Plataforma online para la consulta y búsqueda de datos bibliográficos sobre escritos ingleses del siglo XVIII (ECEG)

Autor: Carlos Martel Lamas

Tutor académico: Francisco Javier Carreras Riudavets

índice

Organización de la memoria

Veamos brevemente el contenido de los apartados del Trabajo de Fin de Máster:

- En la **Introducción** hablamos de la fuente del problema que dio lugar al Trabajo de Fin de Máster así como de la solución que se plantea. También la importancia del proyecto y sus objetivos.
- Por otro lado, en la lista de **objetivos** se destacan los objetivos generales y específicos que se pretenden alcanzar durante el desarrollo del proyecto.
- En el **estado del arte** hacemos una descripción de otras propuestas similares para tener diferentes puntos de referencia y exponer así el panorama actual del tratamiento de datos en aplicaciones o páginas web de temática similar a la nuestra.
- En la descripción de **recursos** necesarios exponemos cada uno de los aspectos que han sido indispensables para hacer realidad este proyecto; incluyendo una descripción superficial de los más relevantes para evidenciar, de esta forma, su importancia en el desarrollo.
- En la **metodología**, tratamos el modelo de ciclo de vida del proyecto que más se ha ajustado a nuestros requerimientos, no sin hablar de sus características principales y justificación. Posteriormente, en el **plan de trabajo** expondremos una tabla con las tareas planteadas para el proyecto y una duración estimada de las mismas.
- En el **análisis** hacemos énfasis en una de las fases más importantes del proyecto de manera pormenorizada, a saber, el estudio de los requisitos de usuario y del software o sistema.
- En el **diseño** describimos los preámbulos de la fase de implementación, como las clases principales (propias o de tercero), el modelo arquitectónico que se seguirá, los paquetes y módulos que se habrán implementado en la aplicación, así como los modelos de interfaz de usuario.
- En las **conclusiones** presentamos un breve resumen de todo lo expuesto en esta memoria a modo de recorrido, además de todo lo aprendido, con la consiguiente reflexión y valoración del proyecto.
- Por último, en la descripción del **trabajo futuro**, hablamos de las mejoras o extras que se irán incorporando al proyecto finalizada su fase de desarrollo.

Introducción

Contextualización - ¿Qué se entiende por gramática en el siglo XVIII?

Tanto hoy en día como en el siglo XVIII, cuando hablamos de gramática, nos referimos tanto a la disciplina como a un volumen o tratado que trate de gramática. Si decimos, por ejemplo, "Dejé la gramática de francés encima de la mesa", cualquiera entiende que dejamos atrás un libro sobre gramática francesa. Y si leemos en la introducción de un diccionario que el volumen contiene una gramática del español, igualmente asumimos que una sección del diccionario se dedica al estudio de la gramática. Esto último indica que podemos encontrar una gramática en un volumen de mayor tamaño que incluya otros conocimientos, generalmente de tipo lingüístico.

Como disciplina, la gramática, en términos generales, comprende dos niveles de la lingüística: la morfología y la sintaxis. En otras palabras, un estudiante de gramática estudia por un lado las distintas formas que puede adoptar una palabra (morfología) y la combinatoria de estas palabras en la construcción de oraciones (sintaxis). Así pues, si nos centramos en las lenguas española e inglesa, la morfología nominal estudia, por ejemplo, las distintas formas que la palabra niño puede adoptar dependiendo de su género (*niño – niña*) o de su número (*niño –niños*), si nos centramos en la morfología adjectival, la morfología de la lengua inglesa se ocupa de las distintas formas que toma un adjetivo para formar el comparativo o el superlativo (*nice – nicer – the nicest*), y si nuestro interés es la morfología verbal, observamos que en español es muy compleja en el presente simple (*como – comes – come – comemos – coméis – comen*), frente a la inglesa, que presenta únicamente una variación en la tercera personal del singular (*read – reads*). La sintaxis, por su parte, se interesa por las construcciones de oraciones. Así, por ejemplo, estudia las oraciones simples así como la unión de de oraciones en un mismo nivel (coordinación) o en niveles distintos de dependencia (subordinación).

En la Inglaterra del siglo XVIII, sin embargo, el concepto de gramática era mucho más flexible (Yáñez-Bouza 2015: 913-914) y estaba abierto a otras disciplinas que hoy consideramos distintas. Así, si bien es cierto que la mayor parte de las gramáticas del XVIII inglés se dedicaban principalmente a la *morfología* (también llamada etimología en esos momentos) y a la *sintaxis*, más de la mitad de las gramáticas incluía el estudio de la *ortografía* y, en menor proporción, el de la *prosodia* (término que hace alusión a la

pronunciación y a la versificación). En ocasiones la pronunciación se estudiaba en sección aparte y se denominaba entonces *ortopedia*, y la versificación se dejaba entonces solo para la prosodia, que en ocasiones también trataba la literatura, el estilo y la retórica (Yáñez-Bouza 2015: 922).

Por otro lado, las gramáticas incluían otras secciones que, aunque no eran estrictamente el objeto de estudio de las gramáticas, su inclusión las hacías más atractivas en un mercado editorial cada vez más interesado en la adquisición de gramáticas (Sundby, et al. 1991: 5;[1] Rodríguez-Álvarez 2016: 106), es lo que se ha denominado convencionalmente *subsidiary material o subsidiary contents* (Michael 1970: 195)[2], que incluía cuestiones tan variadas como instrucciones para leer en voz alta, listas de abreviaturas, breves historias de la lengua inglesa, instrucciones para redactar correctamente, reglas de puntuación, etc. El estudio de este material complementario ha ayudado a comprender los intereses y las necesidades que tenía un estudiante de la época para obtener una completa formación.

Contextualización - La base de datos ECEG y su buscador. Problema y solución.

"ECEG (*Eighteenth Century English Grammars*) es un nuevo recurso electrónico para el estudio de la tradición gramatical del siglo XVIII. Más concretamente, se trata de una base de datos que recoge, por primera vez, información sobre la gramática inglesa del siglo XVIII recogida de bibliografías, colecciones y estudios académicos anteriores publicados en los últimos cien años (1927-2008). En el mayor detalle posible, ECEG incluye no sólo información bibliográfica sobre gramática, sino también información biográfica sobre los autores de la gramática. Además, la información biobibliográfica ha sido minuciosamente codificada por campos temáticos como el año de publicación, el lugar de impresión, el público al que va dirigida, el género y la ocupación del autor, etc.

ECEG fue lanzado en diciembre de 2010 en formato electrónico gratuito, fácil de usar, navegable y con capacidad de búsqueda.

[1] Sundby, Bertil; Bjørge, Anne Kari & Haugland, Kari E. 1991. A Dictionary of English Normative Grammar 1700-1800. Amsterdam/Philadelphia: John Benjamins.

[2] Michael, Ian. 1970. English Grammatical Categories and the Tradition to 1800. Cambridge: Cambridge University Press.

Aunque se dirige principalmente a los estudiosos que trabajan en la tradición gramatical del siglo XVIII, se espera que la naturaleza biobibliográfica de la ECEG beneficie a los estudiantes e investigadores de otras disciplinas como los estudios literarios (por ejemplo, el interés por la vida de los gramáticos-escritores) y los estudios históricos (por ejemplo, el interés por la producción de libros y la historia de las publicaciones). En última instancia, en la creencia de que este siglo "fue una fase clave en el desarrollo de la lengua inglesa" (Görlach 2001: 12), también se espera que ECEG contribuya al proceso de "des-cinderelización" del siglo XVIII en la historia del inglés (ver Pérez-Guerra et al. 2007: 12-13).

La versión actual de la base de datos (2012) consta de 323 gramáticas escritas por 275 autores diferentes. Cada libro ha sido examinado a fondo y anotado en veintiún campos diferentes, agrupados temáticamente en tres categorías principales (gramáticas, autores, referencias). ECEG es altamente valorado por su diseño interdisciplinario e innovador, especialmente en lo que respecta al enfoque más amplio de la noción de "gramática inglesa".

ECEG fue alojado en la web de la Universidad de Manchester, usando HTML y Javascript del lado del cliente y PHP y MySQL del lado del servidor. Se crearon dos diseños en la aplicación basada en web: uno para la navegación en cada libro de gramática y otro para llevar a cabo la búsqueda de libros. Los datos obtenidos también se podía descargar desde cualquiera de los diseños en un archivo CSV. El formato original de la base de datos es una base de datos relacional en Microsoft Access que consta de tres "formularios" y veintiún "campos" separados. La versión actual fue migrada por última vez en julio de 2012. Contiene 323 artículos (gramáticas inglesas) escritos entre 1700 y 1800 por 275 autores diferentes.”[3]

Para poder llevar a cabo el uso de este valioso recurso, se precisa de un servicio web que brinde a la comunidad investigadora la capacidad de indagar entre los registros que hay disponibles mediante la posibilidad de hacer búsquedas a medida, flexibles y complejas que satisfagan el mayor abanico de combinaciones que el usuario final pueda formular. Este desarrollo es necesario puesto que el proyecto ya no está alojado en la web de la Universidad de Manchester por haber quedado descontinuado y sin soporte a actualizaciones. Esta necesidad ha estado sin cubrir desde la primavera de 2016.

[3] (Diciembre de 2019. Texto recuperado, traducido e interpretado de la página del proyecto en la web de la Universidad de Manchester)

Objetivos

Objetivos generales

Los objetivos generales que se quieren alcanzar con este proyecto son:

1. Crear una aplicación web que permita la búsqueda y tratamiento de los datos que hayan sido introducidos en la base de datos ECEG.
2. Promocionar y dar visibilidad a las actividades de I+D+i del IATEXT de la ULPGC.

Objetivos específicos

Los objetivos específicos que se quieren alcanzar con este proyecto son:

1. Auditar y rediseñar algunos aspectos de la base de datos ECEG.
2. Maximizar la accesibilidad de los contenidos a través de la organización de una interfaz lo más simple e intuitiva posible.
3. Potenciar la navegación a través de los contenidos, reflejando las interacciones disponibles entre las dos vistas del servicio web.
4. Permitir la exportación de los resultados de las búsquedas en diversos formatos así como decidir qué campos exportar.
5. Ofrecer un servicio que cumpla su labor de manera rápida y eficiente.
6. Llevar a cabo pruebas de rendimiento y así poder medir de manera cuantitativa el grado de eficiencia del proyecto.
7. Profundizar el aprendizaje en el campo de la programación web, concretamente en el framework ASP.NET, en el lenguaje de programación C# y el entorno de programación Microsoft Visual Studio, así como en el manejo del sistema de gestión de base de datos Microsoft Access y del lenguaje SQL.

Estado actual / Estado de la cuestión

A día de hoy, aún habiendo una evolución rápida y constante en los servicios que la informática puede proveer, se puede comprobar de una manera bastante simple que los formularios de búsqueda no están avanzando con la misma velocidad debido a la poca complejidad que los usuarios suelen necesitar a la hora de tener que buscar

información. Podemos comprobar como, normalmente, algunas de las búsquedas más complicadas se limitan a una serie de un tamaño relativamente moderado (10 campos como máximo) y prácticamente sin interacción entre ellos.

El problema aparece cuando, por cualquier razón, surge la necesidad de llevar a cabo una búsqueda compleja.

Las gramáticas inglesas del siglo XVIII, debido a su influencia, han tenido una gran relevancia para la evolución y el desarrollo de una de las lenguas más habladas en el mundo y que se ha convertido en el idioma de comunicación internacional. Sin embargo, este tipo de documentos son estudiados sólo parcialmente y con una metodología donde la tecnología no puede intervenir para facilitar el trabajo.

Hasta el momento, la única herramienta tecnológica que podía ayudar en este problema era el buscador *The ECEG-Database* de la Universidad de Manchester y así salvar el desfase entre el estudio de las gramáticas y la era tecnológica. La búsqueda era compleja y con un diseño algo desfasado. No obstante, permitía consultar la base de datos ECEG mediante cada uno de los 21 campos que existen. Sin embargo, debido a una actualización en la página web de la Universidad de Manchester, el proyecto quedó sin soporte.

The ECEP Database

Por otra parte, recientemente se ha empezado a desarrollar un proyecto que, si bien no pertenece al ámbito de las gramáticas inglesas del siglo XVIII, sí que trata sobre el siglo XVIII dentro de la lengua inglesa. Concretamente hablamos de la fonología. Al proyecto en cuestión se le llama ECEP (*Eighteenth Century English Phonology*). Según la página de la Base de Datos de Recursos sobre Corpus, "*ECEP es una base de datos diseñada para el estudio de la fonología del inglés del siglo XVIII, que permite a los usuarios investigar la distribución social, regional y léxica de las variantes fonológicas en el inglés del siglo XVIII. Sirve como un banco de datos para estudios cuantitativos y cualitativos, satisfaciendo así las demandas de la creciente comunidad de investigación en fonología histórica y dialectología en particular (por ejemplo, Honeybone & Salmons 2015) y en inglés moderno tardío en general (por ejemplo, Mugglestone 2003, Hickey 2010).*

La base de datos incorpora datos en forma de transcripciones de IPA de once diccionarios de pronunciación publicados en la segunda mitad del siglo XVIII. Hemos

anotado tantas de las 1.737 palabras ejemplo individuales utilizadas para ejemplificar los Juegos Léxicos Estándar de variantes vocálicas de John C. Wells (1982) como las que se pueden encontrar en las fuentes seleccionadas, a las que hemos añadido cinco juegos suplementarios de variantes consonantes (204 palabras ejemplo individuales).”[4]

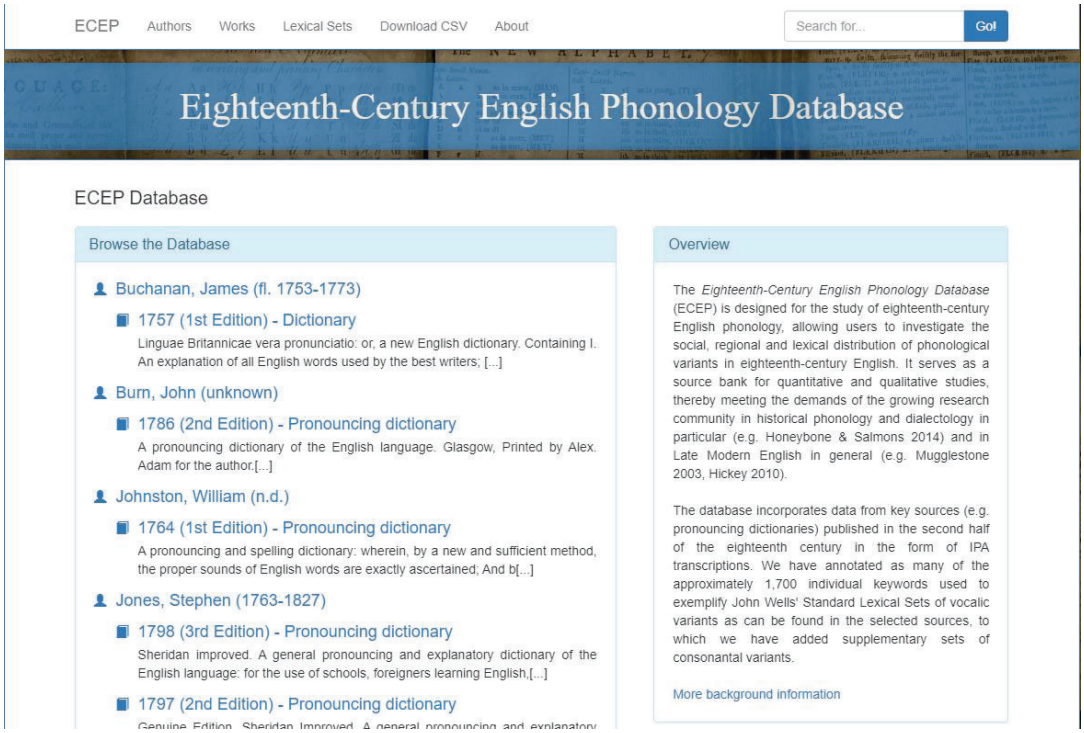
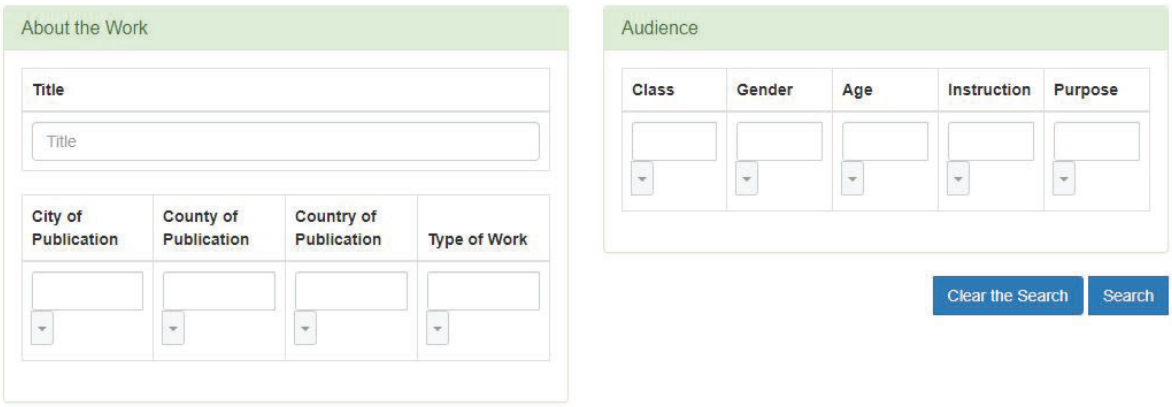


Figura 1.

Este servicio permite hacer una búsqueda dentro de su base de datos por muchos de sus parámetros, ya sea buscando dentro de los campos del tipo de obra y su audiencia objetivo o por las diferentes características de las transcripciones que están alojadas en la base de datos.



[4] Texto extraído y traducido de http://www.helsinki.fi/varieng/CoRD/corpora/ECEP/index.html

Figura 2.

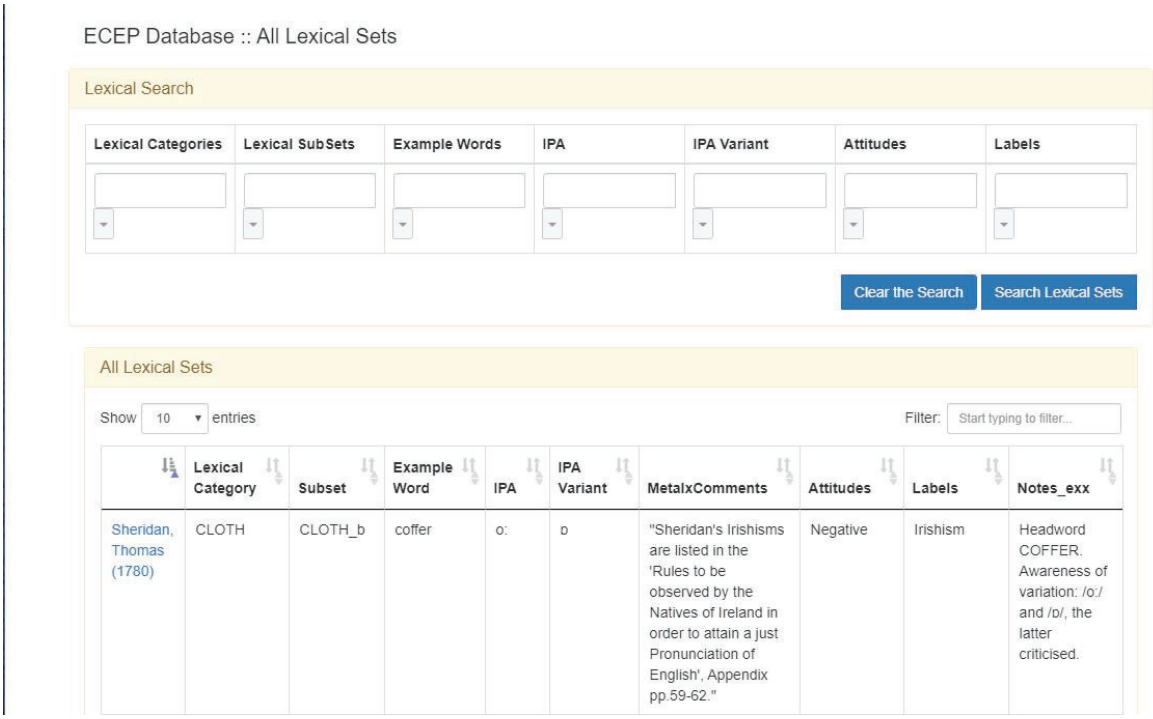


Figura 3.

Desafortunadamente, aunque se pueden combinar las diferentes características que están presentes en los formularios para realizar la búsqueda, en la actualidad permiten solo introducir un valor en cada característica.

Veamos el resumen de prestaciones:

- La distribución y organización de la página web es simple e intuitiva.
- Las listas de resultados están paginadas para una mejor navegación.
- No tiene una sección de ayuda para el usuario.
- La página web tiene ciertas características responsive (funciona bien en ordenador y si usamos un dispositivo móvil en horizontal. Aunque la tabla de resultados no se redimensione, el resto de elementos sí que se adaptan) .
- El contenido solo se encuentra disponible en inglés.

Phi Latin Texts

En lo que a buscadores se refiere, cabe mencionar también el presente en la web de Phi Latin Texts[5].

Lo primero que aparece según accedemos a la página y aceptamos los términos de la licencia es un listado de los autores disponibles ordenados alfabéticamente (no siempre por el nombre, a veces por su apellido). Como podemos observar en la figura a continuación.

Figura 4.

Cuando accedemos a uno de los autores podremos ver un listado de las obras disponibles. En la figura siguiente podremos observar las obras del autor **Lucius Accius.**

[5] Web del proyecto en https://latin.packhum.org/browse

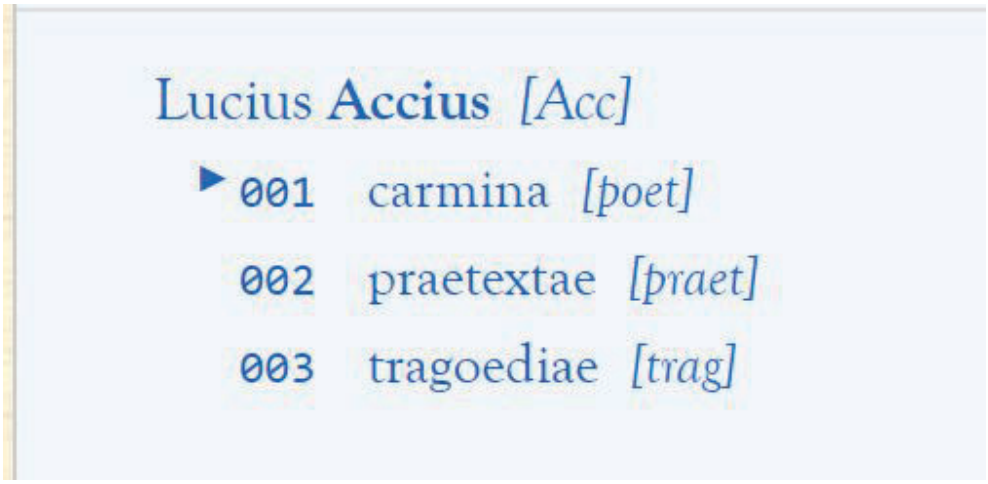


Figura 5.

Y una vez seleccionemos la obra que queramos, la veremos transcrita en la web. Lo podemos comprobar en la siguiente figura.

