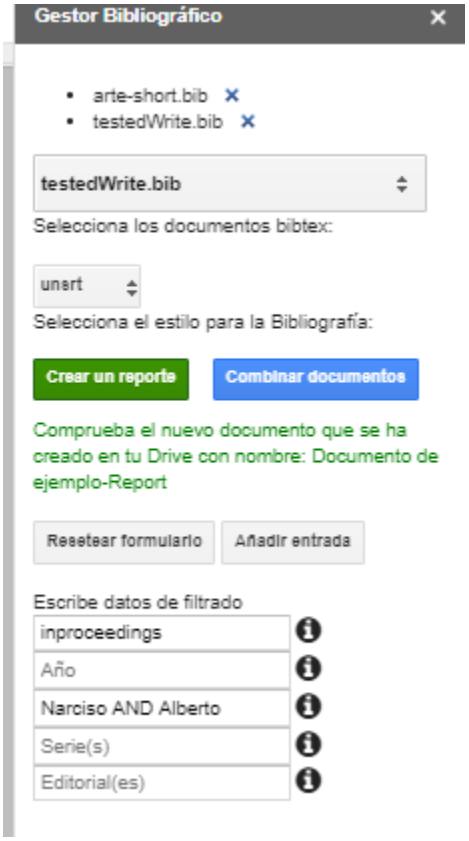


Figura 66: Mensaje de éxito al filtrar por tipo de entrada y autores

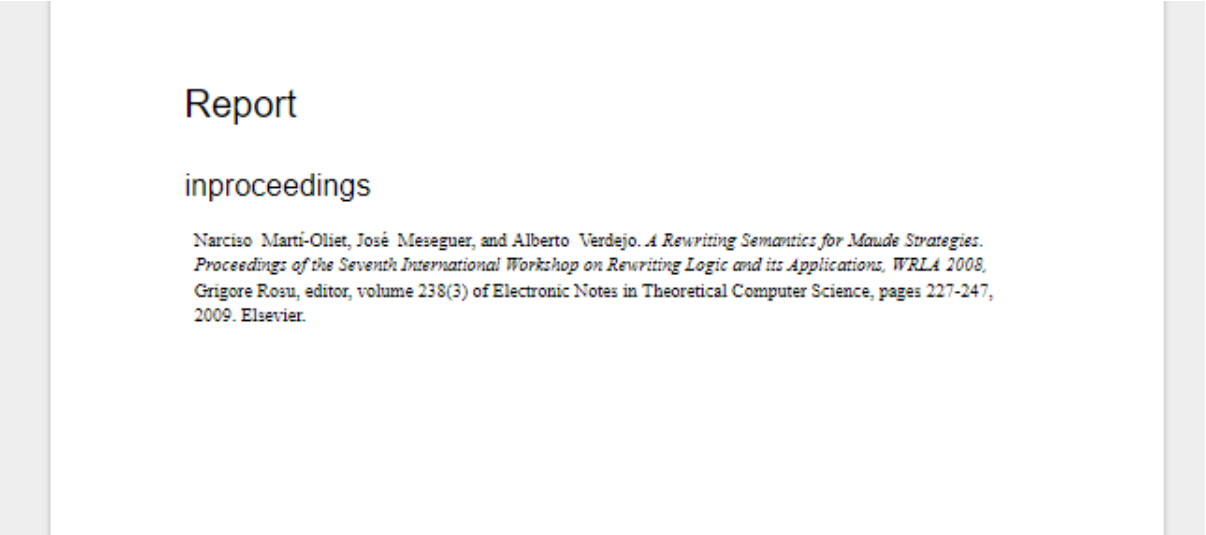


Figura 67: Reporte filtrado por tipo de entrada y autores

Además si no se entiende bien el funcionamiento de un filtro, siempre lo podremos consultar pasando el puntero encima del símbolo ℹ como se muestra en la figura 68.

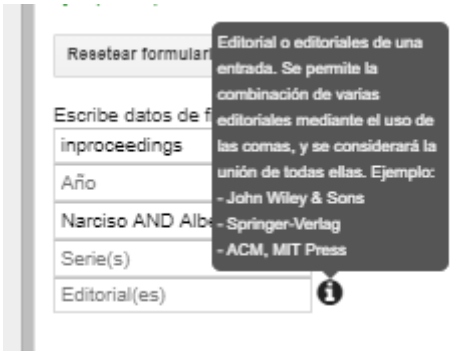


Figura 68: Mensaje de ayuda al lado de los filtros

Finalmente, si deseamos añadir una entrada nueva a un fichero .bib, deberemos hacer *click* sobre el botón blanco con el texto *Añadir entrada* mostrado en la figura 69, el cual nos mostrará una ventana como la representada en la figura 70.