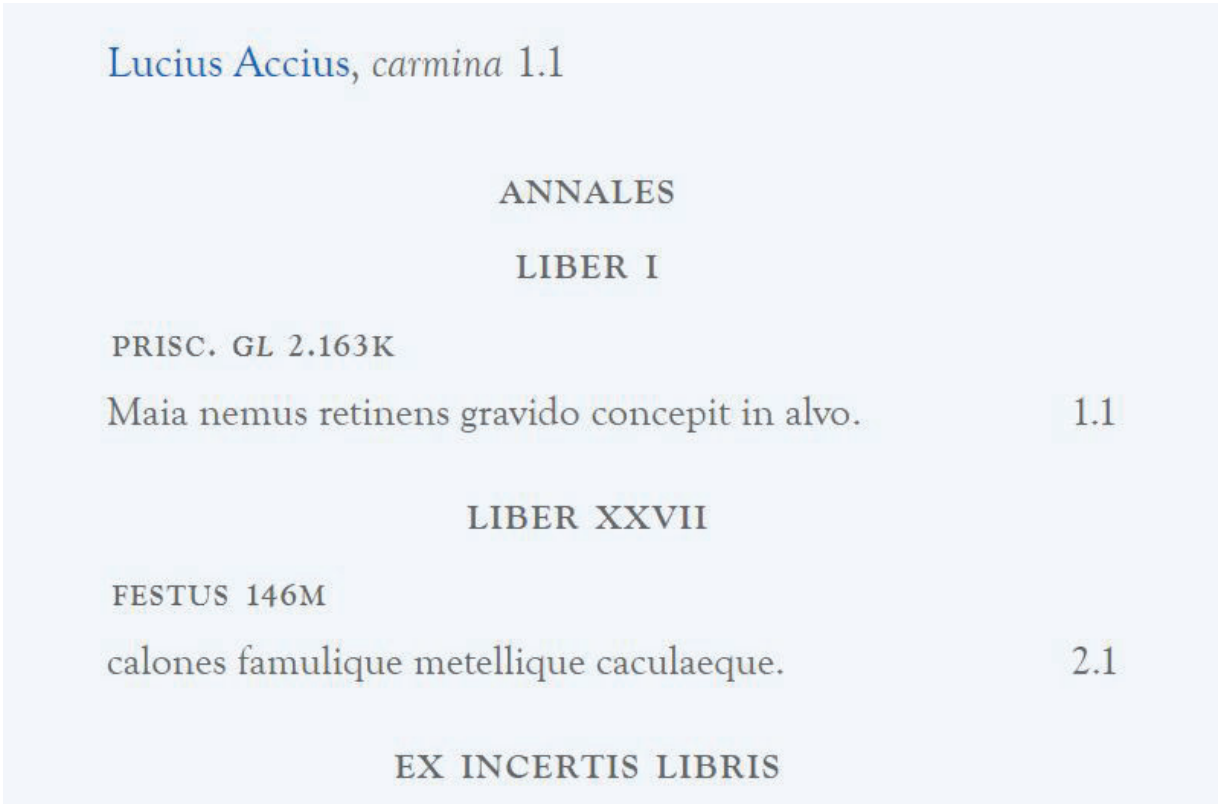


Figura 6.

También podemos hacer click en el nombre del autor para ver la lista de autores que existen dentro del sistema así como sus obras.

Figura 7.

La parte superior de la página está ocupada por un menú de navegación mediante el cual podremos acceder a diferentes secciones de la web. La segunda pestaña nos permite acceder al buscador. Si buscáramos, por ejemplo, la palabra "amor", encontraría extractos con esta misma palabra o con la palabra "amorem" o "amore", entre otras, como puede observarse en la siguiente figura. Es posible variar el comportamiento por defecto mediante el uso de operadores, como veremos más adelante.

Figura 8.

Además, podemos buscar más de una palabra, pero hay que tener en cuenta el orden en el que se escriben. Si buscáramos, por ejemplo, “amor mater”, no encontraría resultado alguno (figura 3.23), a pesar de que ambas palabras se encuentren dentro de un mismo extracto pero en diferente orden. Es decir, que por defecto no se realiza una búsqueda de palabras completas, sino de sintagmas o subcadenas. Aclaramos, nuevamente, que este comportamiento puede variar mediante el uso de operadores.

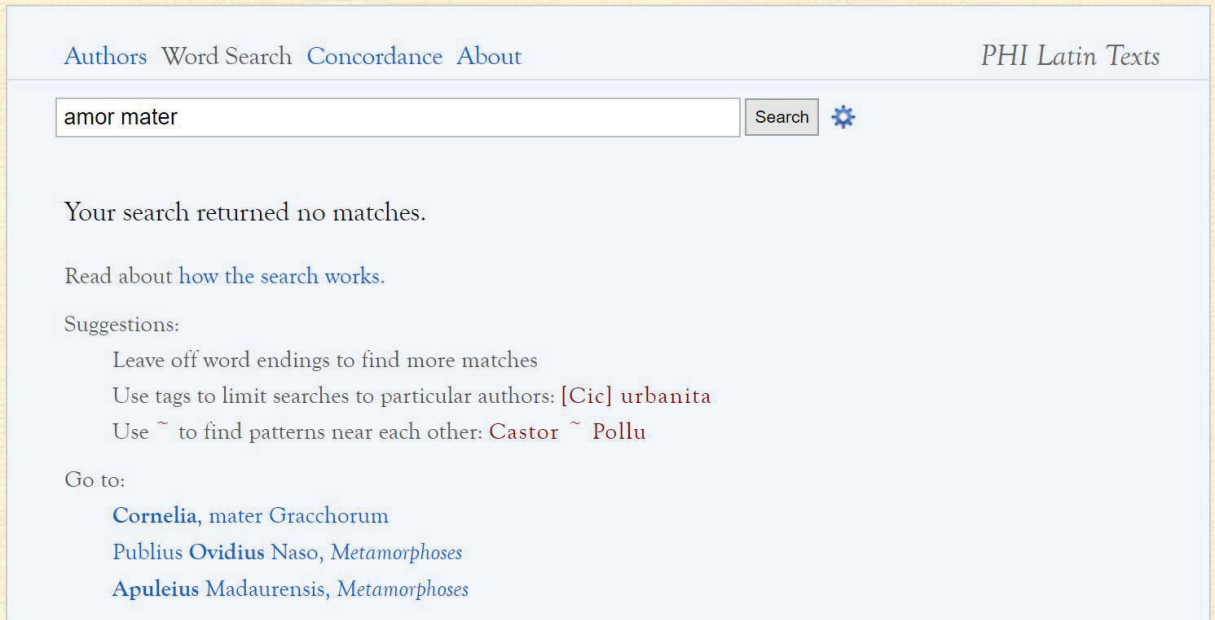


Figura 9.

En cambio, si la búsqueda la cambiamos por “mater amor”, sí que podemos encontrar las obras que estábamos buscando en un principio.

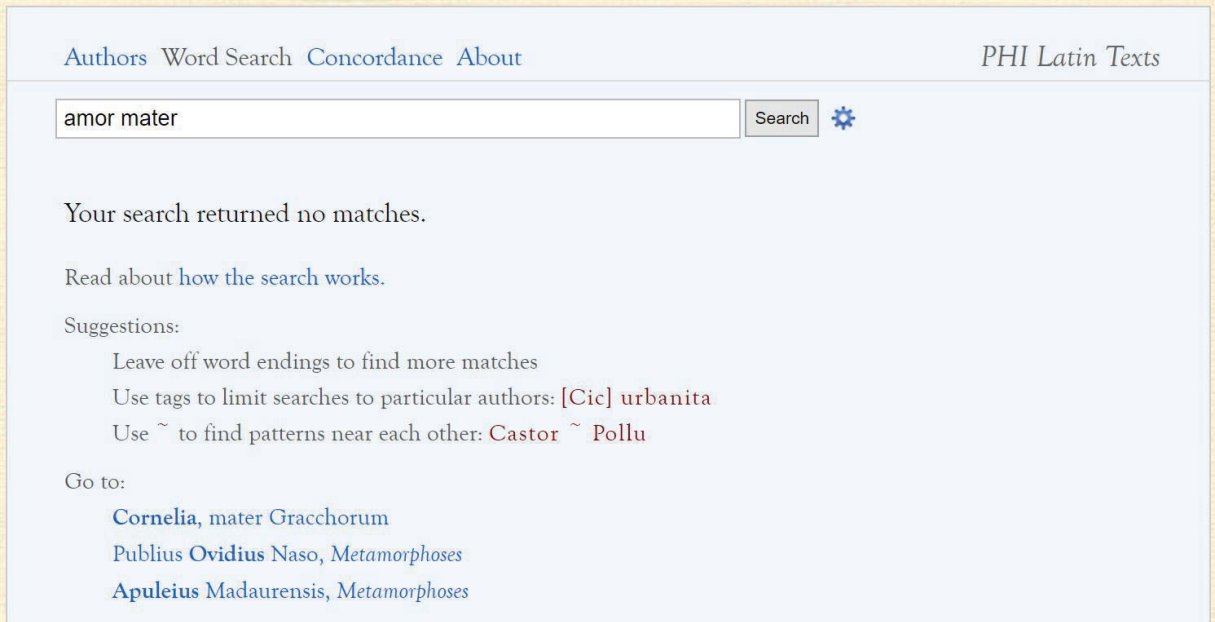


Figura 10.

Cuando no se encuentran resultados se nos da la opción de consultar la guía de ayuda del buscador (también encontramos otro enlace a esta misma guía en la cuarta pestaña del menú de navegación).

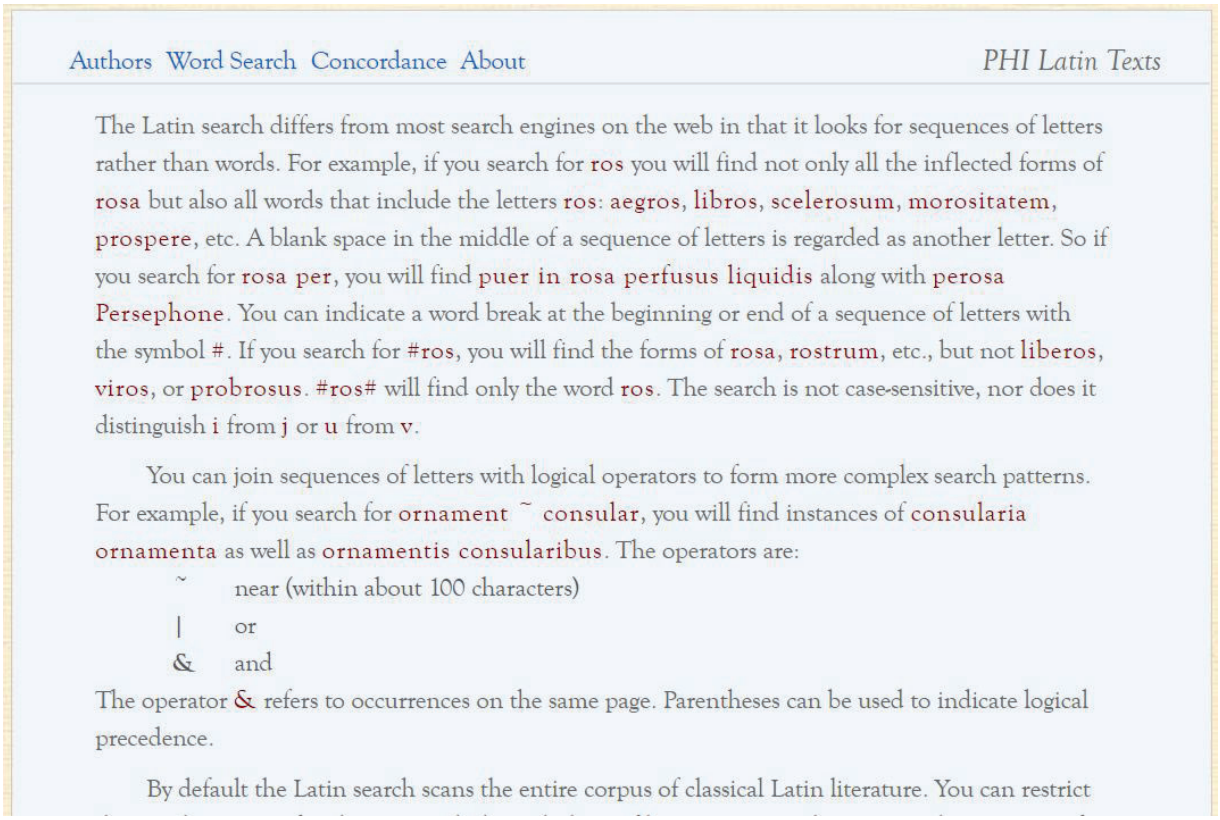


Figura 11.

Por otra parte, es posible llegar a un resumen de la ayuda al pulsar sobre el icono del engranaje que hay al lado del campo de texto del buscador.

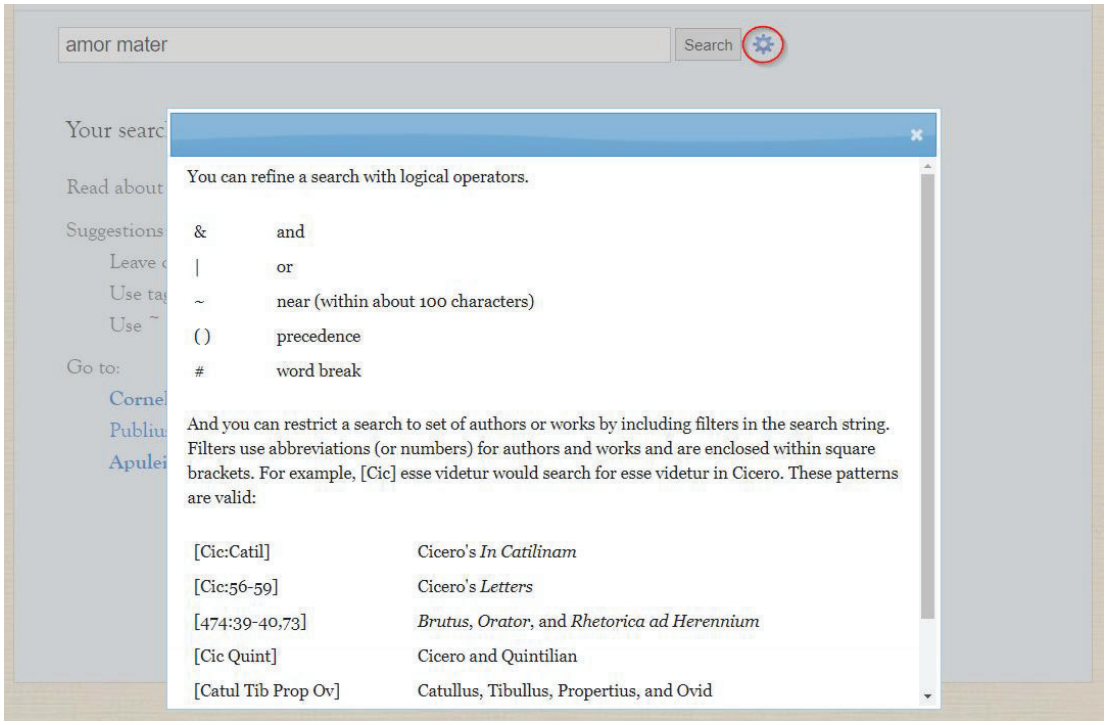


Figura 12.

Podemos destacar que la búsqueda de palabras del sistema es mediante una búsqueda aproximada de la cadena completa del texto, no de cadenas por separado. Es decir, el espacio no actúa como un separador.

Gracias al uso de operadores es posible buscar palabras para un determinado autor u obra metiendo uno o ambos (autor u obra) entre corchetes. Podemos referenciar estos mediante abreviaturas o mediante el nombre/título completo. También es posible aplicar un código numérico que puede consultarse en la lista de autores y obras de la que hablamos inicialmente (Figura 7) . En la Figura 13 podemos ver los resultados para la búsqueda de la cadena “in amor” más el autor Propertius.

Figura 13.

Como ya hemos hablado, al no buscar palabras completas admite que una palabra esté dentro de otra. Por ejemplo, si buscamos “ros” encontrará derivados como “rosa”, “libros”, “prospere”, etc. Pero, si tuviéramos que buscar la palabra “ros” como palabra completa, tendríamos que utilizar el operador # y escribir “#ros#”. Se permite el uso de este operador en uno solo de los lados. Por ejemplo, “#ros” buscaría solo palabras que terminasen en “ros” o la propia palabra completa en su defecto.

Otros operadores admitidos son “&” (conjunción) y “|” (disyunción). También se admite el uso de paréntesis para establecer precedencia de operadores y aplicar cierto orden en la búsqueda. Si quisiéramos buscar cadenas de palabras sin que el buscador tenga en cuenta el orden (por ejemplo, buscar “in amor” pero sin que tuviera que aparecer una seguida de la otra), habría que hacer uso del operador “~” en medio de ambas. En la Figura 14 se ilustra un ejemplo en el que buscamos “ornamentis consular ~ex”, debido a que queremos que las dos primeras palabras aparezcan una tras la otra, haciendo una posible excepción con la tercera.

Figura 14.

Un dato interesante a mencionar es que si hacemos doble click sobre algunas de las palabras de un extracto, se llevará la búsqueda automáticamente. En el caso de haber demasiados resultados, la lista se pagina.

La tercera sección se diferencia de la segunda en el modo en el que se representan los resultados. Si buscáramos la palabra “amor” (Figura 15), se mostraría una lista ordenada alfabéticamente de todas las coincidencias. Resulta de sumo interés por el modo en el que se representan las transformaciones o variaciones que va sufriendo dicha palabra.

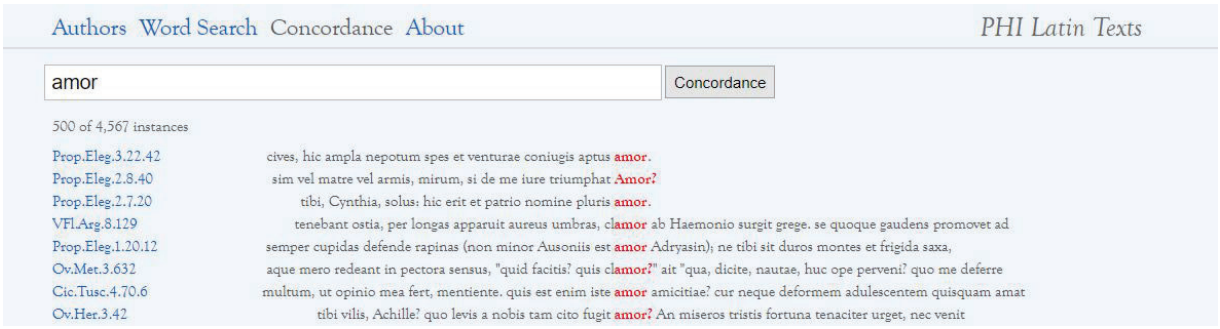


Figura 15.

La potencia de este buscador es igual al que nos encontramos en la segunda sección, pero en este caso carece de sentido buscar cadenas de palabras, a no ser que queramos filtrar por autor, obra o ambos, ya que lo que verdaderamente se busca aquí es ver la transformación que va sufriendo una palabra en los distintos extractos.

La cuarta sección es meramente informativa. Nos da información general de la aplicación web y proporciona acceso a la ayuda para los buscadores explicados anteriormente.

Veamos el resumen de prestaciones:

- La distribución y organización de la página web es simple e intuitiva.
- La lista de autores no está paginada y no es posible aplicar diferentes criterios de ordenación: los resultados siempre se muestran alfabéticamente.
- La ayuda del buscador está muy bien detallada.
- A pesar de la ayuda, el filtro de búsqueda es muy complicado. Se basa demasiado en el uso de operadores y si el usuario no leyera previamente la ayuda, le resultaría difícil realizar búsquedas compuestas si quisiéramos incluir autores u obras.
- Quizás fuera buena idea darle al usuario un buscador más intuitivo para búsquedas simples y otro para búsquedas más complejas o avanzadas.
- La tercera sección, concordancia, es muy interesante.
- No es una página web *responsive* (el contenido no se adapta diferentes tamaños del navegador, lo que dificulta la navegación en tablets y smartphones).

- El contenido solo se encuentra disponible en inglés.

Como hemos podido comprobar, la organización de la información textual en bases de datos con buscadores simples o complejos es ya muy frecuente en los estudios filológicos, especialmente en el campo de los estudios de corpus o de autores, épocas o géneros literarios[6]. No obstante, en los puntos anteriores pudo verse reflejada la necesidad de un buscador que pueda ser flexible y capaz de poder dar respuesta a filtros mucho más complejos y con campos combinados entre sí sin entrar a un grado de complejidad abrumador para el usuario.

Aportaciones / Importancia del proyecto

El proyecto de la base de datos ECEG y su buscador ha podido ser utilizado por diversos equipos de investigación y ha estado presente en un gran número de artículos que encontraron su servicio como una fuente fundamental de ayuda. De entre las investigaciones que se han llevado a cabo gracias a esto podemos destacar algunas de las siguientes:

- Domínguez-Rodríguez, M. Victoria. 2016. "A Corpus-based study of abbreviations in eighteenth-century English grammars". *English Studies* 97 (5): 528-545.
- Domínguez-Rodríguez, M. Victoria. 2017. "Author(itie)s and sources in the prefatory matter to eighteenth-century English grammar for children". *Atlantis. Journal of the Spanish Association of Anglo-American Studies 39* (2): 125-145.
- Domínguez-Rodríguez, M. Victoria & Rodríguez Álvarez, Alicia. 2015. "'The reader is desired to observe…'. Metacomments in the preface to English school grammars of the eighteenth century". *Journal of Historical Pragmatics* 16 (1): 86-108.
- Fernández-Martínez, Dolores. 2014. "Eighteenth-century female English grammar writers: Their 'critical' voice in the prefaces to their Grammars". *Nordic Journal of English Studies* 13 (1): 78-103.
- Fernández-Martínez, D. 2016. "Genre Structure in the Prefaces to Eighteenth-Century English Grammars: A Study on a Selection of Prefaces from the ECEG Database". *Neuphilologische Mitteilungen* 117 (2): 289-313

[6] S. Rampsay, "Database", S. Schreibman, R. Siemens, J Unsworth (eds.) A Companion on Digital Humanities, Malden-Oxford-Victoria, 2004, pp. 177-197

- Rodríguez-Álvarez, Alicia. 2016. "'Tis not so easy a matter to read well": directions for reading aloud in eighteenth-century English grammars". *Moderna Sprak* 110 (2): 105-132.
- Rodríguez-Álvarez, Alicia. 2017. "An approach to the historical sketches of the English language in eighteenth-century grammars of English". *Language & History* 60 (2): 79-94.
- Rodríguez-Álvarez, Alicia. 2018. "Cataloguing the first histories of the English language written from the late 16th to the end of the 18th century". *Historiographia Lingüistica* 45 (1/2): 99-132.
- Rodríguez-Álvarez, Alicia & Rodríguez Gil, María Esther. 2013. "Common topics in eighteenth-century prefaces to English grammars: An application of the ECEG database". *Transactions of the Philological Society* 111 (2): 202-221.
- Rodríguez-Gil, María Esther. 2013. "Readers in eighteenth-century teaching grammars". En: Rosario Arias, Miriam López Rodríguez, Antonio Moreno Ortiz y Chantal Pérez Hernández (eds.). *Hopes and Fears: English and American Studies in Spain. AEDEAN 36 Proceedings*. Málaga: Departamento de Filología Inglesa, Francesa y Alemana, Universidad de Málaga, 272-278.
- Rodríguez-Gil, María Esther & Yáñez-Bouza, Nuria. "The ECEG-database: a bio-bibliographical approach to the study of eighteenth-century English grammars". En: Ingrid Tieken-Boon van Ostade & Wim van der Wurff (eds.). *Current Issues in Late Modern English*. Bern/Berlin: Peter Lang, 153-182.
- Yáñez-Bouza, Nuria. 2011. "Mapping 18th-century grammar writers in the British Isles (and beyond)". *Studies in Variation, Contacts and Change in English 7. Special Issue.: How to Deal with Data: Problems and Approaches to the Investigation of the English Language over Time and Space 7* http://www.helsinki.fi/varieng/series/volumes/07/yanez-bouza/
- Yáñez-Bouza, Nuria. 2012. "Grammar writing and provincial grammar printing in the eighteenth-century Birtish Isles". *Transactions of the Philological Society* 110 (1): 34-63.
- Yáñez-Bouza, Nuria. 2015. "Senses of 'grammar' in the eighteenth century English tradition". *English Studies* 96 (8): 913-943.
- Yáñez-Bouza, Nuria. 2018. "Grammar writing in the eighteenth century". En: Terttu Nevalainen, Minna palander-Collin & Tanja Säily (eds.). *Patterns of Change*

in 18th-century English. A Sociolinguistic Approach. Amsterdam/Philadelphia: John Benjamins, 27-42

Recursos

En esta sección vamos a hacer un recorrido sobre los distintos recursos tecnológicos que nos han sido requeridos para poder desarrollar este proyecto, tanto en lo que se refiere a aquellos aspectos relacionados con la programación, como aquellos relacionados con la elaboración de la documentación, por no mencionar la instalación y el posterior mantenimiento.

Recursos hardware

Para realizar este proyecto, hemos hecho uso de un ordenador de sobremesa para el desarrollo web e implementación del servidor así como para también la escritura de la memoria y la presentación del proyecto. También se ofrecen las características mínimas requeridas del servidor en el que irá alojada la aplicación web.

Ordenador de sobremesa:

- Windows 10 Home
- 16Gb memoria RAM DDR4
- Procesador Intel Core i7 6700 3.4Ghz
- Disco duro SSD 256Gb
- Disco duro secundario de 2Tb

Servidor

- Windows Server 2016 Datacenter
- 32Gb memoria RAM
- Procesador Intel Xeon Gold 6132 2.59Ghz (2 unidades)
- Disco duro mecánico de 1Tb

Recursos software

Microsoft Visual Studio 2019

Figura 16.