Figura 69: Botón *Añadir entrada* a la derecha del botón *Resetear formulario*

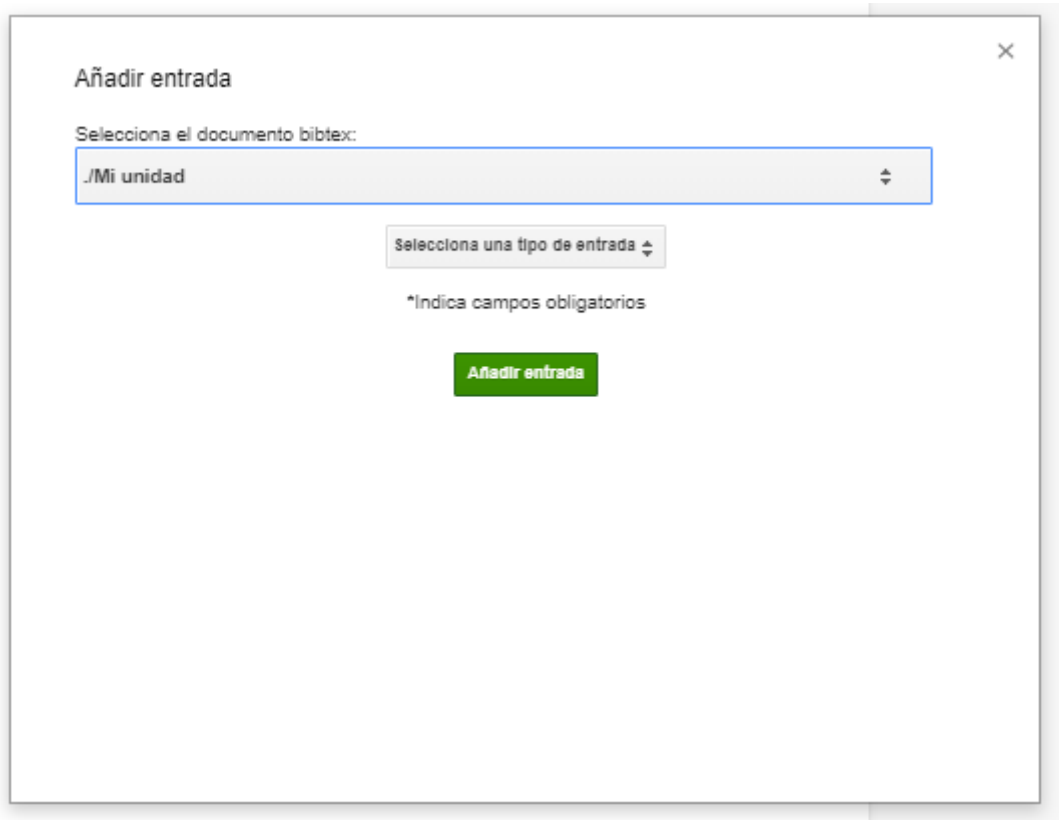


Figura 70: Ejemplo de ventana para añadir entradas

Entonces podremos seleccionar un documento .bib de nuestro Drive, con un contenido similar al que se muestra en la figura 71, en el desplegable así como el tipo de entrada que queremos añadir. Una vez seleccionado el tipo de entrada se mostrarán los campos correspondientes al tipo de entrada seleccionado como se ve en la figura 72.

```
@article{semanticaGermanJLAMP,
  title = "A Theory of Reversibility for {E}rlang",
  author = {Lanesea, Ivan and Nishidab, Naoki and Palacios, Adri\'an and Vidal, Germ\'an},
  journal = "Journal of Logical and Algebraic Methods in Programming",
  volume = 100,
  pages = "71-97",
  note = "{\em To appear}. Available from \url{http://users.dsic.upv.es/~gvidal/german/jlamp18/paper.pdf}",
  year = {1998}
}
```