Microsoft Visual Studio es un Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación, tales como C++, C#, Visual Basic, Java, Python, Ruby y PHP. También soporta frameworks como ASP.NET, Bootstrap, jQuery, Django, etc.

Visual Studio permite a los desarrolladores crear páginas y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET. De esta forma, es posible crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, consolas, etc.

Microsoft Access 2016

Figura 17.

Microsoft Access es un sistema de gestión de base de datos de Microsoft. Combina el motor para bases de datos Microsoft Jet Database Engine (sobre el que muchas aplicaciones de Microsoft han sido construidas) con una interfaz gráfica. Forma parte del popular paquete de aplicaciones Microsoft Office.

Microsoft Access almacena datos utilizando su propio formato, formato basado en el motor Microsoft Jet Database Engine. También es posible importar o enlazar los datos almacenados en una base de datos creada en Access con datos alojados en otras aplicaciones o bases de datos.

Navegadores web

Para comprobar el correcto funcionamiento de la aplicación web resulta necesario ejecutarla en diferentes navegadores. Para este proyecto se han considerado dos de los más populares: Mozilla Firefox y Google Chrome.

Figura 18.

A pesar de la existencia de un estándar, el comportamiento de la ejecución en uno o en otro puede variar en mayor o menor medida, de modo que es necesario realizar ejecuciones en el mayor abanico de navegadores posible para controlar estas posibles variaciones.

- **Google Chrome** es un navegador web desarrollado por Google. Fue compilado con base en varios componentes y frameworks de código abierto. Está disponible de forma gratuita y cuenta con más de 750 millones de usuarios. Es considerado el navegador más utilizado de la Web. Actualmente se sitúa con una cuota del mercado cercana al 54%. A día de hoy está disponible para los sistemas operativos Windows, OS X, iOS, Linux y Android.
- **Mozilla Firefox** es un navegador web libre y de código abierto, desarrollado por Linux, Android, OS X y Microsoft Windows. Entre sus principales características se incluyen la tradicional navegación por pestañas, el corrector ortográfico, la administración de descargas, los marcadores dinámicos, etc. Se pueden incluir nuevas funcionalidades a través de complementos desarrollados por la propia compañía o por terceros, ya sean estos aficionados o comerciales.

Google Docs, Google Sheets y Google Slides

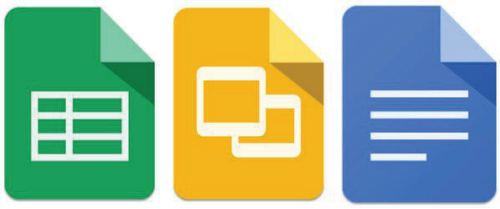


Figura 19.

Google Docs es un procesador de texto y Google Sheets permite crear hojas de cálculo. Junto con Google Slides, para crear presentaciones, forman un paquete software gratuito creado por Google. Este paquete permite a los usuarios crear y editar ficheros online, todo ello con la opción de colaborar con otros usuarios en tiempo real a través de Internet.

Estas tres aplicaciones se encuentran disponibles tanto como aplicaciones web como aplicaciones para dispositivos móviles bajo el sistema operativo Android e iOS. Son compatibles con ficheros del formato de Microsoft Office.

El paquete se encuentra completamente integrado en el servicio Google Drive. Todos los ficheros creados con alguna de estas tres aplicaciones son, por defecto, almacenados en la unidad de Google Drive del usuario propietario.

Mientras que Google Docs ha sido altamente criticado por las carencias funcionales en comparación con el servicio análogo de Microsoft Office, también ha sido elogiado por su simplicidad.

Visual Paradigm

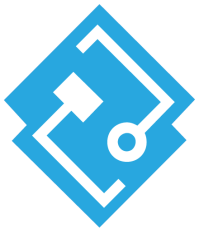


Figura 20.

Visual Paradigm Online Express Edition es un software de dibujo online gratuito que soporta diagramas de clase, diagramas UML, herramientas ERD y herramientas de organigramas. Presenta un editor simple pero poderoso que le permite crear un diagrama de clases de manera rápida y fácilmente. El editor que nos presenta no tiene anuncios y tampoco tiene límite para el número de diagramas o formas que añadir a las plantillas.

Se ha escogido este servicio online también por el hecho de que permite una fácil integración con los recursos de Google Drive tal como Google Docs. Los diagramas de clase presentados en esta memoria se han llevado a cabo gracias al uso de esta herramienta online.

Inkscape

Figura 21.

Inkscape es un editor de gráficos vectoriales de código abierto similar a Adobe Illustrator, Corel Draw, Freehand o Xara X. Lo que distingue a Inkscape es el uso de Scalable Vector Graphics (SVG), un estándar W3C basado en XML abierto, como formato nativo.

Este software es muy útil en cuanto a la confección de aspectos gráficos particulares tales como la creación del logo del proyecto, y diversos recursos gráficos que se tuvieran que crear a mano si no existiera un sustituto ya desarrollado.

Balsamiq Mockups

Figura 22.

Balsamiq Mockups es una aplicación destinada a la creación de bocetos gráficos de interfaces digitales, también conocidos como *mockups*. Cuenta con una amplia librería bocetos de controles, como ventanas de navegador, reproductores multimedia, etc.

Si el usuario tuviera que incluir algún control que no estuviera en el abanico de controles ofrecido por la aplicación, tiene la opción de incluirlo como imagen, descargarlo de *Mockups To Go* (comunidad que ofrece un repositorio online perteneciente a la propia aplicación para descargar nuevos controles que no se incluyen en el paquete de instalación) o enviar una solicitud a los desarrolladores para que lo incorporen en un futuro.

Función	Versión utilizada
Entorno de desarrollo	Microsoft Visual Studio 2019
Sistema de gestión de Base de Datos	Microsoft Access 2016
Navegadores web	Google Chrome y Mozilla Firefox
Elaborado de diagramas de clase	Visual Paradigm
Elaboración de la memoria	Google Docs

Elaboración de la presentación	Google Slides
Creación de elementos gráficos	Inkscape
Creación de los mockups	Balsamiq Mockups

Tabla 1.

Metodología

En este apartado de tratará la metodología aplicada, así como la creación de las tareas que se llevarán a cabo a partir de la misma y, por último, se elaborará una estimación temporal de lo que se tiene que llevar a cabo.

El modelo de ciclo de vida es un marco de referencia que contiene los procesos, las actividades y las tareas involucradas en el desarrollo: la vida del sistema desde la definición de los requisitos hasta la finalización de su uso. Se considera una actividad como un conjunto de tareas y una tarea como una acción atómica que transforma entradas en salidas.

Veamos los factores que hemos tenido en cuenta en la elección del modelo de ciclo de vida:

- La naturaleza del proyecto y de sus aplicaciones.
- Los métodos y las herramientas que se van a utilizar.
- Los controles y entregas que requieran.

El ciclo de desarrollo software se utiliza comúnmente para estructurar las actividades que se llevan a cabo en el desarrollo de un producto software. A pesar de la no existencia de un acuerdo acerca del uso y la forma del modelo, este sigue siendo útil para la comprensión y futuro control del proceso de desarrollo software.

Debido a las características del proyecto, el modelo de ciclo de vida elegido ha sido el **Modelo de Prototipos.**

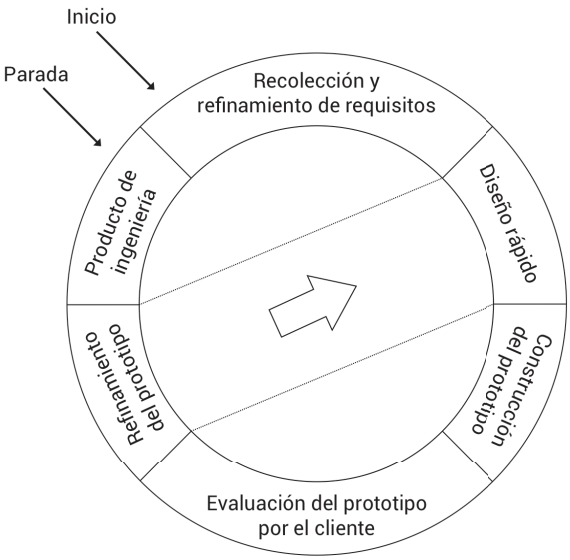


Figura 23.

Descripción y justificación

La metodología utilizada pertenece a los modelos de desarrollo evolutivo. El modelo de prototipos es común a muchas disciplinas de la ingeniería y facilita la creación del modelo software a construir. Ha sido elegido porque se conocen los objetivos generales para el software que se desea desarrollar.

Esta metodología ofrece un mejor enfoque al desarrollador cuando no se está seguro de la eficacia del software, de la adaptabilidad o de la forma que debiera tomar la interacción humano-máquina, centrándose en el diseño rápido de la representación de aquellos aspectos del software que serán visibles para el usuario final.

Posteriormente, el diseño conduce a la construcción de un prototipo, que será evaluado por los tutores para una posible retroalimentación, gracias a la cual se refinan los requisitos del software. Esto permite que el estudiante (desarrollador) entienda mejor lo que se debe hacer que los tutores vean resultados a corto plazo y puedan comprobar la trayectoria del desarrollador en las diferentes etapas del proceso de desarrollo.

Ventajas

El modelo de prototipos presenta las siguientes ventajas:

- Permite identificar claramente los requisitos de usuario y sirve como mecanismo para identificar los requisitos del software.
- Este modelo es útil cuando el cliente conoce los objetivos generales para el software a desarrollar, pero no identifica los requisitos detallados de entrada, procesamiento o salida.
- Ofrece un mayor enfoque cuando el responsable del desarrollo del software está inseguro de la eficacia de un algoritmo, de la adaptabilidad de un sistema operativo o de la forma que debiera tomar la interacción humano-máquina.
- Es posible reutilizar el código.

Desventajas

No obstante, el modelo de prototipos presenta, también, una serie de desventajas a tener en cuenta:

- Crea la tentación de usar el prototipo como primera versión del software al entrar en funcionamiento, bien por presión del cliente o bien por comodidad del desarrollador, cosa no recomendable.
- Esto implica que no se consideren aspectos de calidad y mantenimiento, o que la tecnología utilizada (por ejemplo, un lenguaje de programación determinado que proporcione a priori un desarrollo más rápido o eficiente) no sea la más apropiada.

Conclusiones

A pesar de que sea posible que surjan problemas, la construcción de prototipos puede resultar un paradigma efectivo en la ingeniería del software. La clave es definir las reglas del juego desde el principio, es decir, el cliente y el desarrollador deben ponerse de acuerdo en los siguientes puntos:

- El prototipo construido debe servir como mecanismo para la definición y refinamiento de requisitos.
- El prototipo debe ser descartado (al menos en parte).

- Después debe desarrollarse el software real con un enfoque hacia la calidad.

Cronograma

En este apartado se desglosarán las etapas de desarrollo que se han planeado para completar el proyecto. Como acabamos de ver en la metodología aplicada, estas etapas corresponden en gran medida a las especificadas el modelo de prototipos.

Fases	Duración estimada (horas)	Tareas
Análisis	30	Estudio de la documentación disponible
		Análisis de los requisitos de usuario
		Análisis de los requisitos de software
Diseño	50	Diseño de la arquitectura
		Diseño de la interfaz de usuario
Implementación	185	Implementación de los servicios planeados
		Implantación en el sistema
Validación	15	Pruebas de funcionamiento
		Corrección de errores
Documentación	20	Escritura de la memoria
		Escritura de la presentación
Total	300	

Tabla 2.

En la Tabla 2 podemos ver todas las fases con sus sendas estimaciones temporales. Como cabía esperar, la fase de implementación es la más larga. Hemos hecho una estimación bastante egoísta para la primera fase (análisis), dado que no es la primera vez que utilizamos las herramientas y tecnologías software más importantes descritas en el apartado de recursos. No obstante, empleamos parte de esa asignación para poder aprender sobre nuevos recursos que se hayan podido añadir recientemente. Por último, en la Figura 24 podemos observar la representación gráfica a modo de diagrama de sectores de dicha estimación temporal.

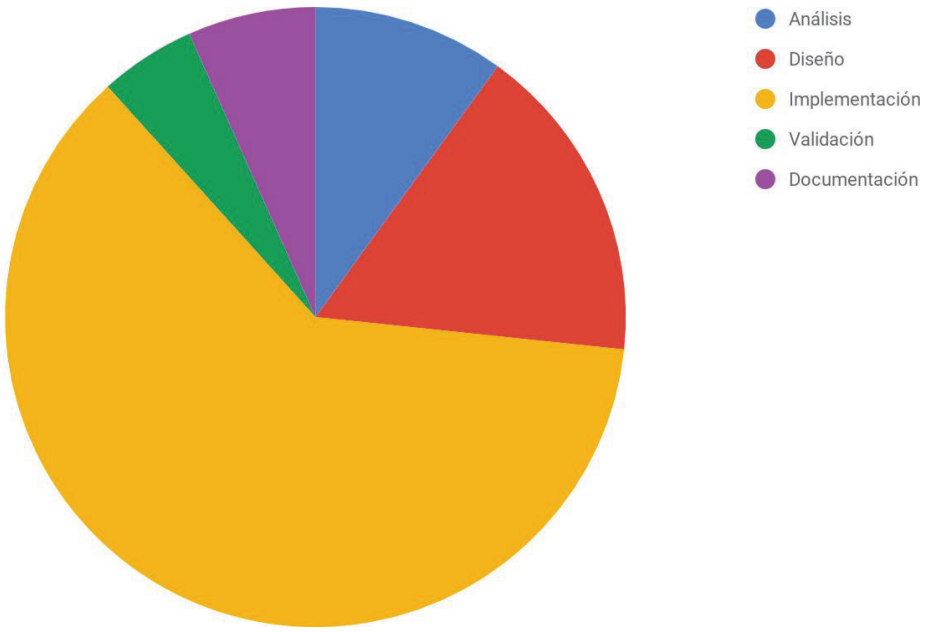


Figura 24.

Análisis

Este apartado describe la fase de análisis del desarrollo del proyecto. Hablaremos primeramente de los requisitos de usuario, para luego continuar con los requisitos del sistema o también denominados requisitos del software. En el primero haremos una descripción pormenorizada del dominio del problema, mientras que en el segundo relacionaremos el ámbito de este con aspectos programáticos. Es decir, una vez que conozcamos los requisitos del usuario, será necesario saber qué necesitará el sistema para la puesta en marcha de la consiguiente implementación del proyecto.

Antes de comenzar a comentar los diversos requisitos que se han encontrado, podemos proceder a explicar los datos de los que vamos a hablar a lo largo de no solo el análisis, sino de aquí en adelante.

Tomando como referencia la tabla de datos presente en la página de la Base de Datos de Recursos sobre Corpus[7] y que tiene en cuenta los datos alojados de la base de datos ECEG, podemos destacar los siguientes campos (así como una pequeña explicación de cada uno) separados en tres grupos principales:

Gramáticas

- Título » Título completo de la gramática así como de sus detalles de impresión.
- Año » Año de publicación de la primera edición que contiene la gramática o de la edición más temprana.
- Edición » primera edición que contiene la gramática o de la edición más temprana.
- Ediciones » todas las ediciones que se han citado en la literatura (incluso de las copias que no se han podido localizar).
- Lugar de impresión » País, condado y ciudad de impresión de la gramática
- Personas que lo imprimieron » Nombre de las personas que estuvieron a cargo de la impresión de la gramática.
- Vendedor » Nombres de las personas que estuvieron a cargo de la venta al público de la gramática.
- Precio » Precio de la gramática a la venta.
- Descripción física » Descripción física de la edición de la gramática.
- Tipo de obra » Obra en la que se incluye la gramática. Puede ser una gramática por sí sola o bien ser una sección dentro de una obra de mayor envergadura.
- División de 'gramática' » Qué partes de la gramática contiene la gramática en particular (morfología, sintaxis, etc)
- Contenido subsidiario » Contenidos extra que no son de la gramática pero complementan el contenido gramatical de la obra haciendo el producto.
- Audiencia objetivo
    - Edad de la audiencia del libro.

[7] Información sustraída y traducida de la siguiente web bajo la tabla de "*Parameters & coding*"http://www.helsinki.fi/varieng/CoRD/corpora/ECEG/basic.html

- Género de la audiencia del libro.
- Tipo de instrucción del libro.
- Finalidad del libro.

Autores

- Nombre » nombre completo y apellidos en el formato { apellidos } , { nombre }.
- Género » Género del autor (puede ser anónimo).
- Lugar de nacimiento » País, condado y ciudad de nacimiento.
- Ocupación » Categoría en la que trabaja el autor (ej Educación, política, religión, etc). Cada categoría tiene, a su vez, un rango de trabajos asociado.
- Detalles biográficos » Edad, lugar de residencia, relaciones, otros trabajos que haya podido hacer, etc.

Referencias

- Bibliotecas en posesión » Biblioteca que conserva la copia de la gramática.
- Referencias » Literatura desde donde se han extraído las fuentes primarias como datos sobre las fuentes y sus autores.
- Comentarios » Observaciones y correcciones de la gramática hecha desde la literatura o de la propia investigación de la base de datos ECEG.

Requisitos de usuario

Identificación de actores

- **Usuario no registrado**: en el caso de esta aplicación, actualmente podemos destacar solo un modelo de actor que hará uso de los procesos del sistema que se ha desarrollado y que interactuará con las interfaces diseñadas para el programa. Este tipo de usuario se trata de un usuario que no tiene ningún tipo de credencial y que accede a la página mediante un enlace web. Tiene acceso a las funciones de búsqueda de la aplicación y a la descarga de datos. Esto es así debido a las condiciones que el cliente de la aplicación dispuso.

Identificación de procesos

- **Hojear**: proceso por el cual un usuario navega entre los registros resultantes de haber llevado a cabo una búsqueda y seleccionado una gramática o bien desde el listado inicial de gramáticas que se presenta en la aplicación.
- **Buscar**: proceso por el cual un usuario aplica un filtro de búsqueda sobre la lista de resultados para consultar solo aquellas gramáticas que cumplan con sus requisitos.
- **Descargar registros CSV/JSON**: proceso por el cual un usuario descarga los registros de una búsqueda en formato CSV o JSON no sin antes seleccionar las propiedades que está interesado en mantener.
- **Exportar listado de datos/detalles de gramática como PDF**: proceso por el cual un usuario vuelca los resultados de una búsqueda en formato PDF para su futura lectura. Por otro lado, el mismo proceso se lleva a cabo a la hora de tener que volcar en un fichero PDF los detalles de una gramática.

Diagramas UML de las actividades

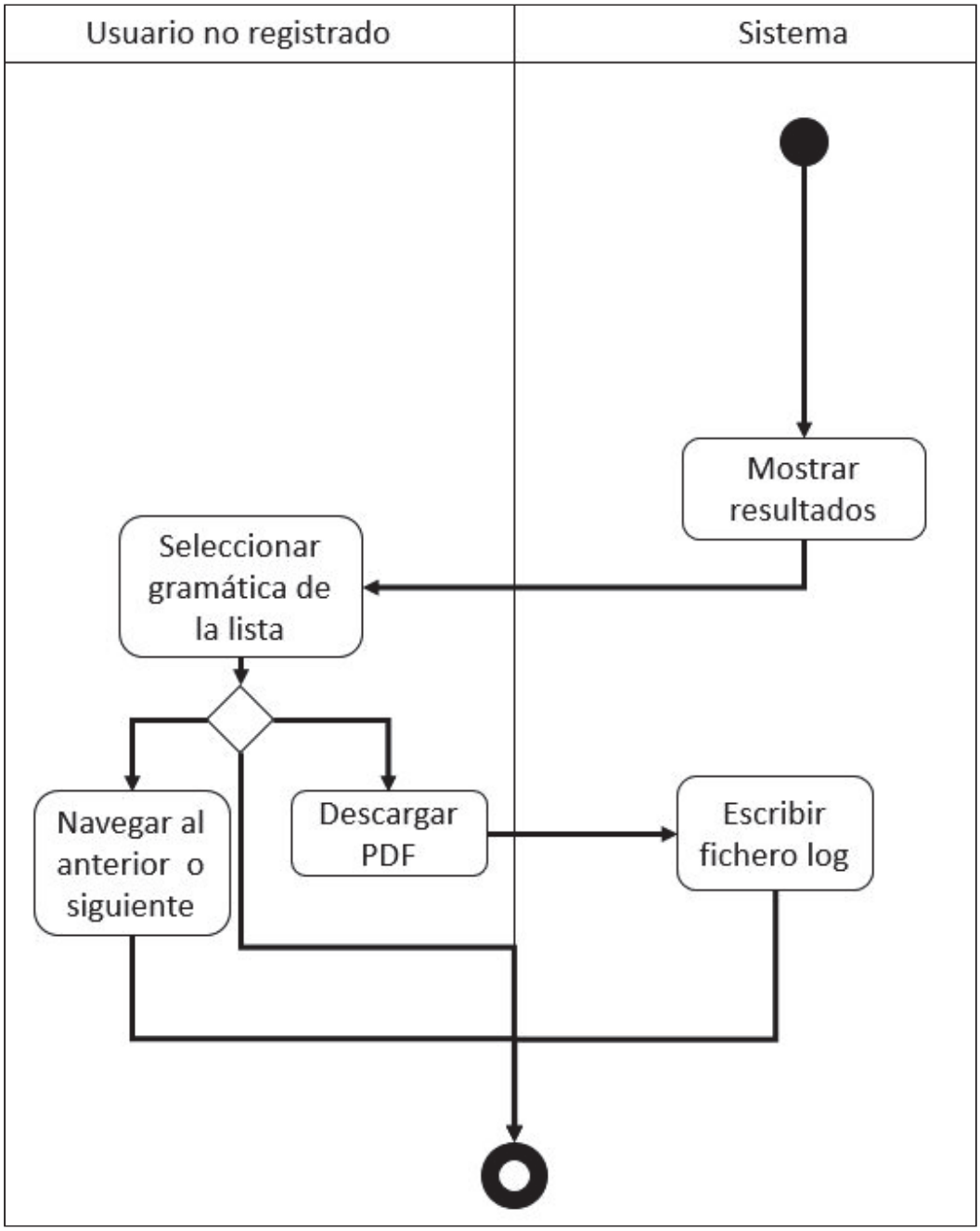


Figura 25.

En el diagrama UML superior podemos observar el proceso *hojear*. Este proceso permite al usuario acceder a los detalles de una gramática que se haya mostrado anteriormente en un listado del cual poder seleccionarla. Cuando se accede a cada uno

de los resultados, el usuario podrá avanzar en el listado anterior desde la gramática seleccionada si fuera el caso o bien poder descargar los detalles en formato PDF, volcando esta acción en un fichero log para el seguimiento del uso de la aplicación.

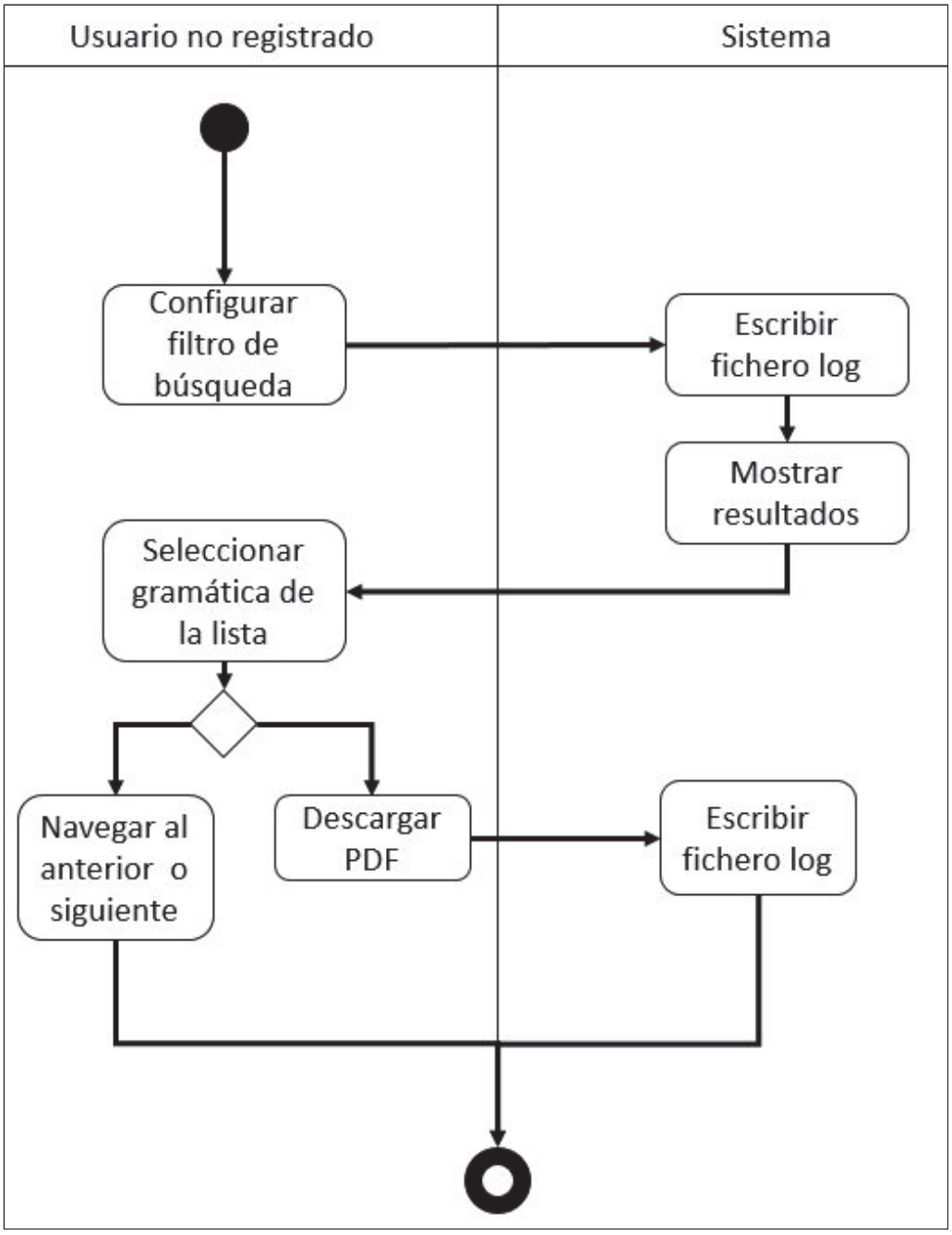


Figura 26.