Figura 71: Fichero .bib antes de la modificación

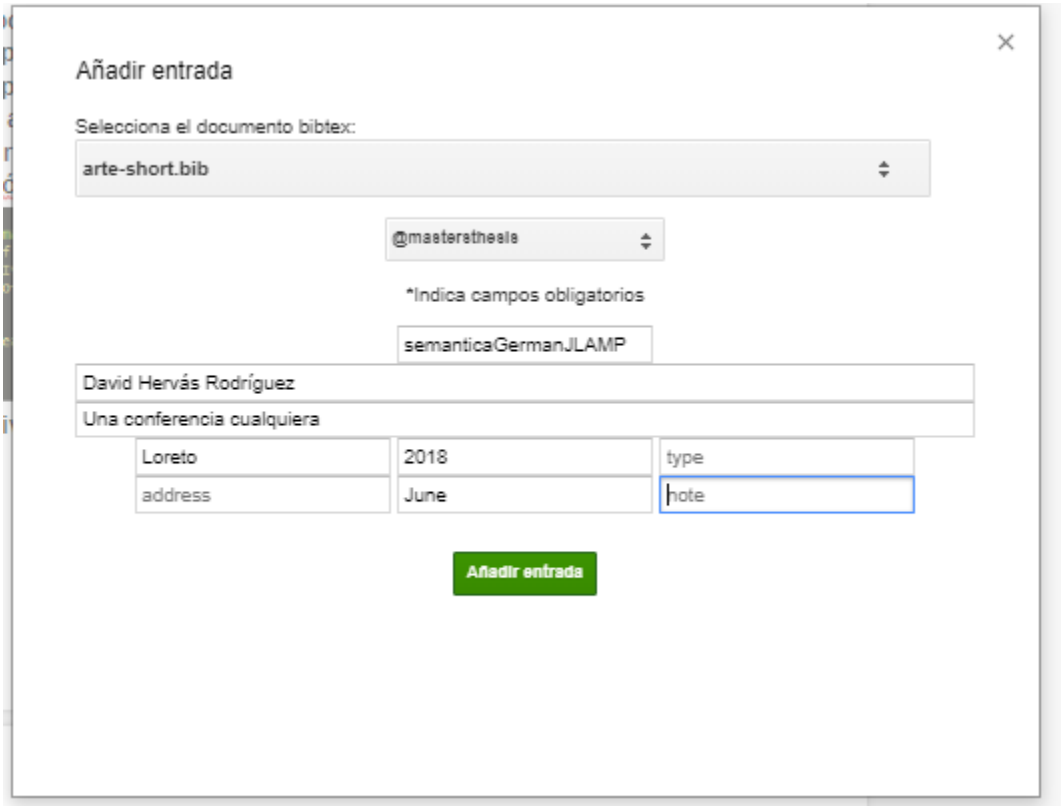


Figura 72: Ejemplo de campos añadidos

Para este ejemplo se intentará primero añadir una entrada con el ID repetido, de la forma que se obtendrá un mensaje de error en la ventana como el de la figura 73.

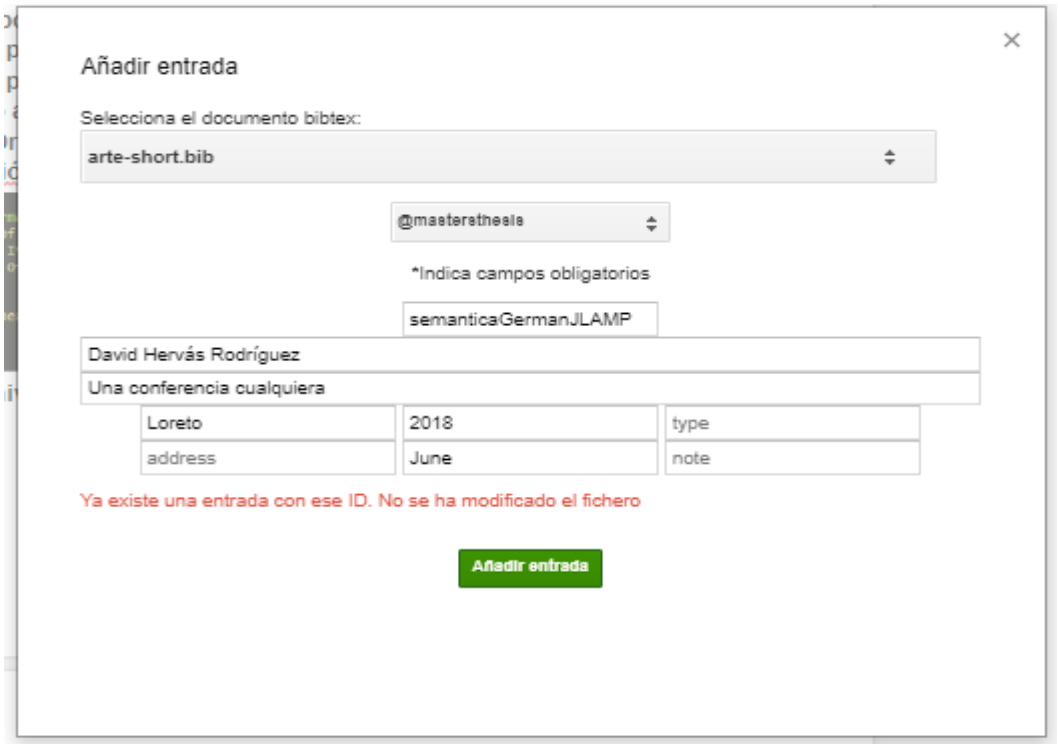


Figura 73: Mensaje de error por ID repetido

En el caso de que no hayamos rellenado un campo obligatorio (denotados con un asterisco *) se mostrará un mensaje de error diferente como el de la figura 74.

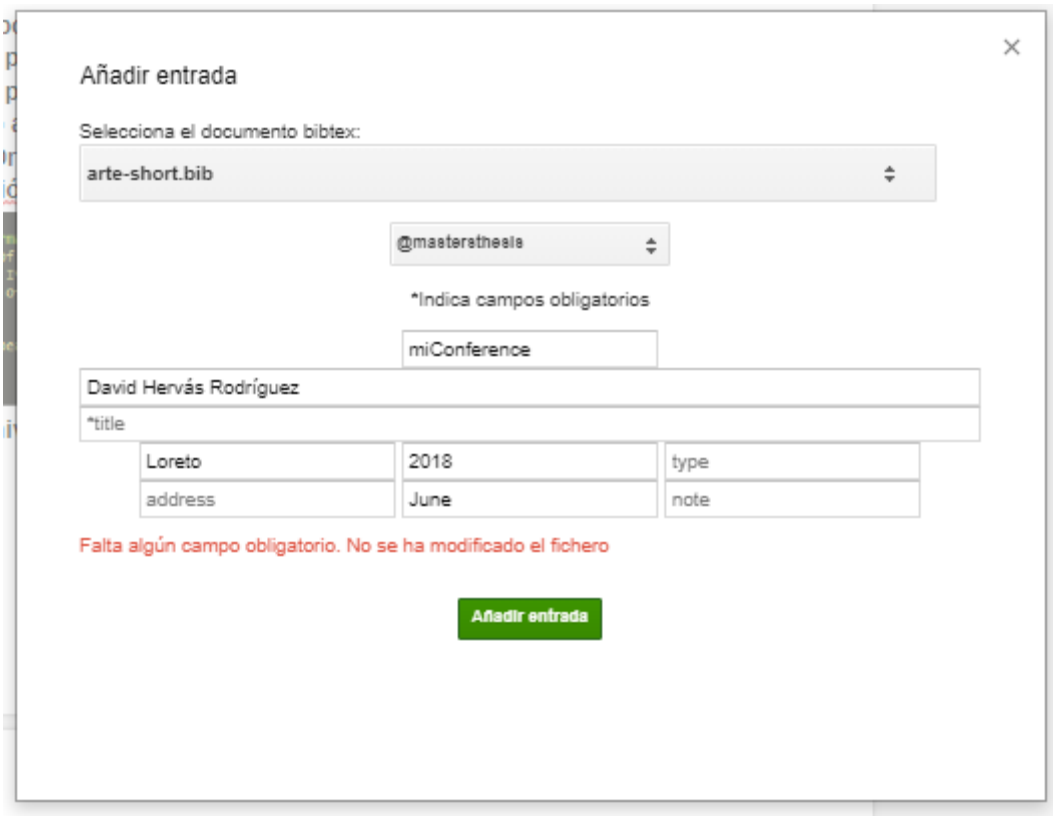


Figura 74: Mensaje de error por campo obligatorio vacío

Si se han introducido correctamente los datos y no hay otra cita en el fichero .bib con el mismo ID, se llevará a cabo el proceso de adición y se mostrará un mensaje de éxito como el representado en la figura 75.