El proceso *buscar*, representado en la figura 26 permite realizar búsquedas aplicando un filtro de búsqueda haciendo uso de los parámetros ofertados al usuario.

Una vez se configure acorde a los requerimientos personales de cada visitante, se muestran los registros (gramáticas) que cumplen con las especificaciones requeridas. Esta acción se lleva a cabo tras insertar una nueva entrada en el fichero de log con los parámetros introducidos por el usuario.

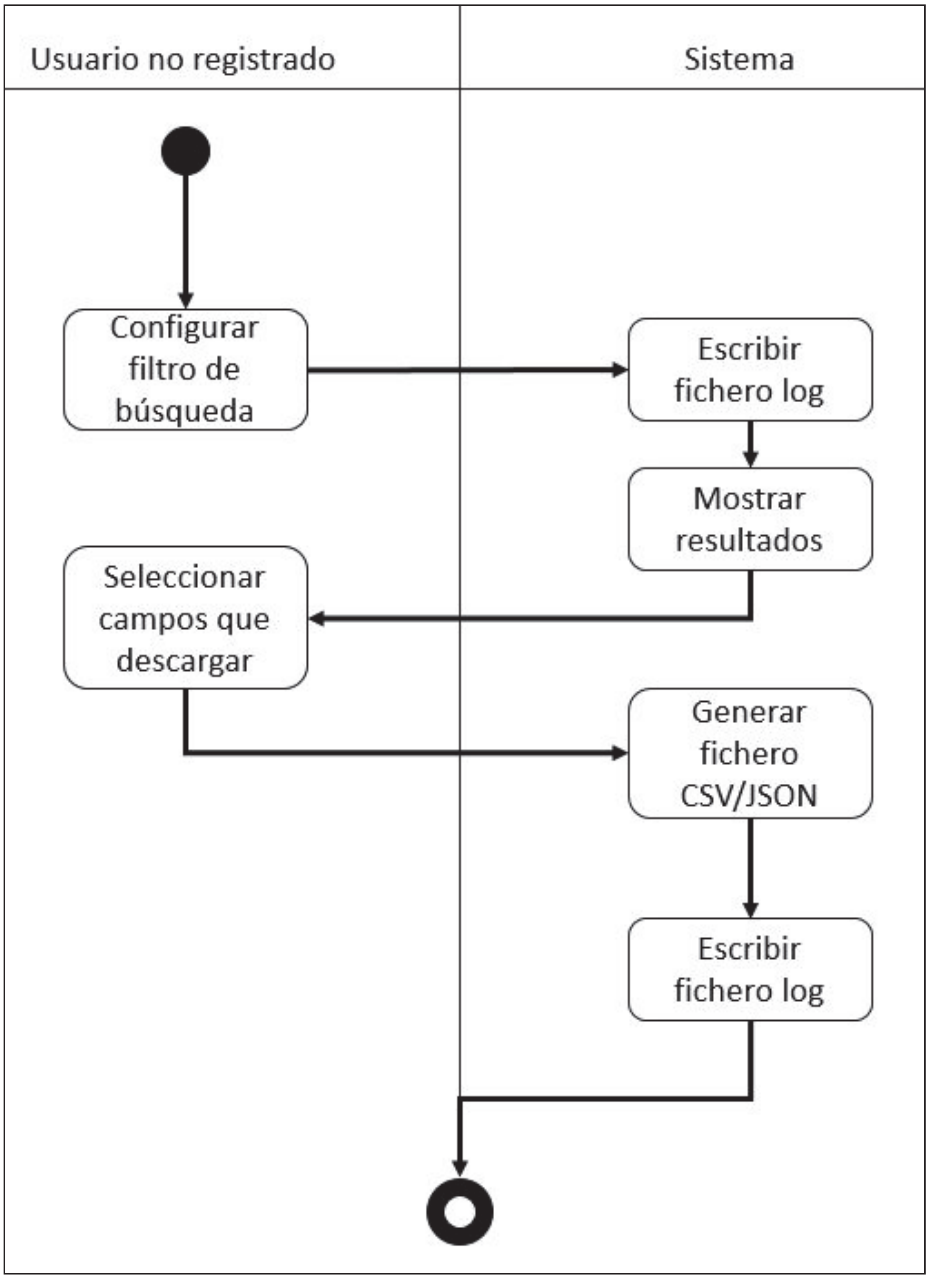


Figura 27.

En el siguiente diagrama UML podemos observar el proceso de *descargar CSV/JSON* mediante el cual un usuario podrá descargar de la lista de resultados de una

búsqueda que haga los campos que crea convenientes. La generación de estos ficheros en el formato elegido por el usuario de entre dos disponibles es volcada luego en un fichero log para mantener un seguimiento del uso de la aplicación.

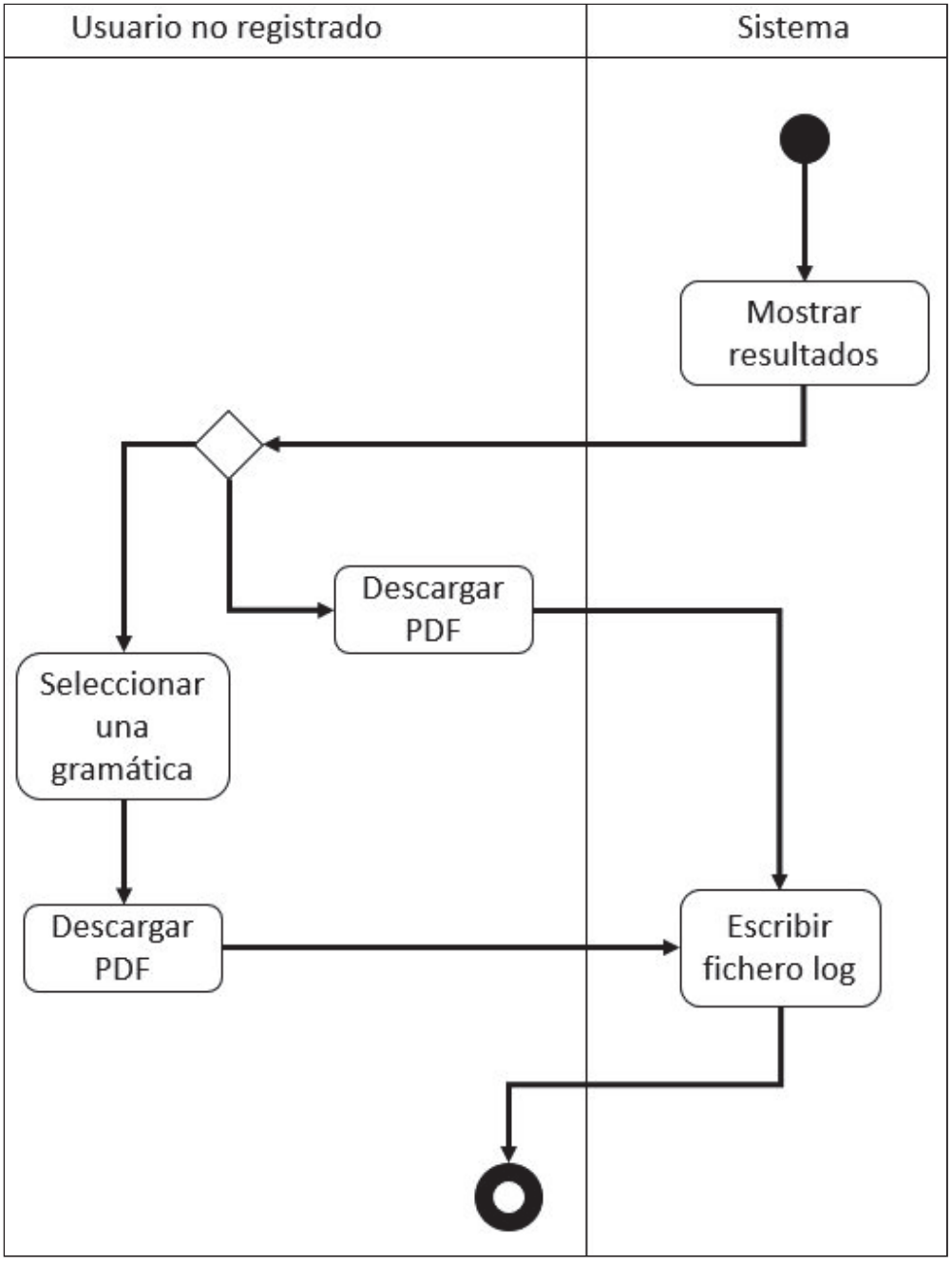


Figura 28.

Por último podemos encontrar el diagrama UML correspondiente al proceso *descargar PDF*. Los archivos PDF son algunos de los más utilizados hoy en día, así que se le ha dado la posibilidad al usuario de poder volcar los resultados de una búsqueda

que haya hecho o los resultados iniciales en uno de estos ficheros. En el caso de estar revisando los detalles de una de las gramáticas, el usuario también tiene la posibilidad de volcar la información en uno de estos. Estas acciones son llevadas a cabo para después volcar cierta información de seguimiento en un fichero de log y así aumentar las posibilidades y la calidad de la información que se puede analizar de la aplicación.

Requisitos del sistema

Modelo relacional de la base de datos

Para poder realizar el desarrollo de la aplicación, partimos de la base de datos Access que nos ha proporcionado el cliente:

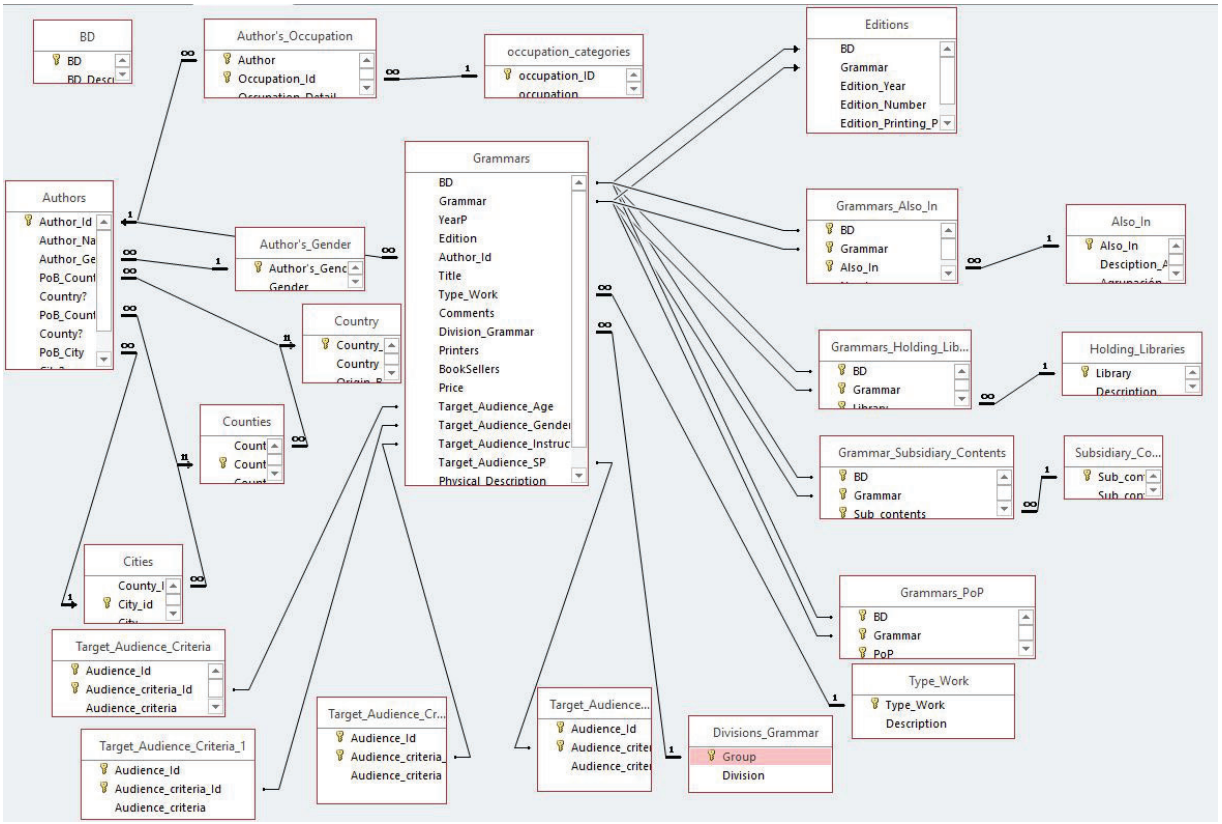


Figura 29.

De la que podemos destacar lo siguiente:

- Una gramática tiene solamente un autor.
- Una gramática puede tener más de una edición.
- Una gramática puede estar alojada en varias bibliotecas debido a que puede tener varias ediciones.
- Una gramática puede tener varias referencias bibliográficas.

- Cada gramática tiene un lugar de publicación.
- Una gramática pertenece solamente a un tipo de obra.
- Una gramática puede tener varios contenidos subsidiarios.
- Una gramática tiene una sola división de gramática principal.
- Una gramática tiene solamente una serie de audiencias objetivo concretas.
- Un autor tiene solo un género
- Un autor puede tener varias gramáticas asociadas
- Un autor puede haber tenido trabajos en varias ocupaciones
- Un autor tiene solamente un país, condado y ciudad de nacimiento

Cabe destacar que el hecho de partir de una base de datos ya conformada tiene tanto sus puntos positivos como negativos. Esto quedará comentado más adelante, en la sección de las conclusiones del trabajo, puesto que para el buen funcionamiento del programa se debieron de llevar a cabo varios arreglos en la misma por errores de funcionamiento en el fichero y aumentar la eficiencia del programa.

Especificaciones de los casos de uso

A continuación podemos ver las plantillas de operaciones para las acciones relacionadas con los usuarios:

Nombre	Hojear
Descripción	El usuario navega hacia delante o hacia atrás entre los detalles de los registros de la última búsqueda hecha.
Parámetros	Identificador del registro actual y lista de gramáticas.
Precondiciones	-
Ejecución	Se procesa no solo el siguiente registro sino que en el caso de haber espacio, también se hace con los siguientes 5.
Postcondiciones	-
Excepciones	Se intenta navegar más allá del último registro o por debajo del primero.
Roles	Usuario no registrado

Tabla 3.

Nombre	Buscar
Descripción	Se muestran las gramáticas que cumplan con el filtro de búsqueda que el usuario ha configurado.
Parámetros	Cada uno de los campos que se ha configurado para la búsqueda así como los valores que se les ha dado. Esto es así tanto para una búsqueda simple (búsqueda por año de publicación, primera edición conocida y/o título) como para la búsqueda avanzada (resto de parámetros de cada una de las pestañas que se le presentan al usuario).
Precondiciones	-
Ejecución	Se genera un predicado en tiempo de ejecución acorde a la configuración que el usuario ha introducido en la aplicación mediante la interfaz, filtrando las gramáticas que estén en el programa que cumplen con las condiciones establecidas. En el caso de que no se aplique ningún filtro o se limpie el formulario de búsqueda se mostrarán todas las gramáticas.
Postcondiciones	Se muestra una lista con las gramáticas que cumplen con las condiciones del usuario.
Excepciones	Error en la consulta de la lista de gramáticas
Roles	Usuario no registrado

Tabla 4.

Nombre	Descargar CSV/JSON
Descripción	El sistema genera un fichero CSV o JSON de las propiedades seleccionadas de las gramáticas.
Parámetros	Propiedades que el usuario quiere tener en cuenta y la opción de JSON o CSV
Precondiciones	Mínimo de un campo de las propiedades tiene que ser seleccionado.
Ejecución	El sistema crea una lista de objetos mutables en tiempo de ejecución a los que se le van añadiendo las propiedades de cada una de las gramáticas que queremos descargar. Posteriormente, el objeto se descarga desde el navegador.

Postcondiciones	El objeto queda guardado en el sistema del usuario con el formato que ha decidido.
Excepciones	No hay ninguna gramática en la lista.
Roles	Usuario no registrado

Tabla 5.

Nombre	Descargar PDF
Descripción	El usuario exporta la lista de resultados de las gramáticas en formato PDF o bien genera un PDF con los detalles de una en concreto para su lectura personal.
Parámetros	Lista de gramáticas a exportar o una gramática en concreto para exportar sus detalles y un indicador sobre si quiere la lista o los detalles.
Precondiciones	En el caso de exportar los detalles de una gramática primero hay que seleccionarla.
Ejecución	El sistema, gracias a los parámetros de entrada, genera un PDF con un formato de lista o un fichero preparado para visualizar las propiedades de la gramática
Postcondiciones	-
Excepciones	No hay permisos para escritura en el directorio de descarga
Roles	Usuario no registrado

Tabla 6.

Diseño

En este apartado trataremos sobre la representación del diseño arquitectónico de los datos y componentes que estarán disponibles en el programa así como sus propiedades y las relaciones entre ellos. Es el paso intermedio entre el análisis del proyecto y la especificación de requisitos y la implementación en sí del programa. Por así decirlo, es como un plano que un arquitecto sigue para poder construir una casa.

Empezamos a explicar, en primer lugar, el diseño de los datos y el diagrama de las clases que serán utilizadas en el software. Aquí estarán descritas solo aquellas clases que vayamos a virtualizar en el sistema. Asimismo, mostraremos el diagrama relacional de la base de datos y explicaremos su estructura debido a que es un elemento que ha sido proporcionado por un tercero. Podemos seguir explicando el diseño de la arquitectura software que se ha tomado para la implementación del mismo y así poder mostrar las relaciones entre las estructuras del proyecto. Después hablaremos sobre el diseño de las diferentes interfaces de usuario que se han ideado para el proyecto y cómo el software se comunica con el usuario. Por último, trataremos el diseño a nivel de componentes y su distribución lógica en el proyecto.

Diseño de datos

Modelos

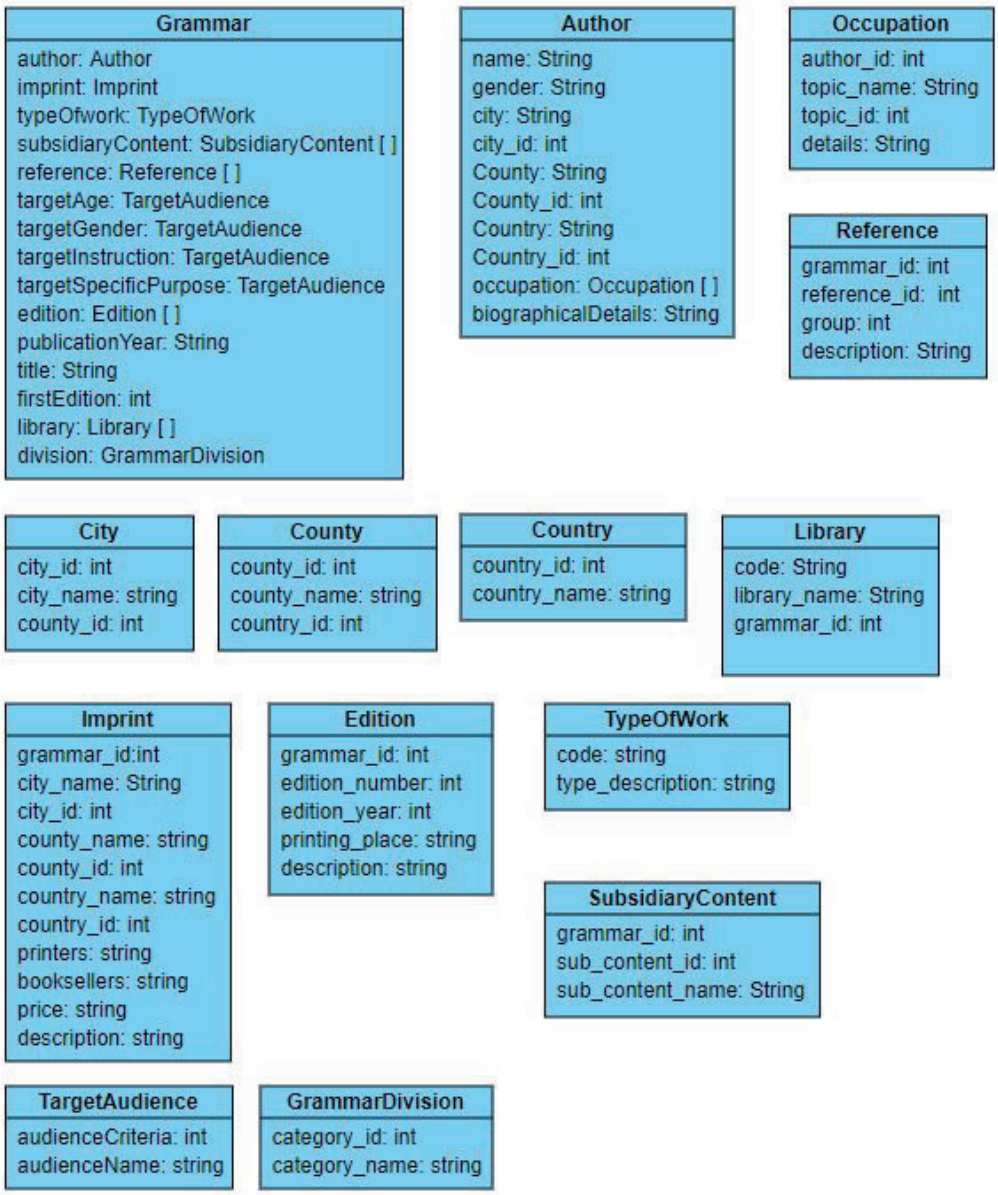


Figura 30.

Cuando hablamos de virtualización de elementos reales en un programa estamos hablando de modelos. Son representaciones en código de la realidad que necesitamos gestionar. En este caso podemos encontrar una gran variedad de modelos

que se necesitan virtualizar en el entorno para hacer efectivo el producto software. Por así decirlo, un modelo es una colección de propiedades y objetos que representan algo de la vida real en un programa.

Con esta pequeña explicación, podemos empezar a explicar cada una de nuestras clases de modelo por separado.

Clase Grammar

La clase Grammar es la clase principal de nuestro programa. Es la encargada de representar cada gramática de la base de datos en la interfaz y es un contenedor para todos los datos que van relacionadas con la misma, sus atributos son:

- **author**: es el autor de la gramática. Un objeto con todas las propiedades del autor que la escribió
- **imprint**: es un objeto con la información que se tiene sobre la impresión de la gramática con la que se trabaje.
- **typeOfWork**: está relacionado con el tipo de obra que se ha escrito (libro de gramática, diccionario, etc)
- **division**: se refiere a qué parte de la gramática está enfocada esta obra (sintaxis, ortografía, morfología, etc)
- **subsidiaryContent**: es una colección de temas que la obra trata aparte de los temas de su propia división. Por ejemplo, en el caso de un diccionario podíamos encontrar secciones sobre ortografía, pero también podíamos ver que habían nociones de léxico, mapas, etc.
- **Reference**:
- **targetAge, gender, instruction, specificPurpose**: se refieren a las diferentes propiedades de las audiencias a las que la obra iba dirigida.
- **editions**: información sobre las numerosas ediciones que hay de la misma gramática, así como su año de publicación, descripción de la misma y demás información.
- **firstEdition**: información sobre la primera edición conocida de la gramática. Tiene un tratamiento especial diferente al del campo *editions* debido a que no siempre podemos saber a ciencia cierta si fue la primera edición o si simplemente se cree que lo fue.
- **publicationYear**: año de publicación de la primera edición de la gramática gramática.
- **title**: título de la gramática.
- **Library**: contiene una colección de las bibliotecas que contienen algún ejemplar de la gramática con la que se está trabajando.