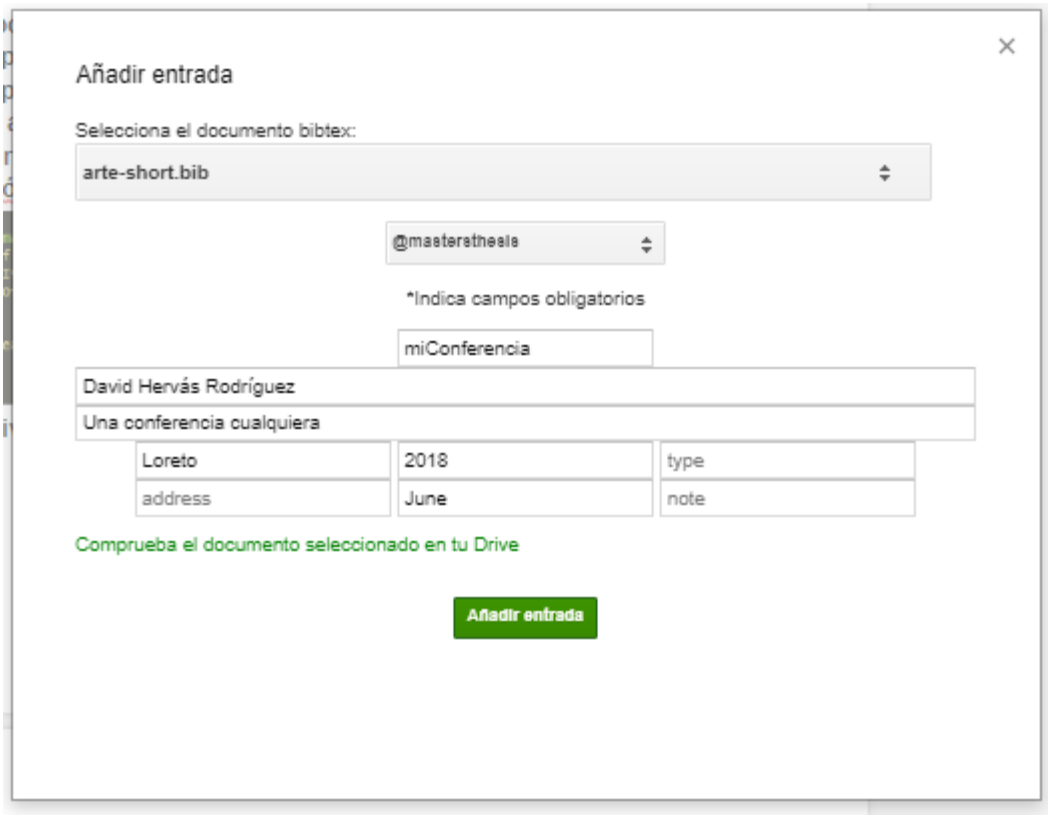


Figura 75: Mensaje de éxito al añadir una cita

En nuestro Drive veremos un fichero modificado recientemente de la forma que se ve en la figura 76, el cual contendrá la nueva entrada al final como se aprecia en la figura 77.

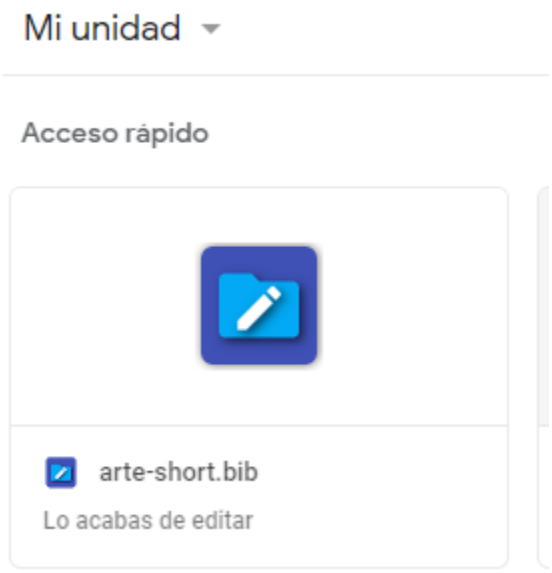


Figura 76: Fichero .bib modificado en Drive

```
@article{semanticaGermanJLAMP,
  title = "A Theory of Reversibility for {E}rlang",
  author = {Lanesea, Ivan and Nishidab, Naoki and Palacios, Adri\'an and Vidal, Germ\'an},
  journal = "Journal of Logical and Algebraic Methods in Programming",
  volume = 100,
  pages = "71-97",
  note = "{\em To appear}. Available from \url{http://users.dsic.upv.es/~gvidal/german/jlamp18/paper.pdf}",
  year = {1998}
}



@mastersthesis{miConferencia,
   author={David Hervás Rodríguez},
   title={Una conferencia cualquiera},
   school={Loreto},
   year={2018},
   month={June}
}
```

Figura 77: Contenido del fichero .bib modificado con la nueva entrada

5. CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se explican las conclusiones del TFG y el trabajo futuro que considero útil para ser implementado si se continuara con el desarrollo de este proyecto.