- **References**: literatura desde donde se han extraído las fuentes primarias como datos sobre las fuentes y sus autores.

Clase Author

Esta clase contiene información sobre el autor de una gramática. Contiene no solo información biográfica sino también otros aspectos más personales. Para poder representar lo que era interesante para el proyecto, se eligieron los siguientes campos:

- **Name**: nombre del autor
- **Gender**: género/sexo del autor
- **City y city_id**: nombre y código dentro de la base de datos de la ciudad de nacimiento
- **County y county_id**: nombre y código dentro de la base de datos del condado de nacimiento.
- **Country y country_id**: nombre y código dentro de la base de datos del país de nacimiento
- **Occupation**: trabajos que ha desarrollado el autor
- **biographicalDetails**: colección de datos biográficos del autor

Clase Occupation

Dentro de la clase Author podemos encontrar una referencia a los diversos trabajos que ha tenido. Debido a que podían tener diversos valores y dentro de cada elemento teníamos el trabajo principal y detalles sobre el mismo, se decidió encapsularlos junto a los códigos de la base de datos en una clase aparte.

- **Author_id**: autor del que trata el trabajo
- **Topic_name**: ámbito en el que se desarrolla el trabajo (educación, política, etc)
- **Topic_id**: código del ámbito en el que se desarrolla el trabajo dentro de la base de datos
- **Details**: especificaciones de las labores que desempeñaba el autor/a

Clase Imprint

Cuando una gramática se descubre y se anota en la base de datos, se hace también con todos los datos que se hayan podido descubrir en cuanto a la impresión de esta: lugares, precio, … Creemos que es una colección de datos importante debido a que podemos, gracias a esto, conocer qué lugar producía más libros, el precio medio de un tipo en concreto, etc.

- **Grammar_id:** gramática relacionada con la impresión
- **City y city_id**: nombre y código dentro de la base de datos de la ciudad de impresión
- **County y county_id**: nombre y código dentro de la base de datos del condado de impresión.
- **Country y country_id**: nombre y código dentro de la base de datos del país de impresión
- **Printers**: personas que imprimieron la edición de la gramática
- **Booksellers**: personas que pudieron vender la gramática
- **Price**: precio de la gramática
- **Description**: descripción física de la gramática.

Clase Edition

En esta clase hemos encapsulado los datos que tienen que ver con el resto de ediciones de una gramática sin contar con la primera.

- **Grammar_id**: gramática relacionada con la edición
- **Edition_number**: número de la edición en concreto
- **Edition_year**: año de impresión de la edición
- **Printing_place**: lugar de impresión de la edición
- **Description**: notas sobre la descripción física de la edición impresa en el caso de que hubiera

Clase GrammarDivision

Cuando hablamos de una gramática, tenemos que destacar de manera necesaria el tema del que trata la misma. En este caso hablamos de una división de la gramática.

- **Category_id**: código de la división de la gramática dentro de la base de datos
- **Category_name**: nombre de la división de la gramática

Clase SubsidiaryContent

Pese a que una gramática pertenece a una división, tenemos que tener en cuenta que las obras pueden contener diferentes secciones. Cada sección puede tratar de un tema diferente como es la morfología, lexicología, prosodia, etc.

- **Grammar_id**: gramática relacionada con el contenido subsidiario
- **Sub_content_id**: código dentro de la base de datos que nos ordena los contenidos subsidiarios
- **Sub_content_name**: nombre del tipo de contenido subsidiario

Clase Reference

Para poder generar los registros disponibles en la base de datos ECEG, se ha tenido que acudir a una gran cantidad de obras y lugares. Estas referencias son literaturas desde donde se han extraído las fuentes primarias como datos sobre las fuentes y sus autores.

- **Grammar_id**: gramática de la que extraemos las referencias
- **Reference_id**: código en la base de datos de la referencia
- **Group**: campo de uso interno. Es usado por los creadores de la base de datos para comprobar que se habían consultado todas las fuentes que tenían en mente.
- **Description**: información sobre la referencia. Normalmente se refiere al documento u obra al que se ha acudido para obtenerla

Clase Library

Esta clase modela las bibliotecas que tienen posesión de una copia de la gramática con la que estamos trabajando.

- **Code**: código de la biblioteca o universidad a la que se ha acudido
- **Library_name**: nombre completo de la biblioteca
- **Grammar_id**: referencia a la gramática con la que se está trabajando

Clase TargetAudience

Cada obra está dirigida a un grupo de lectores específico. Esta clase se encarga de aglutinar las características de ese grupo.

- **audienceCriteria**: código del tipo de audiencia de la gramática de la que se está consultando (edad, género, etc)
- **audienceName**: nombre de la audiencia del tipo de gramática (jóvenes, adultos, público femenino, etc)

Clase TypeOfWork

Esta clase se encarga de encapsular las características de la obra de gramática con la que estamos trabajando. Aquí podemos encontrar diversos tipos de escritos como por ejemplo: diccionarios, enciclopedias, libros de pronunciación, etc.

- **Code**: Código de la abreviatura del tipo de obra que se ha escrito
- **Type_description**: nombre completo del tipo de obra de la que se trata

Clase City

Esta clase encapsula las características que podemos recabar de la base de datos que tienen que ver con una ciudad.

- **City_id**: código de la base de datos relacionado con la ciudad
- **City_name**: nombre de la ciudad
- **County_id**: código del condado al que pertenece la ciudad

Clase County

Esta clase encapsula las características que podemos recabar de la base de datos que tienen que ver con un condado.

- **County_id**: código de la base de datos relacionado con el condado
- **County_name**: nombre del condado

- **Country_id**: país al que pertenece el condado

Clase Country

Esta clase encapsula las características que podemos recabar de la base de datos que tienen que ver con un país.

- **Country_id**: código de la base de datos relacionado con el país
- **Country_name**: nombre del país

Relación de los modelos

En este apartado podemos observar la manera en la que se relacionan los modelos que hemos representado anteriormente entre sí. Algunos de los modelos presentados serán usados no en esta figura, sino en las clases que vienen a continuación.

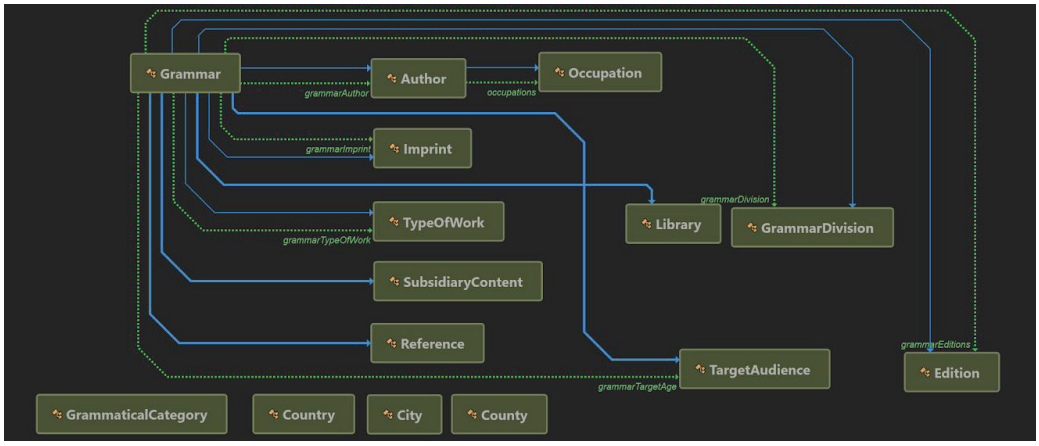


Figura 31.

Cabe comentar que algunos de los campos de las clases no se han utilizado todavía, sino que se han importado desde la base de datos para mantener la misma estructura en los modelos y así poder ampliar su uso más adelante en el caso de que se requiera.

Conexión con la base de datos

Para poder llevar a cabo las búsquedas de los datos, es necesario que estén disponibles dentro de programa. Por ello, un módulo que se encargara no solo de la conexión con la base de datos sino que convirtiera los registros de la misma en instancias de las clases que hemos explicado anteriormente era necesario. Por ello, dentro de este módulo hemos ubicado dos clases: una de ellas se encarga de conectar

con la base de datos y obtener los resultados en una forma que sea correcta y útil para el programa y la otra es la encargada de hacerlos disponibles para los usuarios que quieran hacer uso del servicio que se les brinda.

Cabe destacar que, para una mayor modularidad, se ha podido abstraer todo el tema que tiene que ver con el tratamiento de los datos y así dedicar las páginas como tal solo a mostrar el contenido de los datos y a poder comunicar las acciones que el usuario quiere realizar con el módulo correspondiente.

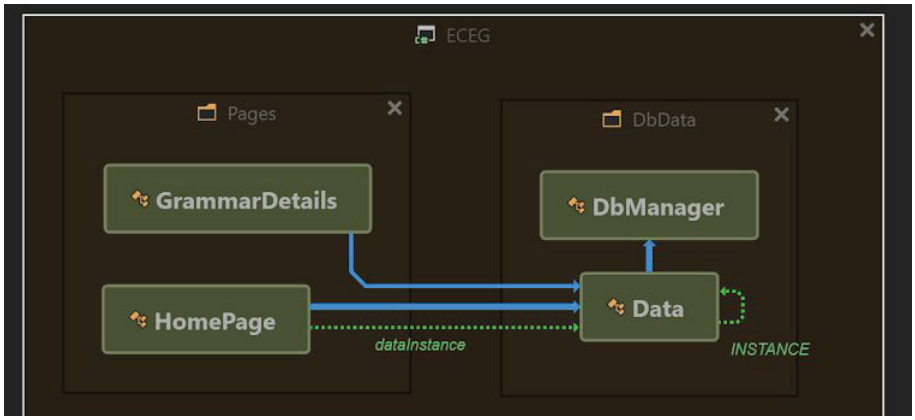


Figura 32.

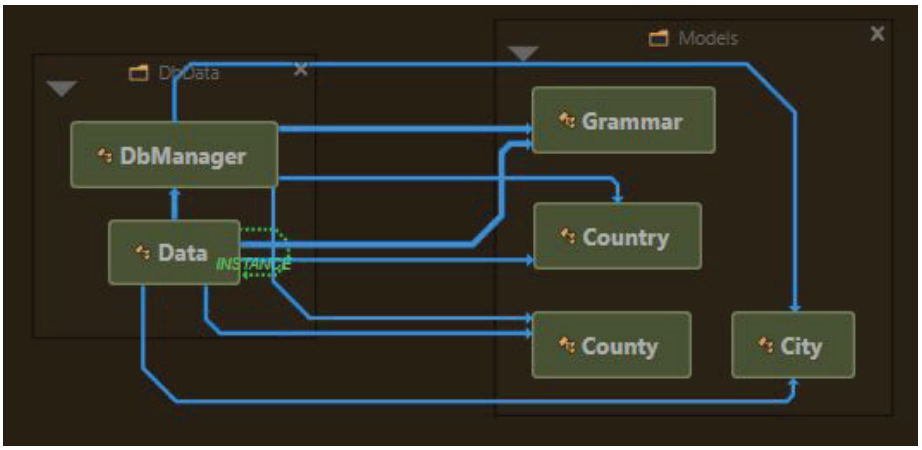


Figura 33.

Clase DBManager

Dentro del programa, una de las clases más importantes que podemos encontrar es de la que estamos hablando ahora mismo. Es la encargada de abrir las conexiones pertinentes con la base de datos, de traer los datos y de hacer una primera fase de tratamiento para que la siguiente clase del mismo módulo acabe por hacer disponibles los modelos de las gramáticas al usuario.

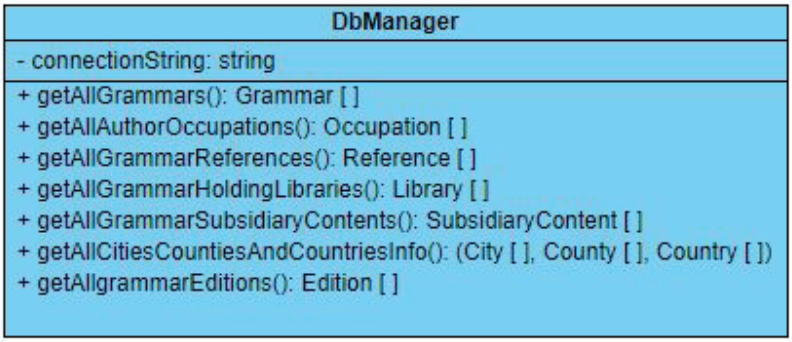


Figura 34.

Esta clase solamente tiene un atributo y es aquel al que vamos a hacer mención ahora mismo: connectionString. Es una cadena de texto en la que especificamos el tipo de conexión que vamos a hacer a la base de datos, la versión de la conexión y, por último aunque no menos importante, el nombre de la base de datos a la que nos vamos a conectar. Sin este atributo no seríamos capaces de poder haber hecho nada de la aplicación en sí. Por ello consideramos que este módulo es, junto con el módulo de los modelos, uno de los más importantes de la aplicación.

Para poder poblar nuestra aplicación con los datos que recogemos, se han creado varios métodos que llevan a cabo diferentes funciones dentro de la gestión de la base de datos. Aunque, antes de explicarlos brevemente, cabe mencionar que son métodos de lectura, manteniendo en todo momento la integridad de la base de datos sin verse comprometida.

Método	Tipo de retorno	Funcionalidad
getAllGrammars( )	Grammar [ ]	Accede a la base de datos y devuelve una colección de gramáticas con varios de los campos rellenos.
getAllAuthor Occupations( )	Occupation [ ]	Accede a la base de datos y devuelve una colección completa de los trabajos que se han llevado a cabo por todos los autores.
getAllGrammar References( )	Reference [ ]	Accede a la base de datos y devuelve una colección de

		todas las referencias guardadas en la base de datos.
getAllGrammar HoldingLibraries( )	Library [ ]	Accede a la base de datos y devuelve una colección de todas las bibliotecas a las que se ha acudido para poder obtener la información disponible en la base de datos.
getAllGrammar SubsidiaryContents( )	SubsidiaryContent [ ]	Accede a la base de datos y devuelve una colección con el contenido subsidiario de todas las gramáticas de la base de datos.
getAllCities Counties AndCountriesInfo( )	(City [ ], County [ ], Country [ ])	Accede a la base de datos y devuelve una tupla con tres colecciones que contienen la información de todas las ciudades, condados y países presentes en la base de datos.

Tabla 7.

Esta manera de operar se lleva a cabo no sin antes haber sopesado otras opciones. La complejidad de las relaciones de las tablas de la base de datos nos llevó a pensar que era más eficiente dividir los accesos a la base de datos y traer en cada acceso mucha información que tuviera que ver con todas las gramáticas en general que tener que acceder a la base de datos por cada gramática por separado.

Debido a la forma que la base de datos tiene, todos los campos de una gramática no pueden ser recogidos de una sola vez, por lo que por cada gramática harían falta 6 accesos diferentes, disparando el tiempo que tendríamos que esperar para tener todas las gramáticas disponibles en el programa. La manera que se ha llevado a cabo solamente requiere de seis accesos en total para obtener toda la información necesaria de todas las gramáticas de la base de datos, reduciendo el tiempo de espera significativamente.

Por otra parte, no solo hemos hecho uso de los métodos y funciones propias sino que hemos usado una librería que viene dentro de los paquetes system.Data y

system.Data.oledb. De entre clases del paquete que se han usado podemos destacar las de la siguiente tabla:

Clase	Funcionalidad
OleDbConnection	Proporciona una conexión usada para poder comunicarnos con la base de datos.
OleDbCommand	Nos permite ejecutar acciones sobre la fuente de datos, como consultas, inserciones, modificaciones (actualizaciones) o eliminación de datos relacionales.
OleDbDatareader	Permite procesar eficientemente una lista grande de resultados en un solo registro a la vez

Tabla 8.

Clase Data

Como puente entre la base de datos y el programa está, como hemos mencionado, el paquete con las clases relacionadas con la conexión a la base de datos y una clase que completa el proceso de transformación de los datos provenientes de la conexión de la base de datos para que el resto del programa pueda trabajar con estos de la manera más eficiente y correcta posible.

Esta clase Data se ha implementado teniendo en mente el patrón de programación Singleton de tal manera que la misma instancia de las gramáticas cargadas en el programa del servidor esté disponible para todos los usuarios que quieran hacer uso del servicio que se ha puesto en marcha. Este patrón de programación tiene sus razones de existir:

"En ingeniería de software, singleton o instancia única es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella."[8]

Debido a esta razón y teniendo esto en mente, podemos afirmar que este patrón de diseño era el idóneo para una clase de la que solamente se quiere tener una instancia que esté en el servidor y que los clientes puedan consultarla. Al ser un conjunto de valores que no son de escritura sino de solo lectura no habrá ningún tipo de corrupción en la integridad de los datos y la carga del servidor será mucho menor.

De haber aplicado en este mismo caso un patrón de diseño más tradicional, es decir, creando una instancia de los datos por cada cliente que accede al servicio, podemos observar en la siguiente gráfica que la carga en la memoria del servidor se dispara, aumentando considerablemente su uso y, por ello, reduciendo la eficiencia del sistema donde estuviera alojado el programa.

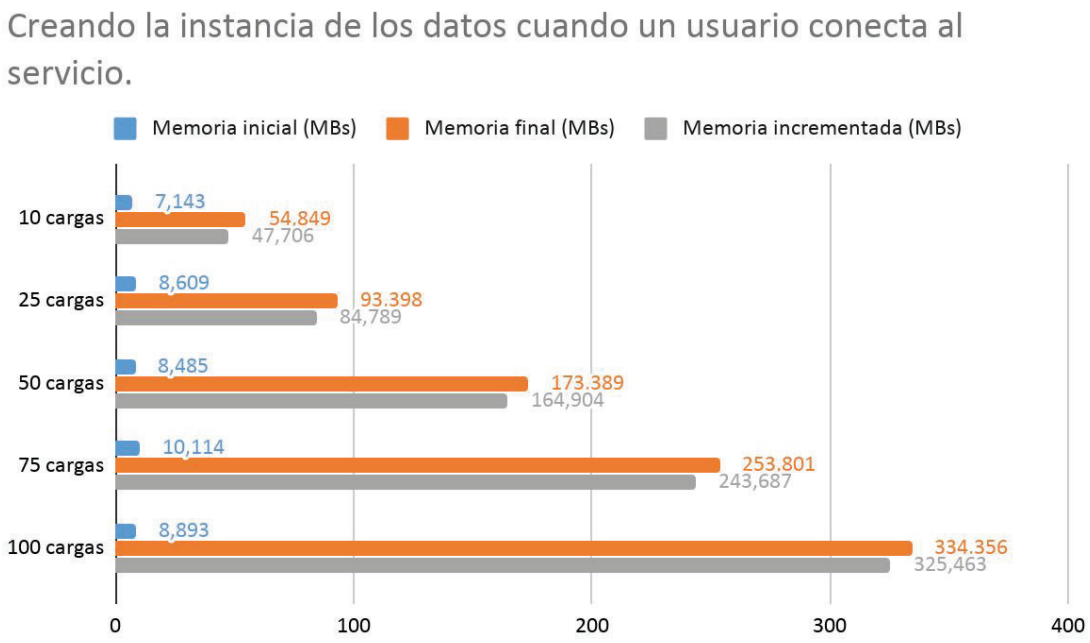


Figura 35.

Como podemos observar, ya en diez instanciaciones de los datos de las gramáticas el uso de la memoria roza unos 50 MBs, escalando a casi 330 MBs en el caso de las 100 instanciaciones de los datos. Esto podría escalar mucho más en el caso de superar las cargas supuestas en la gráfica.

[8] Definición extraída de https://es.wikipedia.org/wiki/Singleton

Por otra parte, mediante el uso de la clase como un singleton, podemos apreciar en la gráfica siguiente que el aumento de memoria es casi inapreciable y se mantiene igual durante las pruebas con el mismo número de cargas que en el caso anterior.

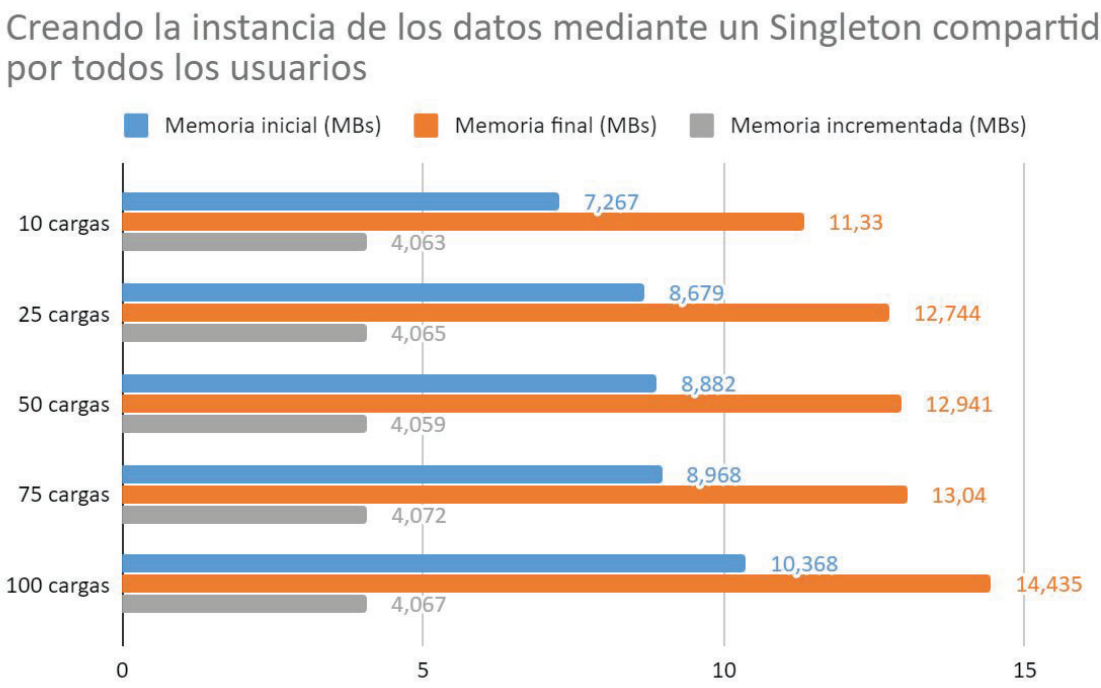


Figura 36.

Gracias a la gráfica mencionada, podemos observar como el incremento en la memoria del servidor se mantiene estable en todas las situaciones. Siempre alrededor de los 4 MBs que ocupa el objeto con la lista de las gramáticas y todo los objetos que dentro habitan.

Esta clase es aquella que las páginas usarían para poder acceder a los datos de las tablas, ya una vez terminado el formateo y el tratamiento de los datos. El mecanismo por el que la clase termina este tratamiento es llamando a los métodos de DbManager y colocando los valores que podrían hacer que la consulta SQL fuera desmesuradamente grande, evitando así, en gran medida, la complejidad en las consultas efectuadas debido a que no ha sido necesario el tener que anidar demasiados niveles de profundidad.

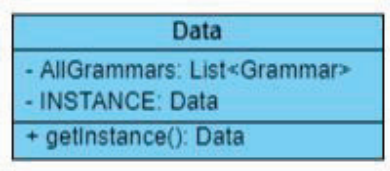


Figura 37.

Por razones del patrón de diseño, la clase solo posee un método: getInstance. Y es el encargado de darle a cada usuario la copia de todas las gramáticas de la base de datos en forma de lista de objetos. Como último comentario sobre esta clase podríamos decir que sus atributos son: por una parte el elemento imprescindible de una clase con un patrón singleton, su instancia y, por otra parte, el objeto que contiene todas las gramáticas que hemos podido extraer de la base de datos.