5.1. Conclusiones

Hoy en día es cada vez más común realizar investigaciones o análisis científicos los cuales deben mantener un cierto estilo y tener una gran densidad de información. Toda esa información, suele ser obtenida de fuentes externas al autor que escribe el análisis e indicar esas fuentes externas es una seña de calidad en un trabajo de este tipo. Por ello es conveniente crear métodos para promover la citación en este tipo de documentos, tanto para mantener la calidad del documento final, como para poder dar un reconocimiento a la fuente que provee de información.

El objetivo principal de este trabajo era mejorar un complemento de Google Docs de tal forma que permitiese una mayor variedad de funcionalidades con el fin de cubrir más necesidades de un usuario. La idea era añadir 3 modificaciones principales que se suplementarían con la corrección de *bugs* y la conservación de las funcionalidades anteriores, así como la compatibilización de estas con las nuevas modificaciones. Para la implementación contaba con los conocimientos obtenidos de diversas asignaturas estudiadas a lo largo de la carrera y que se detallarán más adelante, y con las propias guías del lenguaje Apps Script que Google pone a disposición de sus usuarios y, a partir de estos, se obtendrá una clara idea de cómo implementar las nuevas funciones.

A medida que se integraba de manera exitosa una funcionalidad, se realizaba un exhaustivo control de fallos mediante simulaciones de funcionamiento con archivos de longitud y contenido diferentes. Si estas simulaciones no devolvian el resultado esperado o devolvían un resultado erróneo, se procedía a revisar el código y, en una última instancia, a rehacer de nuevo la funcionalidad. En caso de que el resultado obtenido fuese correcto, se procedía a realizar una copia de seguridad y a actualizar el código en el repositorio *online* de GitHub, así como a actualizar el fichero de información del propio repositorio.

Este trabajo, ha supuesto una oportunidad para poder aplicar los conocimientos adquiridos en las distintas asignaturas que se han cursado a lo largo de la carrera a un trabajo con un gran potencial y un impacto real. Entre todas las asignaturas, hay que destacar las siguientes:

- **Fundamentos de la Programación y Tecnología de la Programación** que tienen una gran influencia en la forma de estructurar las diversas partes a modificar y los conocimientos sobre los distintos objetos del lenguaje JavaScript.

- **Métodos Matemáticos de la Ingeniería**, **Métodos Discretos y Lógica Matemática** y **Ampliación de Matemáticas** que me han proporcionado una visión más específica para aplicar expresiones y lógica booleanas al proyecto.

- **Ingeniería del Software** que me permitió tener una idea muy clara desde el principio de la jerarquía de las funcionalidades y el proceso de desarrollo de *testing*.

- **Evaluación de configuraciones** y **Auditoría Informática,** sobre todo, en cuanto a la optimización de los servicios y sus mejoras de cara a un mejor rendimiento.

- **Estructuras de Datos** y **Algoritmos** en una gran parte, me ha proporcionado la lógica para poder pensar en la aplicación de algoritmos tanto iterativos como recursivos y que, finalmente, han sido aplicados.

- **Bases de datos** y **Ampliación de Bases de Datos** más orientadas al trabajo futuro, aunque con presencia en el trabajo actual por el desarrollo de componentes web mediante HTML.

- **Aplicaciones Web** e **Ingeniería Web**, en las cuales aprendí todos los conocimientos necesarios tanto de HTML, CSS y JavaScript para poder haber realizado este trabajo.

- Para finalizar, la asignatura **Diseño de Sistemas Interactivos** que me otorgó la capacidad de dar un paso atrás y aplicar una visión general de cara a los usuarios para desarrollar un complemento más amigable y con menos carga de elementos.

En resumen, durante la realización de este TFG se han acometido de forma exitosa todos los objetivos propuestos desde el primer momento y que están detallados en la sección 3 de esta memoria habiendo superado exitosamente las pruebas de *test* realizadas para verificar el buen funcionamiento de la herramienta modificada.

5.2. Trabajo futuro

Con el objetivo de agilizar la generación de reportes bibliográficos y conseguir un proyecto más completo, se han considerado como trabajo futuro la implementación de los siguientes aspectos que no se cubrieron en la aplicación y final y que surgieron tanto en el proceso de implementación como al final del mismo.