Clases de ayuda

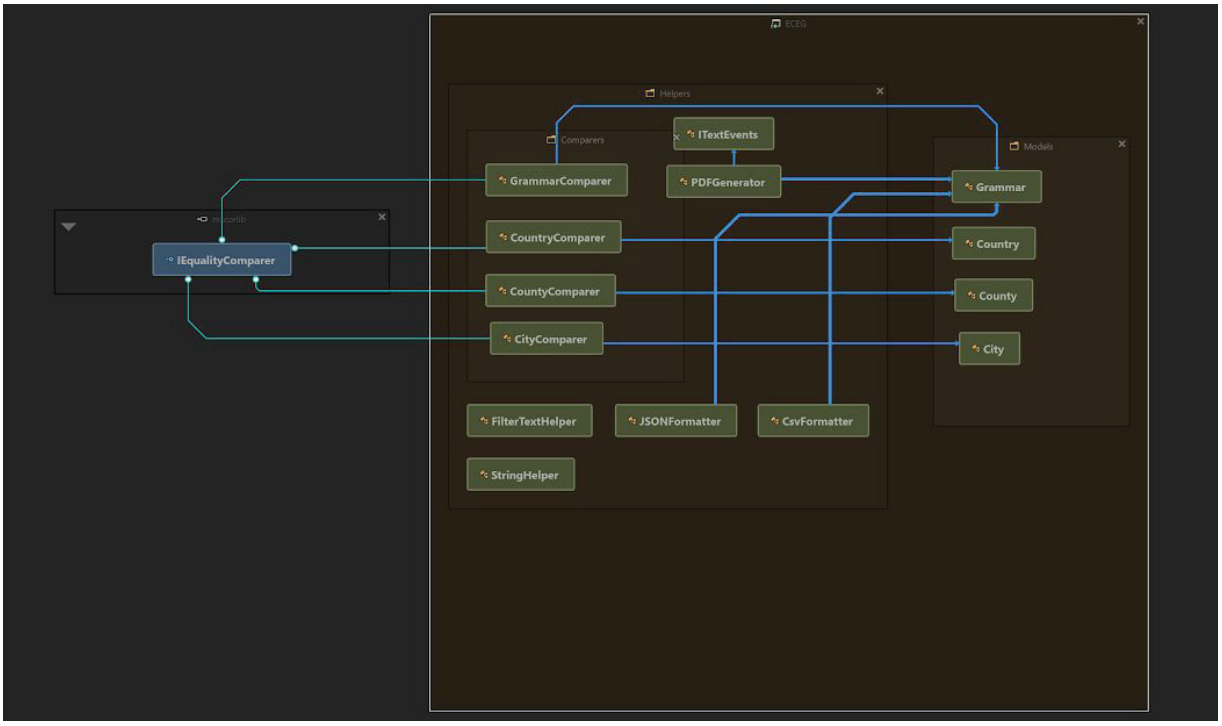


Figura 38.

Para mantener la modularidad del proyecto, se ha implementado un paquete que contiene diversas clases que se usan de tal forma que no solo añaden funcionalidad extra al producto sino que sirven de apoyo para poder extender las características de ciertas clases. Debido a su finalidad, se ha decidido que este módulo sea calificado com un módulo de ayuda o *helper*.

Dentro de este módulo podemos encontrar los diferentes comparadores personalizados que se han creado para poder discernir cuándo hemos añadido elementos a una lista de cuándo no, también podemos ver los generadores de archivos CSV y JSON así como la clase que se encarga de generar un fichero en formato PDF con los resultados de una búsqueda o los detalles de una sola gramática.

Clase JSONFormatter

Como comentamos anteriormente, el módulo de ayuda tiene una clase encargada de ampliar la funcionalidad del programa añadiendo la posibilidad de volcar los resultados de una búsqueda a un fichero JSON para así poder luego cualquier tratamiento que queramos de una manera eficiente, rápida y, sobre todo, cómoda dado que es un formato ampliamente utilizado y estandarizado en los programas informáticos.

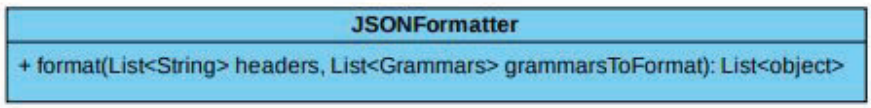


Figura 39.

Dentro de esta clase solamente podemos encontrar un método: aquel que nos realiza la generación de una lista de objetos que se transformarán en un JSON al finalizar la conversión. Este método recibe dos parámetros: headers y grammarsToFormat. El primer parámetro se refiere a los campos que queremos volcar en el JSON, ya que el usuario tiene la libertad de elegir lo que figurará en la estructura y qué no le hace falta que esté. Por otro lado, el segundo parámetro se refiere a los resultados de la última búsqueda que ha hecho y sobre los que se va a hacer la operación de formateo.

Esta clase es una ampliación del servicio inicial, debido a que anteriormente solamente se podían descargar los valores en formato CSV y su gestión por los sistemas no era la más adecuada. Siendo JSON un añadido muy atractivo para futuras herramientas que puedan beber de este servicio.

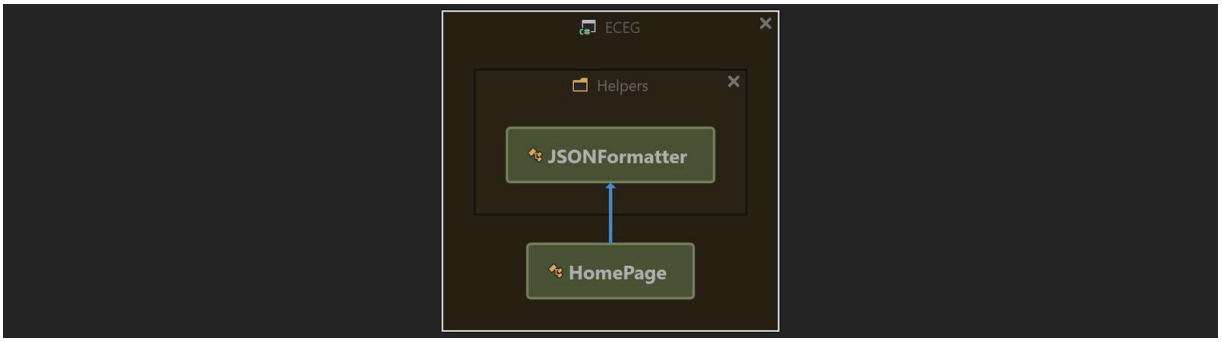


Figura 40.

Como podemos observar en la figura, la clase está implementada de tal manera que su acoplamiento es inexistente, creando una clase altamente modular y reutilizable para muchas otras posibles futuras tareas.

Clase CSVFormatter

Esta clase da al programa una funcionalidad existente en el anterior proyecto. El usuario puede descargar los campos que desee de los resultados de su última búsqueda y exportarlos a un fichero CSV y poder guardarlos donde lo requiera. Es una funcionalidad complementaria al generador de ficheros JSON y se optó por mantenerla por comodidad de los anteriores usuarios que seguirán haciendo uso del servicio.

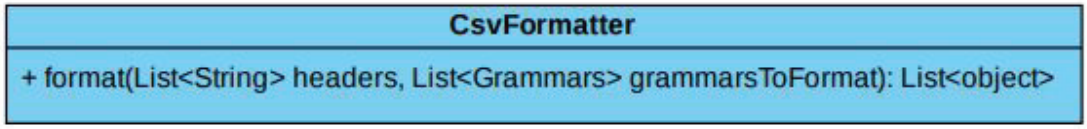


Figura 41.

Dentro de esta clase solamente podemos encontrar un método: aquel que nos realiza la generación de una lista de objetos que se transformarán en un CSV al finalizar la conversión. Este método recibe dos parámetros: headers y grammarsToFormat. El primer parámetro se refiere a los campos que queremos volcar en el CSV, ya que el usuario tiene la libertad de elegir lo que figurará en la estructura y qué no le hace falta que esté. Por otro lado, el segundo parámetro se refiere a los resultados de la última búsqueda que ha hecho y sobre los que se va a hacer la operación de formateo.

Para poder generar el fichero CSV a partir de la estructura de la clase de la que hemos hablado, ha hecho falta la ayuda de una librería de terceros altamente utilizada para estos trabajos: la librería CsvHelper[9].

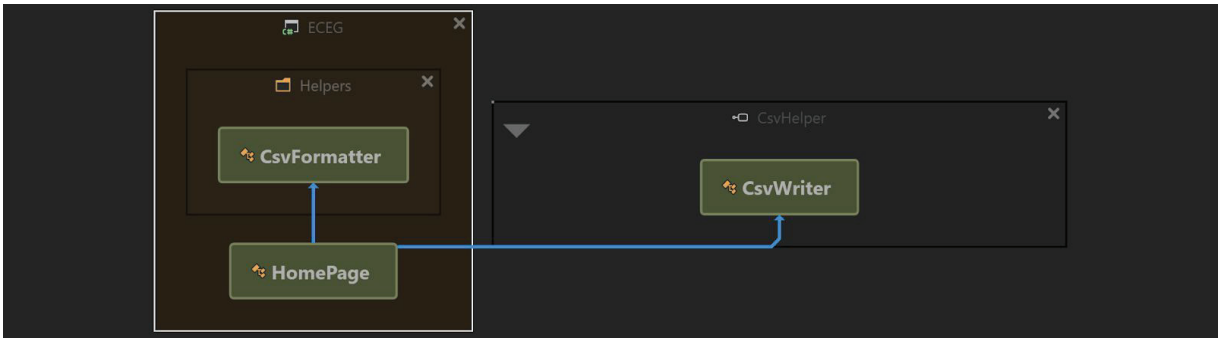


Figura 42.

Clase LogGenerator

Para poder llevar a cabo un seguimiento de lo que los usuarios buscan o si hay algún intento de inundación de peticiones por parte de alguna máquina y demás, se ha decidido implementar una clase que se encarga de volcar en un log los predicados de las búsquedas hechas así como la dirección IP desde la que se ha hecho la petición al servicio.

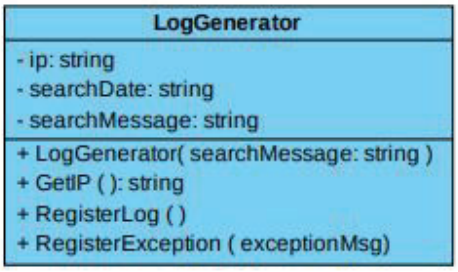


Figura 43.

La clase contiene tres atributos entre los que figuran la ip de la persona que hace la búsqueda, la fecha en la que se realizó y el predicado que se generó al hacer la búsqueda.

A la hora de crear la instancia del log con el predicado generado, se recopila la información sobre la IP del usuario y la fecha de la búsqueda. Una vez generada la

[9] Enlace del repositorio de la librería utilizada: https://joshclose.github.io/CsvHelper/

instancia podemos llamar a los métodos RegisterLog o RegisterException para poder volcar en el fichero de logs dentro del proyecto el evento que queramos guardar.

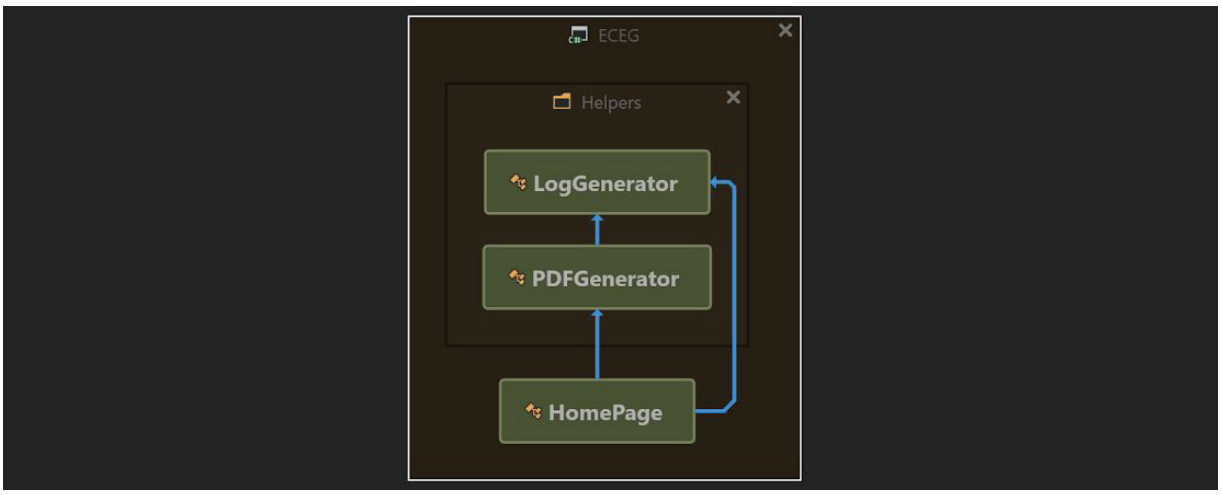


Figura 44.

Clase PDFGenerator

Adicionalmente, se ha añadido al servicio un método mediante el cual el usuario podrá hacer dos acciones adicionales a las que ya estaban disponibles anteriormente: descargar los resultados que han salido de una búsqueda o bien poder descargar los detalles de una gramática en concreto.

Esto se implementó por una simple razón: el usuario objetivo de la aplicación son investigadores que normalmente están acostumbrados a usar libros y elementos físicos para poder realizar las investigaciones. Que para tener que usar la aplicación y usar los resultados tenga que pasar horas delante del ordenador ante la pantalla puede no ser lo mejor para él, por lo que darle la posibilidad de generar un PDF que se pueda imprimir fue lo más sensato.

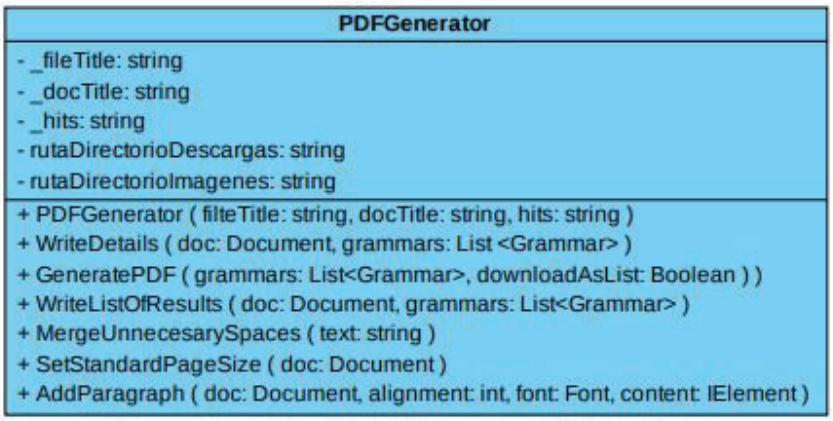


Figura 45.

La clase posee tres atributos principales: el título del fichero que se va a generar, el título del documento, el número de resultados que se van a descargar y las diferentes rutas a los directorios tanto de descargas como de imágenes para obtener los logos.

Por otro lado, el número de métodos es mucho mayor y podemos encontrar desde un constructor para poder emplear las funciones de la clase hasta los encargados de introducir el contenido de la lista de gramáticas en el fichero a generar. Dentro de la clase podemos destacar como método más importante a GeneratePDF.

Método	Funcionalidad
PDFGenerator (fileTitle, docTitle, hits)	Constructor de la clase. Genera una instancia de la clase PDFGenerator.
GeneratePDF (grammars, downloadAsList)	Método encargado de generar un fichero PDF con los resultados que le han inyectado de la manera indicada por downloadAsList. Si ese parámetro es verdadero entonces se generará el fichero en formato de lista. De no ser así, se genera el documento con sus detalles.
WriteDetails (doc, grammars)	Usado para poder volcar en el documento los detalles de una gramática seleccionada por el usuario.
WriteListOfResults (doc, grammars)	Usado para poder volcar en el documento la lista de resultados de la última búsqueda que se haya llevado a cabo.

MergeUnnecesarySpaces (text)	Método auxiliar de la clase que normaliza los espacios presentes en el texto que se le inyecta.
SetStandardPageSize (doc)	Usado para poder especificar las características de la página que se va a generar.
AddParagraph (doc, alignment, font, content)	Vuelca un párrafo en el documento PDF

Tabla 9.

Esta clase, como la totalidad de este módulo, está abierta a una fácil extensión de funcionalidad ya que tiene un gran nivel de abstracción y flexibilidad. A continuación podemos ver cómo está distribuida lógicamente en el proyecto.

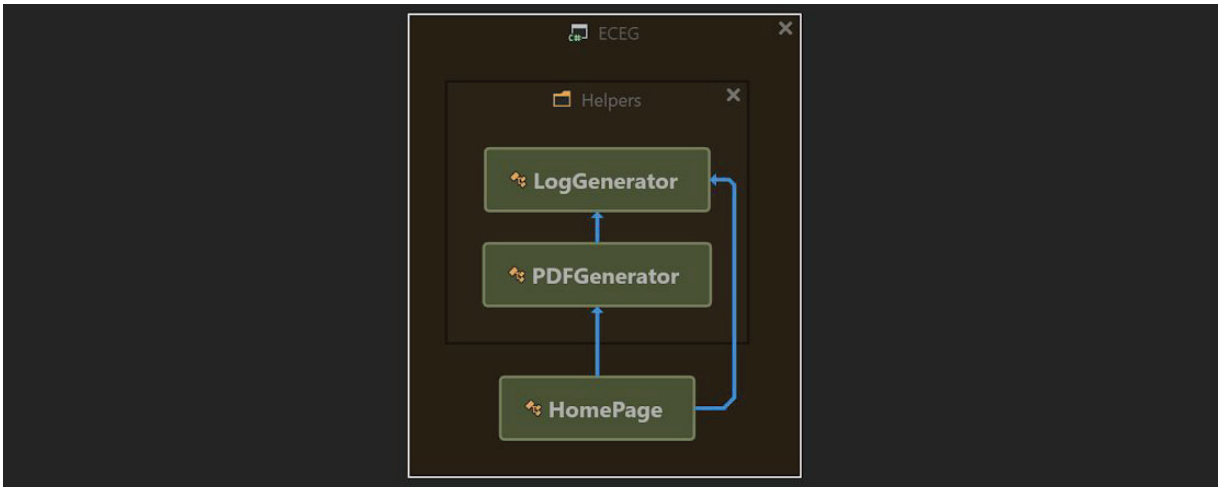


Figura 46.

Esta clase, en el caso de haber algún error en la generación del PDF, añadirá una excepción en un fichero de logs para poder hacer una traza de una forma mucho más cómoda.

Clase FilterTextHelper

En esta ocasión podemos hablar un poco sobre una clase encargada de convertir los parámetros que el usuario ha introducido para llevar a cabo la búsqueda a su forma en lenguaje natural.

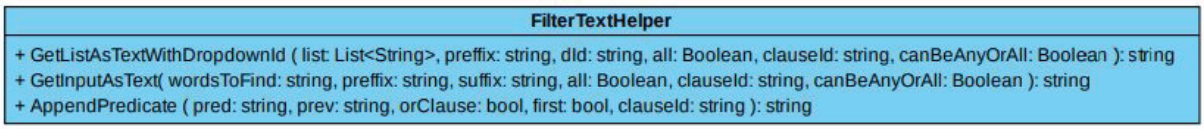


Figura 47.

Podemos encontrar no solo los métodos para poder convertir los parámetros de entrada del usuario en un texto en lenguaje natural, sino también el método que combina todos los predicados generados por los campos de búsqueda del usuario en un predicado final más grande que contiene a todos los demás de manera ordenada.

Para una mayor clarificación de esto, se ilustrará con un ejemplo: la importancia de la clase FilterTextHelper es poder transformar los parámetros que el usuario selecciona en el buscador:

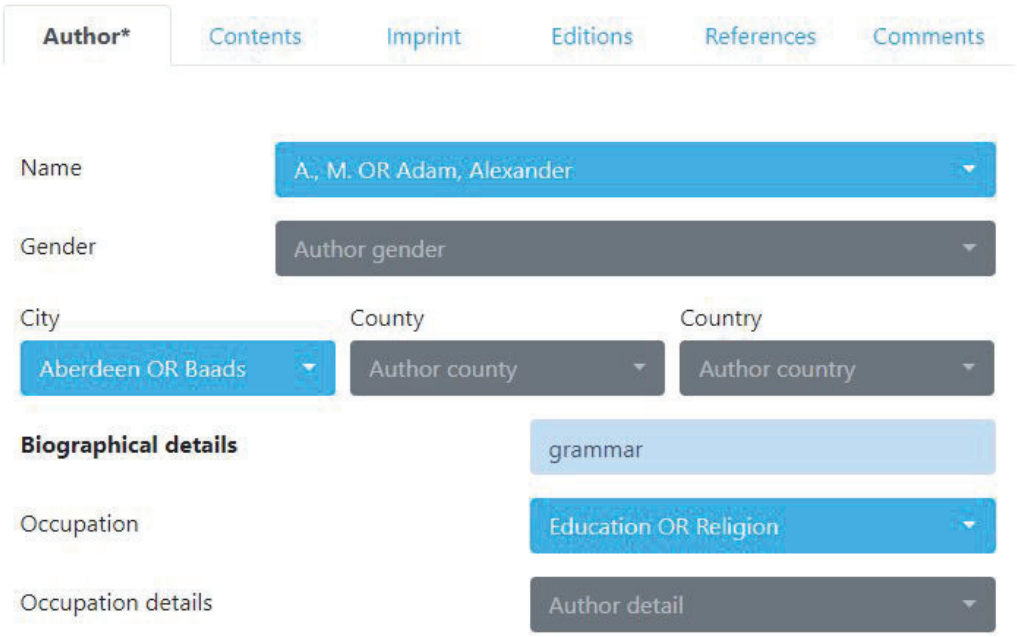


Figura 48.

En un predicado mayor en lenguaje natural que permita su comprensión más sencilla y poder realizar operaciones a posteriori, como el que vemos a continuación:

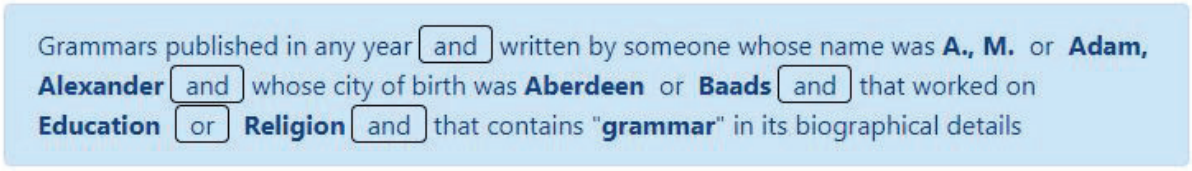


Figura 49.

Este añadido es muy interesante debido a que muchas veces al realizar una búsqueda, también puede ser curioso poder consultar los datos con unos parámetros ligeramente diferentes para comprobar si se da alguna relación y demás.

Para añadirle valor al programa, se ha implementado también una manera por la que el usuario no tiene por qué acceder a los filtros que se le proveen en el caso de querer ampliar la última búsqueda que ha hecho. Para ello, ciertos campos del predicado resultante quedan destacados. Por una parte podemos destacar las píldoras **AND** u **OR**, que pasan de una a otra y viceversa. Por otra parte, algo más interesante, podemos pulsar sobre cada uno de los nombres que aparecen resaltados en la Figura 49 y eliminarlos de la siguiente búsqueda manteniendo todos los parámetros anteriores. Al realizar cualquier cambio en este aspecto, hay que tener en cuenta que la búsqueda no se hace automáticamente, sino que, como podemos ver a continuación, se le avisa al usuario de que tiene que realizar la nueva búsqueda para que los cambios se lleven a cabo.

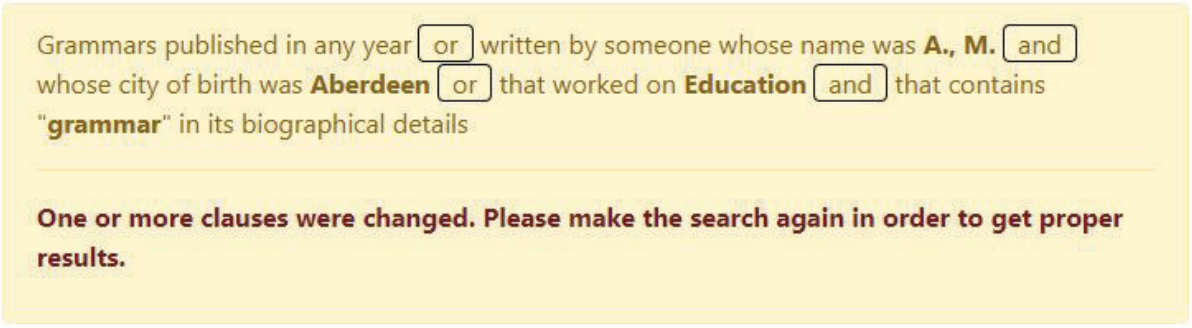


Figura 50.

Cualquier cambio introducido en el predicado que se muestra actualiza también los filtros pertinentes en las pestañas respectivas. Nótese que no ocurre al revés debido a que esta pequeña información es en base a una búsqueda que se ha hecho ya y tiene un mero carácter informativo. Es una vista de la búsqueda previa con la que el usuario puede interactuar mediante acciones puntuales.

Método	Funcionalidad
GetListAsTextWithDropdownId (list, preffix, dId, all, clauseId, canBeAnyOrAll)	Método utilizado para poder convertir los parámetros de un selector en su homónimo en lenguaje natural. Ej: si en el selector tenemos marcado “A., M.” y “Adam, Alexander” serán convertidos a “whose name was “A., M.” or “Adam, Alexander”. Dándole al usuario la capacidad de poder hacer click en el nombre que quiera para eliminarlo de la futura búsqueda.
GetInputAsText (wordsToFind, preffix, suffix, all, clauseId , canBeAnyOrAll)	Es un método con una funcionalidad muy parecida al anterior. La diferencia de este con el otro es que en este caso no se encarga de un selector, sino de un campo de texto libre.
AppendPredicate (pred, prev, orClause, first, clauseId)	Método al que se recurre cuando se ha generado un subpredicado y se necesita añadirlo al predicado final compuesto.

Tabla 10.

Debido a esta funcionalidad, es una clase no solo muy interesante para el proyecto, sino también muy importante en cuanto a la funcionalidad añadida que aporta al servicio. Su distribución lógica es la siguiente:

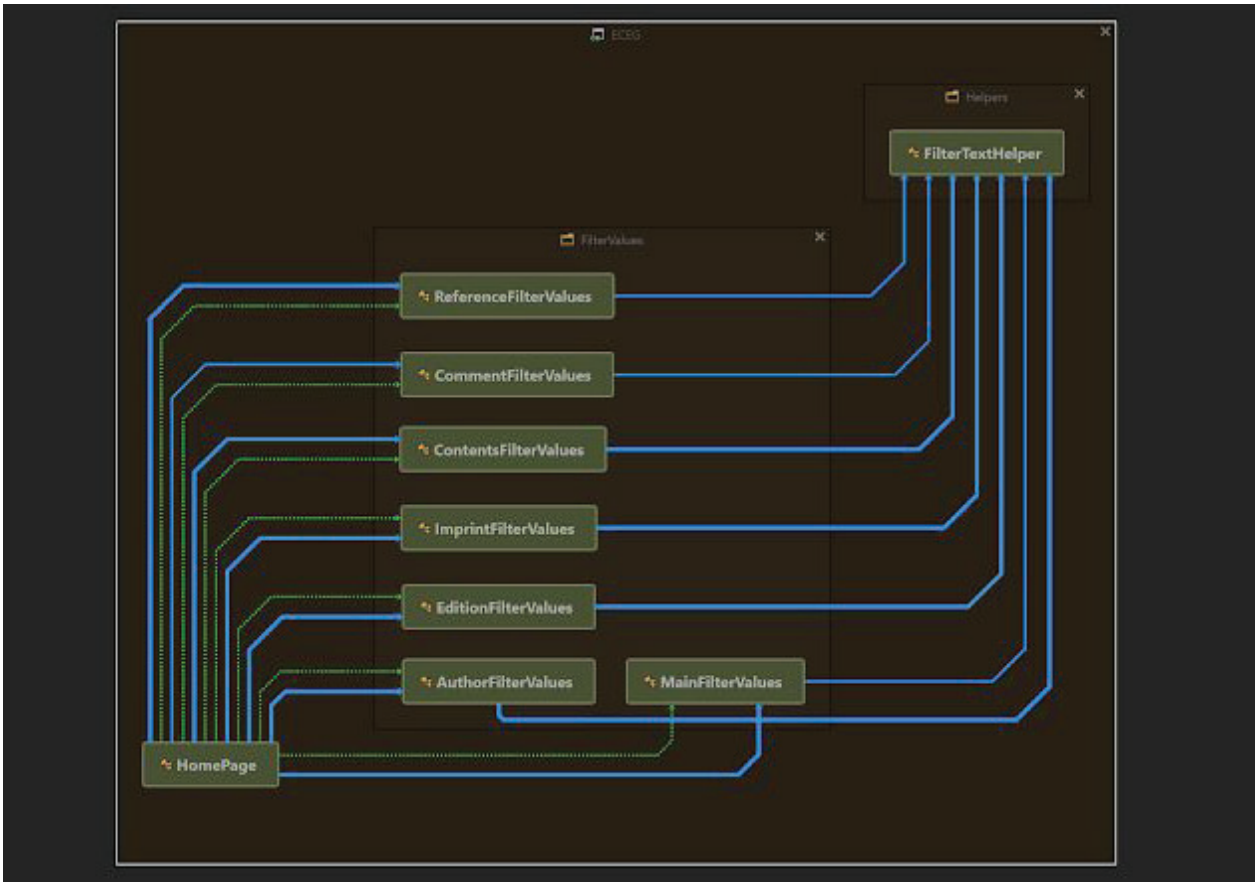


Figura 51.

Clase StringHelper

La última de las clases de este módulo de apoyo al proyecto es StringHelper. Podemos considerarla como una clase de extensión a la clase string que, sin duda, fue totalmente necesaria para poder realizar el buscador del servicio de la manera más eficiente posible.

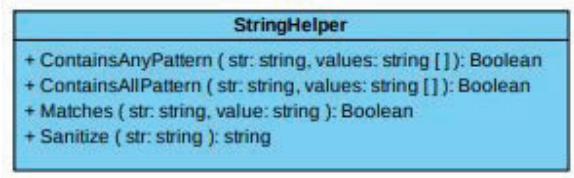


Figura 52.

La clase pretende añadir funcionalidad extra al tipo string mediante algunos métodos sobre contención de elementos de texto dentro de una lista de cadenas, similares a métodos existentes en otros lenguajes de programación como Python.

Método	Funcionalidad
ContainsAnyPattern (str, values)	Busca coincidencias de los patrones de la lista values en la cadena str. Devuelve true en el caso de que algunos de los patrones de la lista aparezcan en la cadena y false en caso contrario.
ContainsAllPattern (str, values)	Busca coincidencias de los patrones de la lista values en la cadena str. Devuelve true en el caso de que la totalidad de los patrones de la lista aparezcan en la cadena y false en caso contrario.
Matches (str, value)	Comprueba si la cadena str cumple con el patrón value
Sanitize (str)	Convierte la cadena de texto str en un formato correcto para poder aplicarle los métodos anteriores de manera que no ocurran errores en el proceso.

Tabla 11.

Submódulo Comparers

Debido a la similitud en el funcionamiento de las clases de este módulo, en vez de tratar cada una por separado se ha decidido englobarlas a todas en un apartado y comentar su funcionamiento.

Las clases que se ven afectadas por los comparadores personalizados son: City, County y Country. Esto se debe a que, de otra manera, al haber nombres iguales en la base de datos, haya duplicados en los campos de búsqueda. Tras comprobar que solamente se diferencian en la cadena de texto y no en un tratamiento especial, se optó por crear un comparador que no hiciera uso de la cadena de texto en el nombre sino más bien de su identificador de la base de datos, así podemos desechar los elementos repetidos y hacer las búsquedas correctamente.

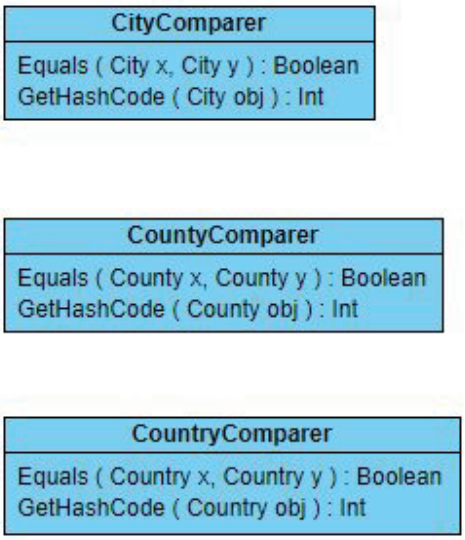


Figura 53.

Los métodos de estas clases son los estándares de un comparador personalizado, teniendo un método que nos dice si dos objetos son iguales y otro para poder obtener un código hash de una instancia.

Estas clases se utilizan a la hora de cargar los registros de los modelos correspondientes en la aplicación. Aunque si bien se podría haber creado el comparador de manera anónima en el método que las cargara, se optó por esta manera para dejar un código más limpio y mejor organizado. A continuación se muestra la distribución lógica de este submódulo en el proyecto.

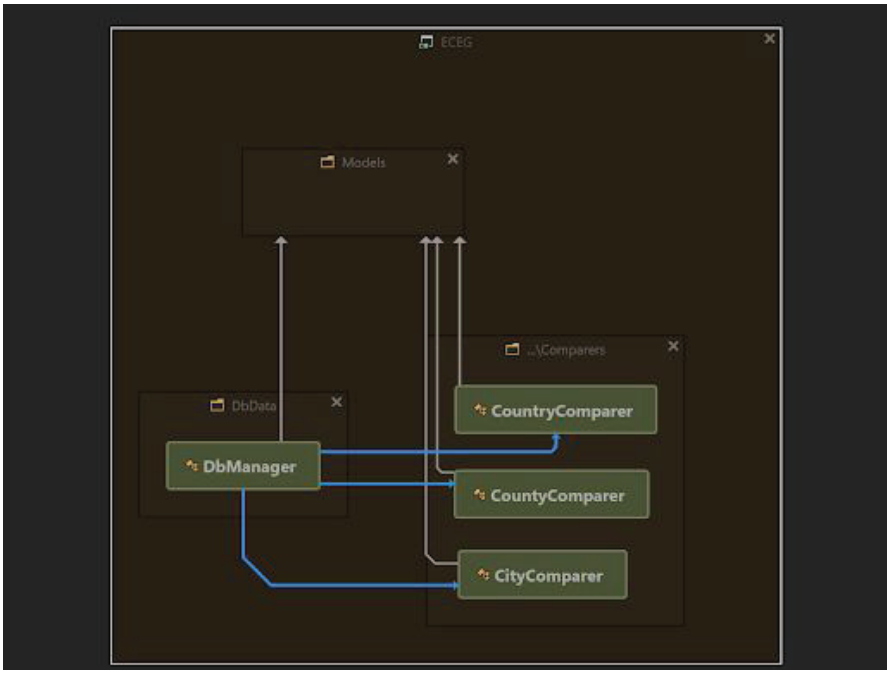


Figura 54.

Agrupadores de campos

Para poder enlazar la interfaz que el usuario utiliza a la hora de explorar los registros de la base de datos y crear un predicado de búsqueda con la clase encargada de crear ese predicado se ha decidido implementar un grupo de clases encargado de contener todos los parámetros que el usuario haya podido marcar a la hora de crear la búsqueda así como los valores por defecto de los mismos en el caso de que no se haya marcado nada en esa sección. Estas clases están divididas en cada una de las pestañas por las que el usuario puede realizar cambios en la búsqueda así como el filtro principal.

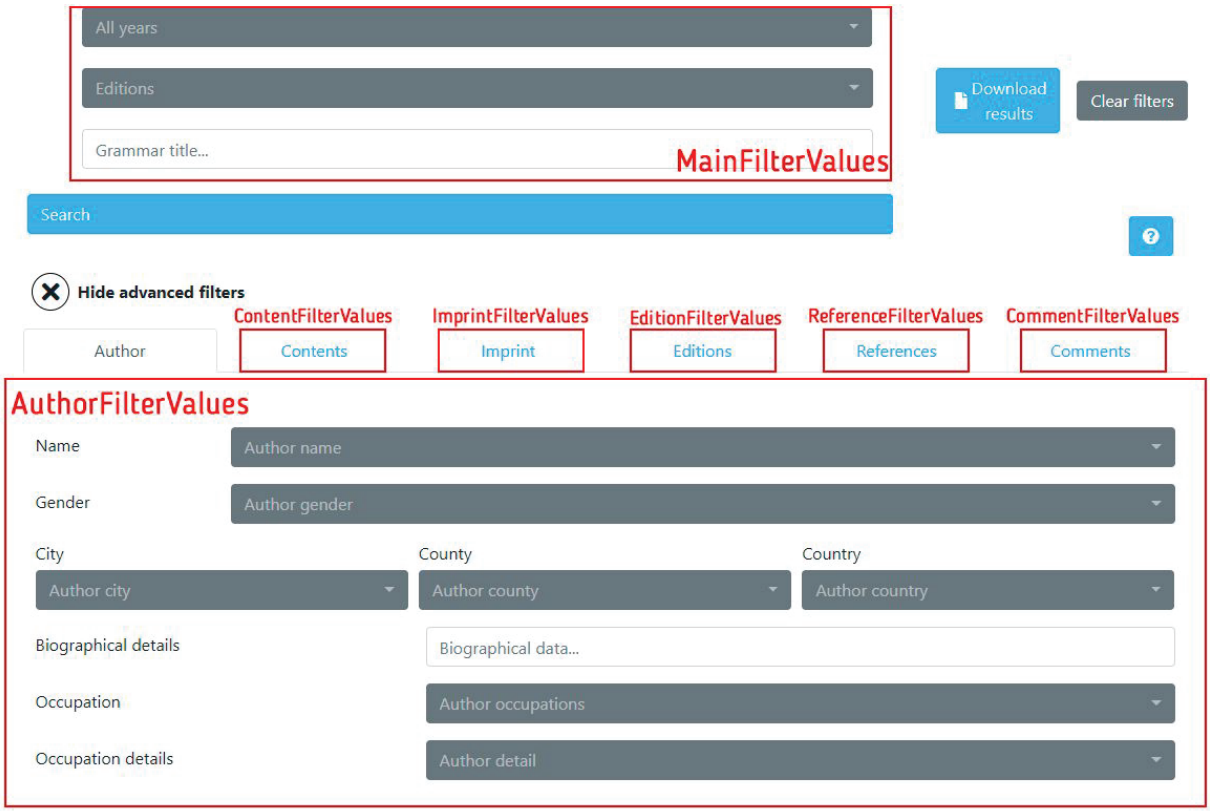


Figura 55.

Debido a que todos tienen exactamente la misma funcionalidad, se ha decidido explicar todas las clases creadas a la vez y no cada una por separado.

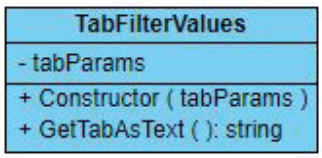


Figura 56.

Se ha optado por ilustrar a todas las clases creadas con una clase sola para una mejor ejemplificación: TabFilterValues. Esta clase contendrá todos los componentes de la vista de cada pestaña. En el caso de, por ejemplo, la pestaña de *Author* podremos ver que contiene varias listas con los nombres de los autores seleccionados, géneros por los que se quiere buscar, países de nacimiento, etc. Así como otros textos tales como las características que tenga el campo de datos biográficos del autor.

Estas clases tienen el constructor que inyecta los valores insertados en la interfaz dentro de los atributos de la clase así como disponer de un método por el cual podremos obtener en lenguaje natural una adaptación de los parámetros que se hayan seleccionado.

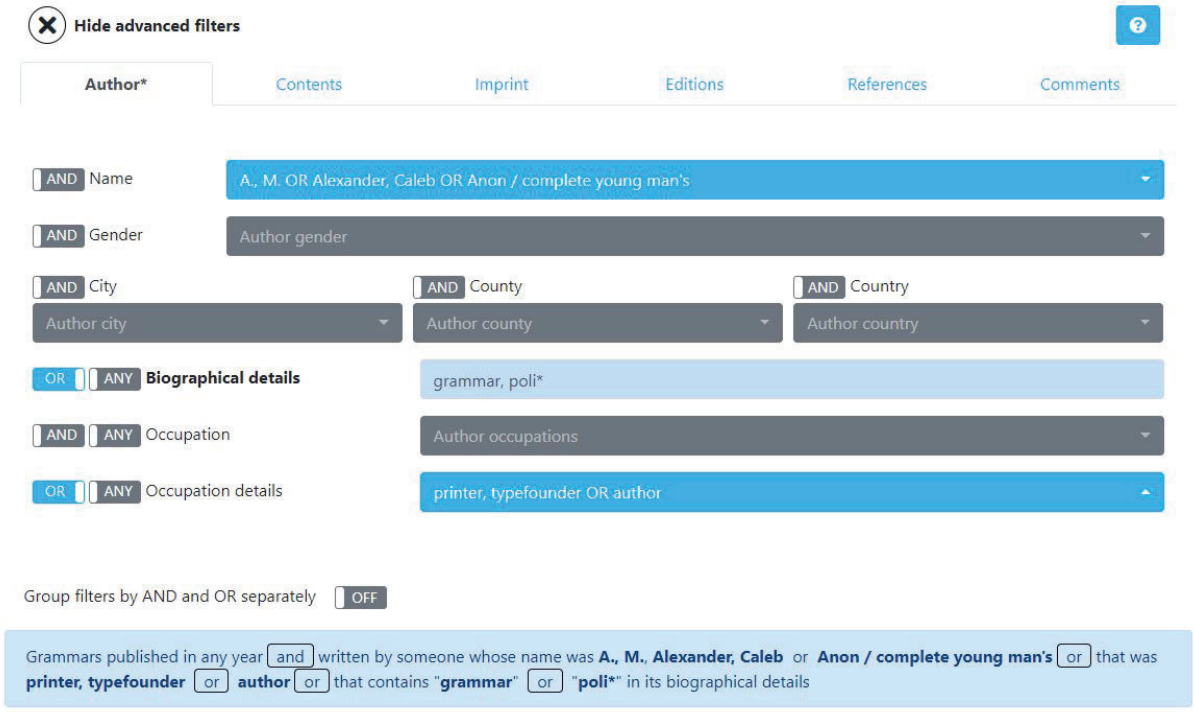


Figura 57.

Se optó por generar una clase por pestaña debido a que de cara a un futuro mantenimiento o actualización podemos tener todo mucho más modularizado, fácil de acceder y con un gran potencial de ampliación.

A continuación podemos observar el uso de estas clases dentro de la organización lógica del proyecto:

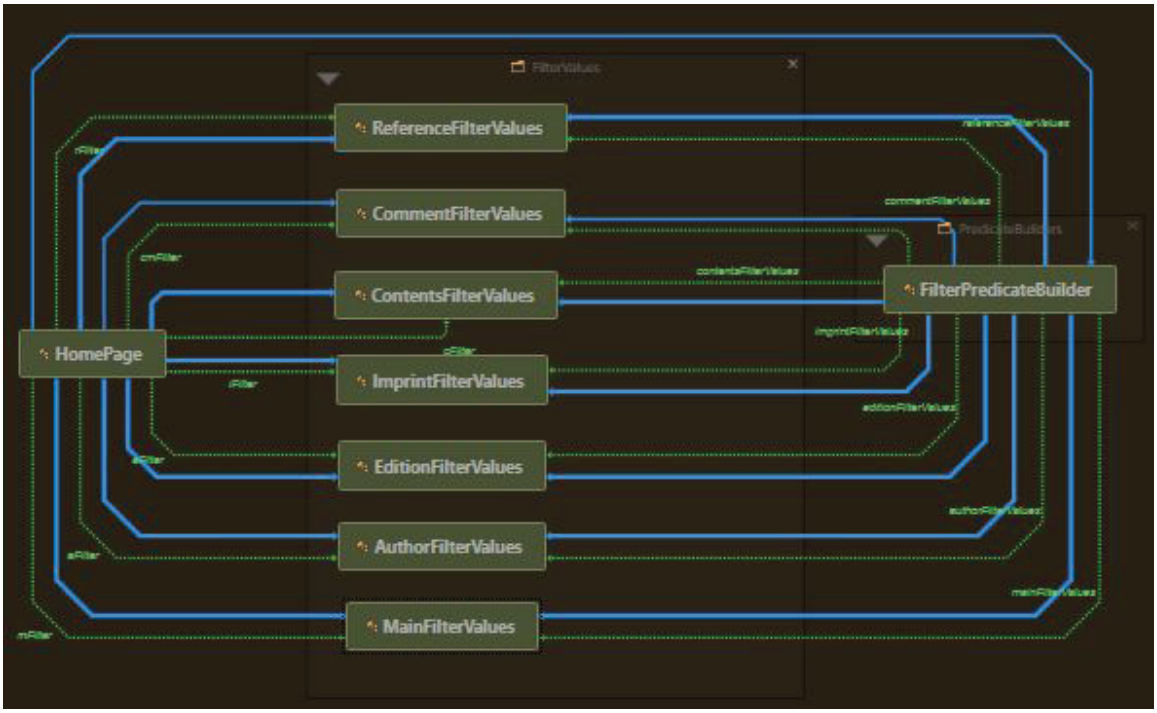


Figura 58.

Creador de predicados

Para poder llevar a cabo la búsqueda de las gramáticas, primero se tiene que formar un predicado que nos indique las caraterísticas que tenemos que encontrar en cada registro y así calificarlo como válido y mostrarlo. La clase FilterPredicateBuilder es la encargada de esa función, siendo una de las clases más innovadoras en ese aspecto, generando en tiempo de ejecución predicados que el usuario puede haber introducido.

Para poder crear el predicado con los parámetros que el usuario ha introducido en el buscador se hace uso de unas clases intermedias entre la capa del usuario y el creador de predicados que se encarga de recoger los parámetros introducidos y ordenarlos de tal manera que le facilita los elementos que esta clase necesita para poder hacer su función.

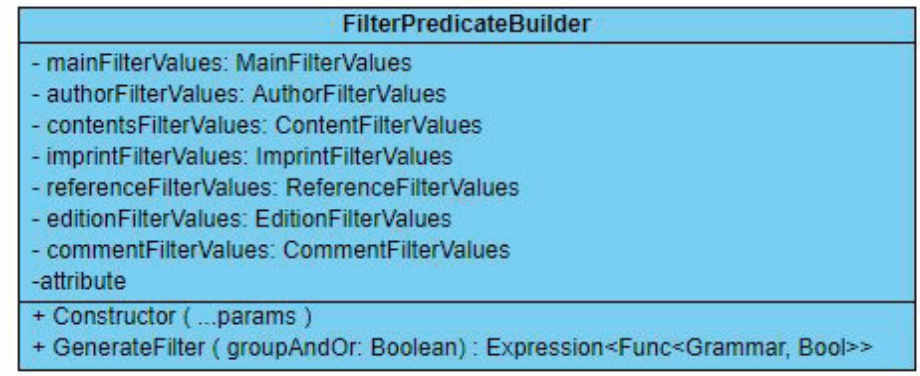


Figura 59.

Esta clase solamente tiene dos métodos: el constructor al que se le inyectan las clases intermedias con los valores que el usuario introduce y luego tiene el método que, gracias a lo que el usuario introduce, genera un filtro con un predicado adecuado a las necesidades. A continuación podemos ver cómo se ha usado esta clase dentro del proyecto.

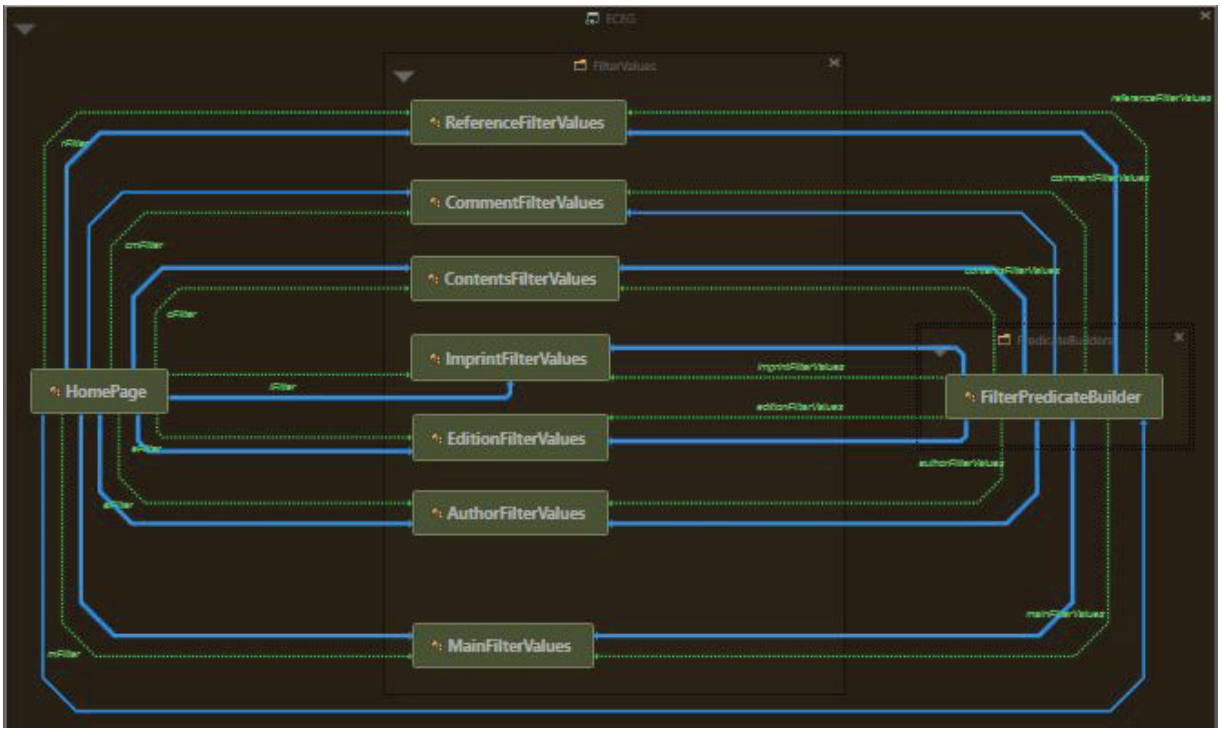


Figura 60.