- **Permitir un uso de base de datos para los archivos bibliográficos:** en vez de tener que seleccionar los archivos bibliográficos cada vez que se inicia el complemento, podría ser posible añadir una base de datos que contuviese las entradas de los ficheros bibliográficos seleccionados, de tal forma que una vez que el usuario se ha identificado en los servicios de Google, se le presente la opción de utilizar su base de datos personal. Dicha base de datos podría ser actualizada por el usuario desde el mismo aplicativo.

- **Ofrecer una mayor explicación en los mensajes de error y permitir una corrección rápida:** en el proyecto, se ofrecen actualmente mensajes de error, pero con un bajo nivel de detalle. Por ello se podrían realizar cambios de cara a mostrar qué archivo (si hay más de un archivo bibliográfico) contiene el fallo y no solamente el ID de la entrada asociada y permitir, mediante una ventana adicional, una forma de que el usuario corrija el error sin tener que borrar el archivo de su Drive, modificarlo y después volver a subirlo a Drive.

- **Comprobación automática al insertar una cita:** una implementación de una función en la ventana para añadir entradas que, de forma asíncrona y

dinámica, permita comprobar si el ID que ha introducido el usuario en el campo correspondiente ya existe. Actualmente, es durante el proceso de adición de la entrada donde se comprueba si existe ese ID en el fichero bibliográfico elegido pero la funcionalidad expuesta comprobaría si ya existe ese ID mientras el usuario sigue rellenando la información de la entrada.

- **Implementación del complemento adaptado a móviles:** a medida que pasa el tiempo, cada vez hay más personas que utilizan un dispositivo móvil por lo que, la aplicación del complemento a dispositivos móviles, sería una buena opción para resultar más atractiva.

- **Permitir un uso de árboles sintácticos para el filtrado:** esta mejora supondría un gran impulso al filtrado, sobre todo para el filtro por autores ya que la implementación de árboles sintácticos que fuesen evaluados para las expresiones del filtro y que devuelvan si una entrada es válida o no, permitiría agilizar el proceso de creación del reporte y, de la misma manera, optimizar el complemento.

- **Implementación de más estilos:** actualmente sólo hay soporte para el estilo *unsrt*, pero también sería útil permitir citar y crear reportes con los diferentes estilos que proporciona BibTeX como pueden ser *alpha, apalike* o *plain*, entre otros.

El principal factor que ha impedido llevar a cabo el desarrollo de estas funcionalidades ha sido la falta de tiempo ya que, como se ha expuesto anteriormente, algunas fueron pensadas durante la implementación de las demás y otras, después de finalizar la implementación. En el caso de la mejora descrita para implementar árboles sintácticos, el factor determinante ha sido no haber cursado una asignatura dedicada a compiladores, que me hubiese permitido conocer de manera más amplia el funcionamiento de dichos árboles de decisión.

5. CONCLUSIONS AND FUTURE WORK

In this chapter the conclusions of the project and the future work that I consider to be useful to be implanted if it was continued by the development of this project are explained.

5.1. Conclusions

Nowadays, it is increasingly common to conduct investigations or scientific analyses that must maintain a certain style and have a great density of information. All this information, it tends to be obtained from external sources to the author who writes the analysis, and indicating these external sources is a sign of honesty and of quality in a work of this type. For that reason, it is suitable to create methods to promote the citation in this type of documents, not only to support the quality of the final document, but also to be able to give a recognition to the primary source that provides the information.

The main aim of this work was to improve a Google Docs's add-on in such a way that it would allow a wider variety of functionalities in order to cover more needs of a user. The idea was to add three principal modifications complementing bug correction and, at the same time, maintaining previous functionalities, as well as the alignment of these with the new modifications. For the implementation I had the knowledge obtained from diverse subjects studied during the Degree, which will be detailed hereinafter, and with the own Google guides for the language Apps Script that are put at the disposal of his users and, from these, a clear idea was acquired of how to implement the new functions.

While a functionality was being integrated in a successful way, an exhaustive control of failures was done by means of simulations of functioning by files from different length and content. If these simulations were not returning the awaited result or they were returning an incorrect one, I started to check the code and, at last instance, I would redo again the functionality. In case the obtained result was correct, a backup was created followed by an update on the code contained in the online repository of GitHub, as well as an update on the file of information from the own repository.

This work has led to an opportunity to apply the knowledge acquired from different subjects studied along the Degree to a big potential and real-impact project. Among all subjects we must highlight the following:

- **Fundamentals of Programming** and **Computer Programming Technology** that had a great impact on how the parts to modify and the knowledge about different JavaScript objects were structured.

- **Mathematical methods for engineering**, **Discrete Mathematics and Mathematical Logic** and **Advanced Mathematics** gave me a more specific vision to apply boolean expressions and logic to the project.

- **Software Engineering** that allowed me to have since the beginning a clear idea about the hierarchy of the functionalities and the testing development procedure.

- **Evaluation of computer systems** and **Information Systems Audit** above all, in terms of service optimization and its improvements regarding a better performance.

- **Data structures and algorithms** mostly provided me the logic to think on the way to apply of algorithms, both iterative and recursive to the project that, in the end, have been applied.

- **Databases** and **Advanced Databases** mostly looking forward on the future work but with presence on the actual project due to web components development by using HTML.

- **Web applications** and **Web engineering**, on which I learnt all HTML, CSS and JavaScript knowledge that was necessary to carry out this project.

- To conclude, the subject called **Interactive Systems Development** that provided me the ability to step back and take an overlook from the user's perspective to develop a more user-friendly add-on with a small amount of window elements.

To sum up, during the accomplishment of this project the main objectives established at the beginning were successfully achieved and are detailed on the section 3 of this document having successfully passed the tests performed to verify the proper functioning of the modified tool.

5.2. Future work

With the aim of improving the generation of bibliographical reports and to obtain a more complete project, it was considered as future work the implementation of the following aspects that were not covered in the final application and that arose both during the process of implementation and at the end of that process.

- **Allow to use a database to keep the bibliographic files:** instead of having to select the desired files each time the add-on is launched, it will be a good alternative to use a database in which the entries from the bibliographic files are saved when they are first selected, so that the user that has previously logged into his Google account could decide to use its personal database. This personal database could also be updated by the user through the add-on itself.

- **Offer more detailed information on the error messages and allow a quick fix:** on the project, message errors are shown but they do not have a high detail level. That is the reason why some changes could be made in order to show which of the selected files (in case there are two or more) contains the error so that not only the ID of the incorrect entry is shown, but also allow, by an additional window, a way for the user to fix the error without having to replace the file on it's personal Drive by a new file previously corrected.

- **Dynamic check while adding a new entry:** an implementation of a function in the add entry window that, in an asynchronous and dynamic way, checks if the ID that the user has written on the corresponding field already exists. At the moment, during the addition process the check of the ID takes place but this functionality will check it while the user still fills in other information fields.

- **Support for mobile devices:** as time goes by, there are more and more people using a smartphone. That is why it will be a good choice to provide the add-on with smartphone compatibility, which will also make the tool more attractive.

- **Allow using syntactic trees in the filter procedure:** this improvement will be a great impulse to the filter procedure, mostly on the authors filter because it will evaluate the expression by the use of a syntactic tree so that the generation of the report will be speeded up and, in the same way, the add-on itself would be optimized.

- **Include more styles:** at the moment, the *unsrt* style is the only style supported but it will be useful to allow cite and generate reports with more BibTeX provided styles such as *alpha*, *apalike* or *plain* among others.

The main factor that has prevented from carrying out the development of these functionalities has been the lack of time since, as it has been exposed previously, some of them were thought during the implementation of the project and others, after finishing the implementation. In case of the improvement described to implement syntactic trees, the determinant factor has been not to have studied a subject dedicated to compilers, which would had allowed me to know deeply the functioning of the above mentioned decision trees.

6. BIBLIOGRAFÍA

[1] Ram, Shri, and John Paul Anbu K. 2014. “The Use of Bibliographic Management Software by Indian Library and Information Science Professionals.” *Reference Services Review* 42 (3): 499–513. doi:10.1108/RSR-08-2013-0041.

[2] “Bibme.” 2012. *Choice Reviews Online* 50 (02): 50–0594. doi:10.5860/CHOICE.50-0594.

[3] Datta, Dilip. 2017. *Latex in 24 Hours: A Practical Guide for Scientific Writing*. Cham, Switzerland: Springer. doi:10.1007/978-3-319-47831-9.

[4] Kottwitz, Stefan. 2011. *Latex Beginner's Guide*. Birmingham, UK: Packt.

[5] Lamont, Ian. 2015. *Google Drive & Docs in 30 Minutes: The Unofficial Guide to the New Google Drive, Docs, Sheets & Slides*. Second edition. ed. Place of publication not identified: I30 Media Corporation.

[6] Ferreira, James. 2014. *Google Apps Script*. 2Nd ed. ed. Sebastopol, CA: O'Reilly Media.

[7] https://www.andy-roberts.net/res/writing/latex/bibentries.pdf