El funcionamiento de esta clase no podría haberse llevado a cabo sin el uso de librerías de Microsoft para la gestión de colecciones de datos. En este caso, la librería

utilizada ha sido Linq, la cual es nativa ya en esta versión del IDE utilizado. Adicionalmente, para poder completar las tareas que habían de llevarse a cabo, se complementó el funcionamiento de esta clase con una librería externa denominada LinqKit[10] [11].

Tal y como dice la página del proyecto, *"LINQKit es un conjunto de extensiones gratuitas para los usuarios avanzados de LINQ a SQL y Entity Framework. Comprende lo siguiente:*

- *Una implementación extensible de AsExpandable()*
- *Una clase base de visitantes de expresión pública (ExpressionVisitor)*
- *PredicateBuilder*
- *Métodos abreviados Linq.Expr y Linq.Func"*[12]

Convirtiéndose, por esta razón, en una librería perfecta para las labores que podemos llevar a cabo gracias a PredicateBuilder debido a que es una clase que se encarga de formar predicados de manera dinámica. Exactamente lo que se busca con la aplicación.

Usando la clase PredicateBuilder se pueden crear las posibles consultas que el usuario puede hacer e irlas añadiendo a un predicado mucho mayor que es el que hará la consulta a la colección de gramáticas disponibles para cada persona.

Distribución lógica del proyecto

Por último en cuanto a la explicación de las clases desarrolladas dentro de este proyecto, se intentará ilustrar cómo se han organizado las conexiones de los módulos presentes en el desarrollo del programa software desarrollado.

[10] Plugin accesible desde https://www.nuget.org/packages/LinqKit/

[11] Podemos encontrar el código fuente del proyecto en https://github.com/scottksmith95/LINQKit

[12] Texto extraído y traducido de https://github.com/scottksmith95/LINQKit

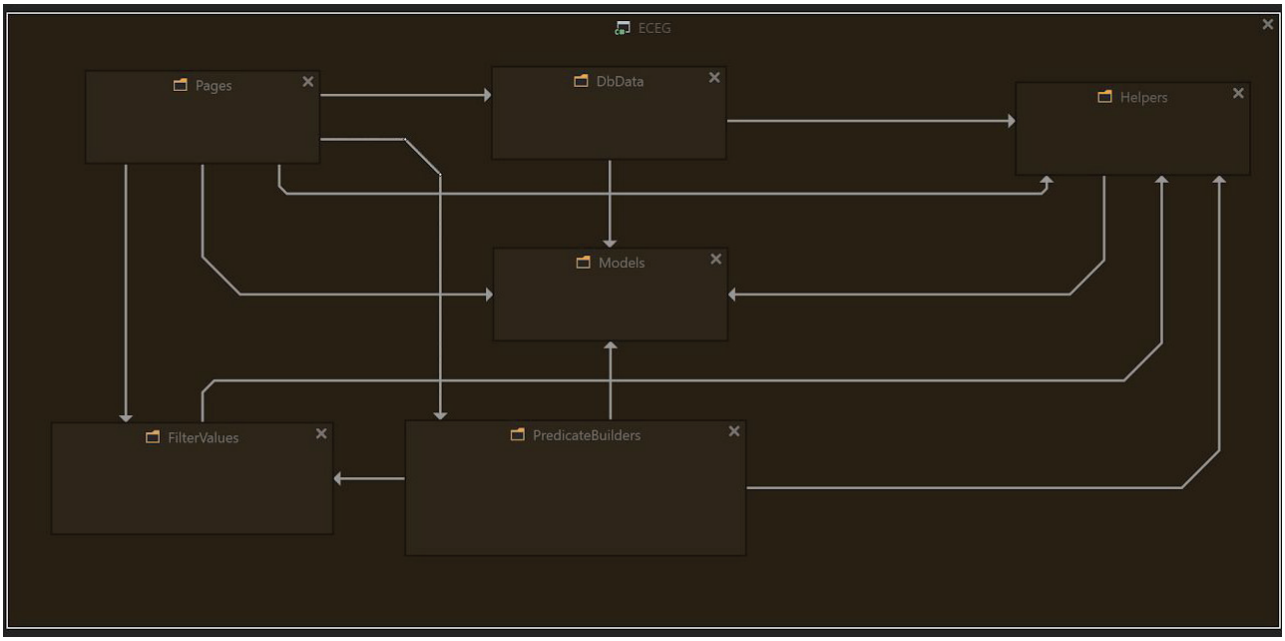

ECEG
Pages
DbData
Helpers
Models
FilterValues
PredicateBuilders

Figura 61.

Diseños de interfaces de usuario

Buscador y pestañas

Filtro principal y pestaña de autor

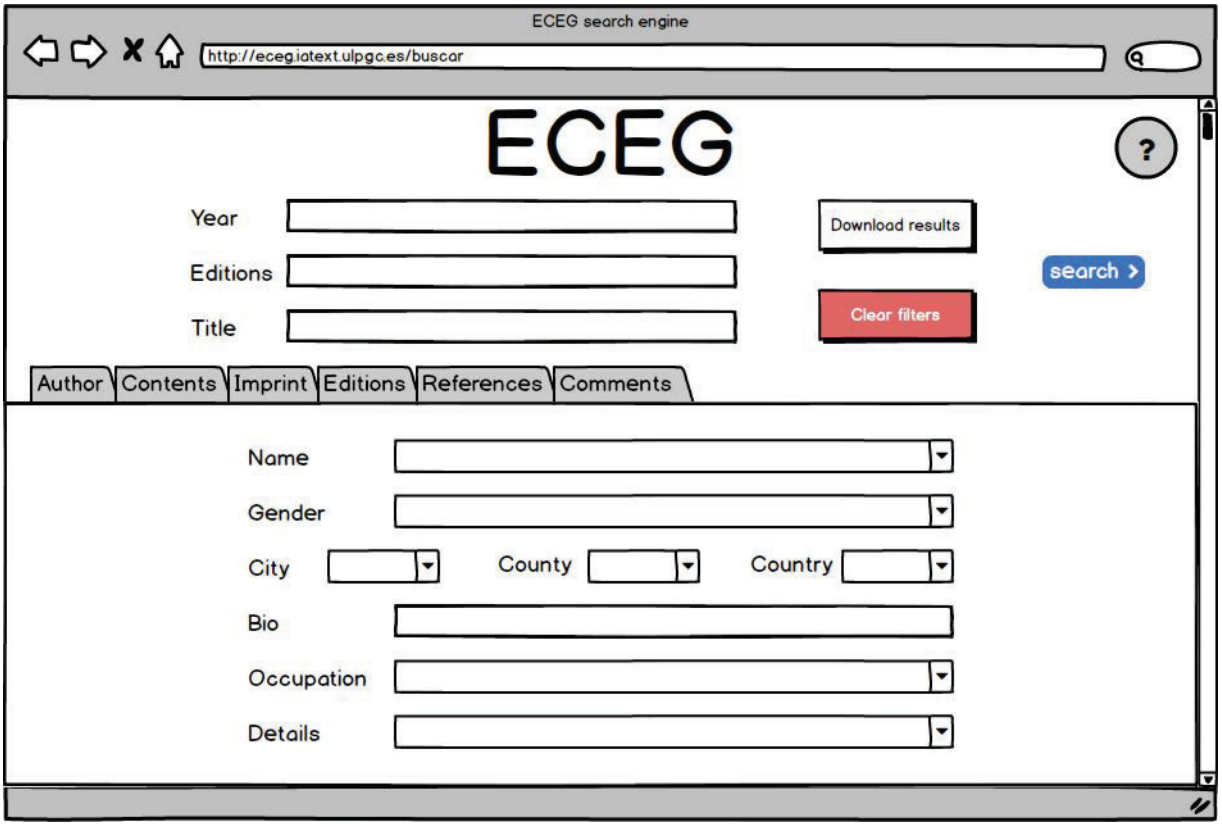


Figura 62.

Podemos observar en la ilustración algunos de los campos usados a la hora de hacer una búsqueda. Concretamente podemos encontrarnos los campos referentes a un filtro principal básico formado por campos de texto para buscar gramáticas publicadas en esos años, así como para las ediciones y el contenido del título de la misma. Adicionalmente, en esta parte podemos encontrar botones para poder descargar los resultados obtenidos y también el botón encargado de limpiar los parámetros que se hayan podido introducir en la última búsqueda así como el botón usado para llevar a cabo la búsqueda en la base de datos. Por otro lado, en la esquina superior derecha podemos encontrar un botón de ayuda que mostrará una ventana emergente con el manual de usuario de la aplicación para que los usuarios puedan exprimir el potencial del producto de la mejor manera posible.

Entrando dentro de las pestañas, en este caso es la relacionada con las características del autor. Podemos destacar no solo las etiquetas para cada campo sino la gran cantidad de desplegables ofertados para el usuario. Esto se ha hecho debido a que se prefiere elegir dentro de un abanico de opciones a tener que escribir así pudiendo eliminar el factor humano dentro de los errores que pudieran aparecer. Aparte de los desplegables, otra de las entradas del usuario es mediante campo de texto libre para la biografía del autor.

Pestaña Contents

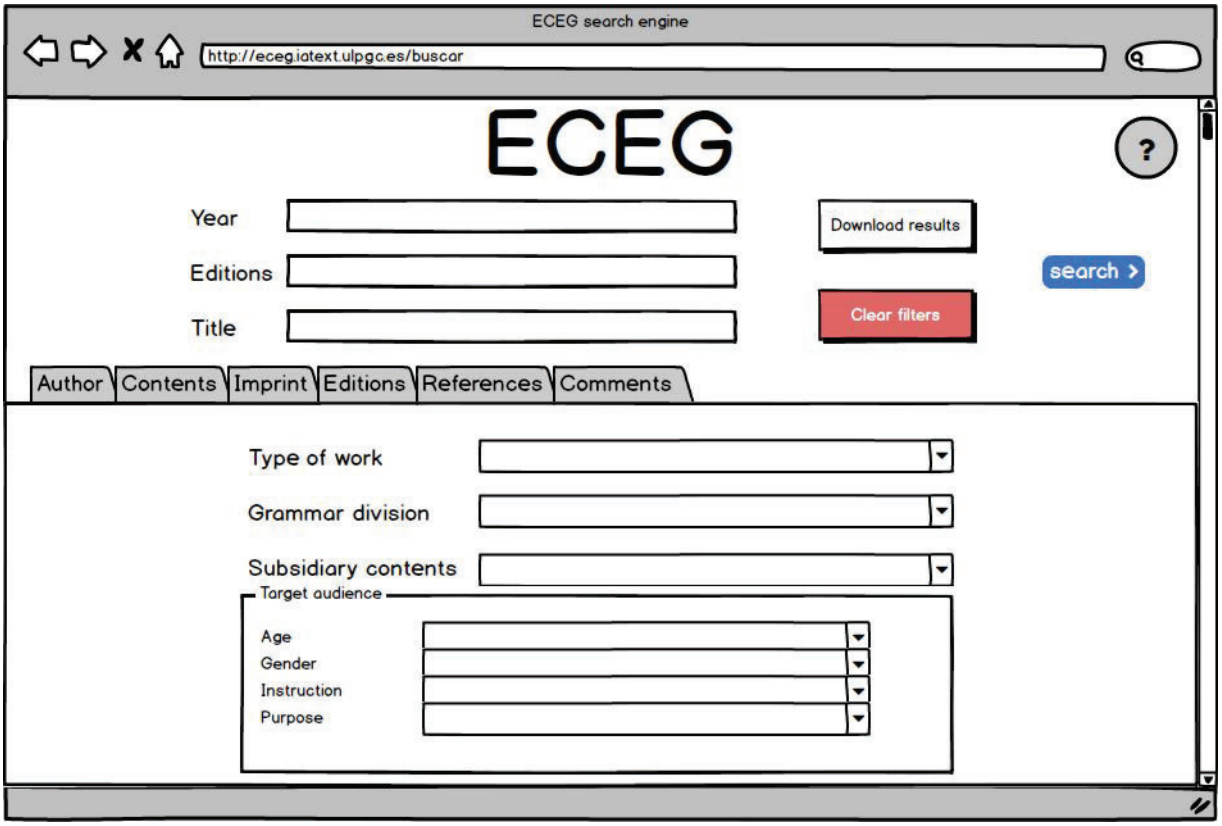


Figura 63.

Podemos seguir con la explicación de la pestaña de contenidos de la gramática. En la vista de esta pestaña podemos comprobar que tiene etiquetas para todos los campos sobre los que procede la búsqueda pertinente en ese grupo, teniendo en cuenta que son todo desplegables multiopción para evitar errores tipográficos por parte de los usuarios, facilitando, por tanto, el uso de la herramienta.

Para mantener la coherencia con el servicio anterior, se ha intentado mantener la misma agrupación en dos partes: una para el tipo de la obra, la división y el contenido subsidiario y un subgrupo en la vista para el público objetivo de la gramática.

Pestaña Imprint

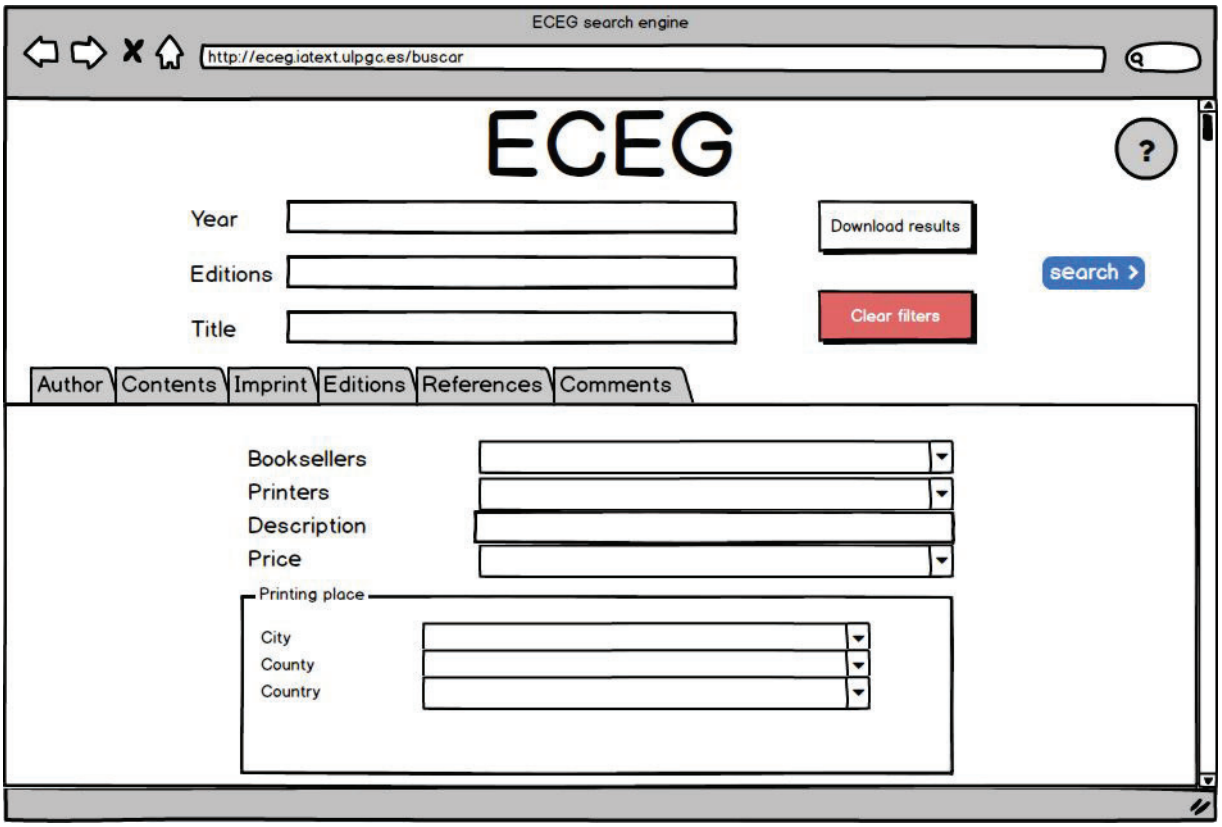


Figura 64.

De la misma forma que la pestaña anterior, se ha intentado mantener una distribución parecida a la que podemos ver en el servicio anterior a este. En la figura podemos observar cómo se han integrado los componentes en la vista mediante etiquetas para los campos y una gran variedad de desplegables para poder evitar el error humano. Cabe mencionar que el campo descripción es de texto libre porque podemos buscar solamente una palabra dentro del texto de la descripción física o una sola característica. Debido a esto, no valía la pena desplegar todas las opciones de la base de datos sino más bien que el usuario decida mediante entrada de texto lo que precisa.

Pestaña Editions

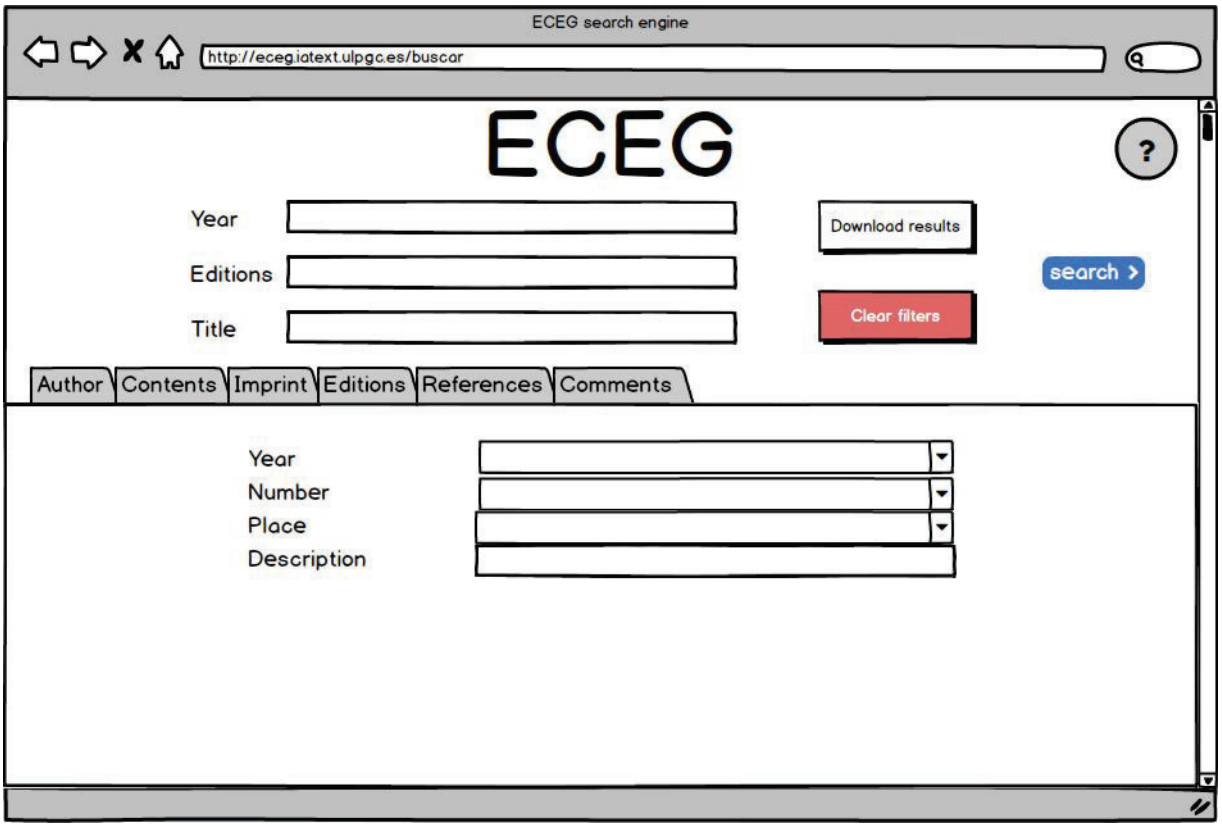


Figura 65.

Siguiendo la línea de las anteriores figuras, podemos observar la vista de la siguiente pestaña, encargada de los apartados de búsqueda en la zona de ediciones de la gramática que se quiera encontrar. Podemos observar cómo la pestaña está formada por 4 desplegables acompañados por sus correspondientes etiquetas. Estos desplegables tienen la capacidad de ser multiopción, por lo que el usuario no está limitado solamente a buscar solamente un valor por atributo presentado.

Pestaña References

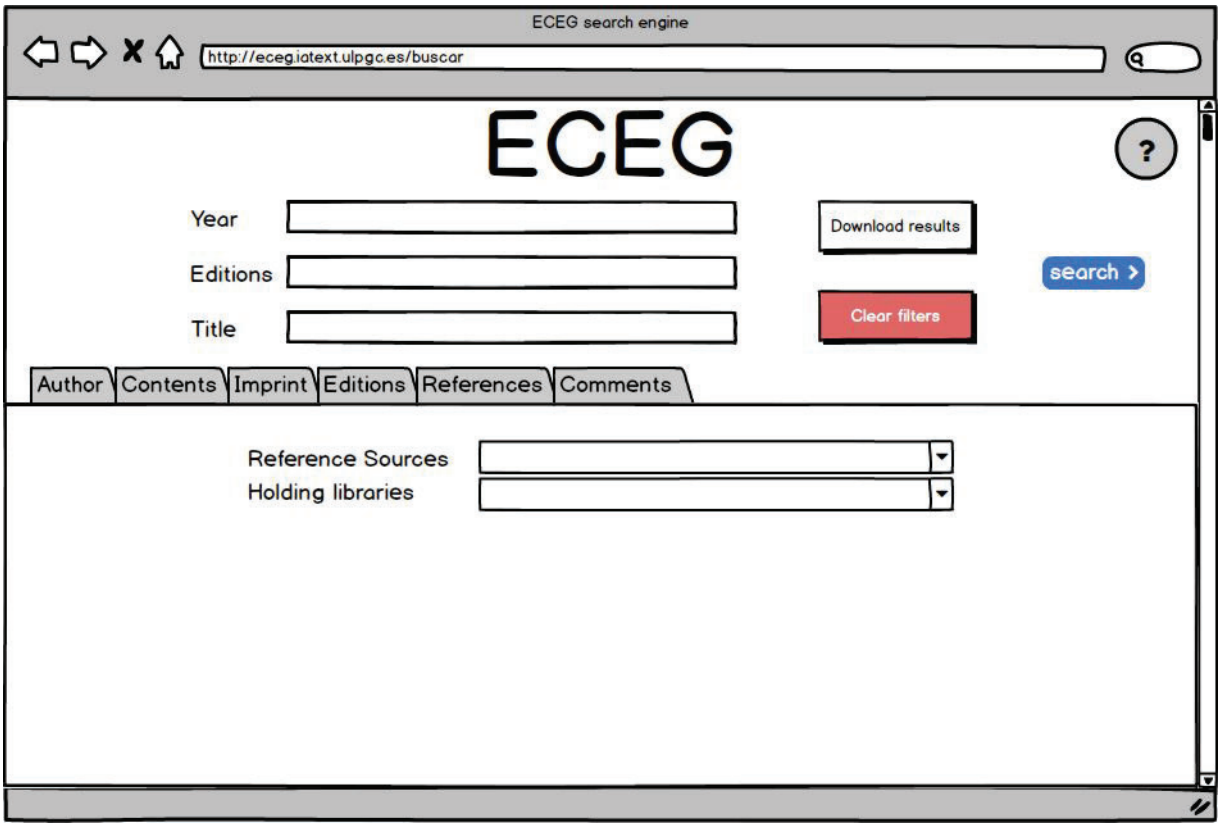


Figura 66.

Podemos observar en esta figura una de las dos últimas pestañas que quedan por dar una explicación. En este caso se trata de la encargada de la zona de las referencias de la gramática: tanto de las librerías que contienen las ediciones de la gramática como de las fuentes de donde se ha obtenido información de la misma.

Está formada por dos desplegables multiopción de la misma manera que el resto de las pestañas para que así el usuario tenga la capacidad de poder seleccionar los valores que le hiciera falta sin cometer ningún tipo de error ortográfico y así evitar una búsqueda incorrecta.

Pestaña Comments

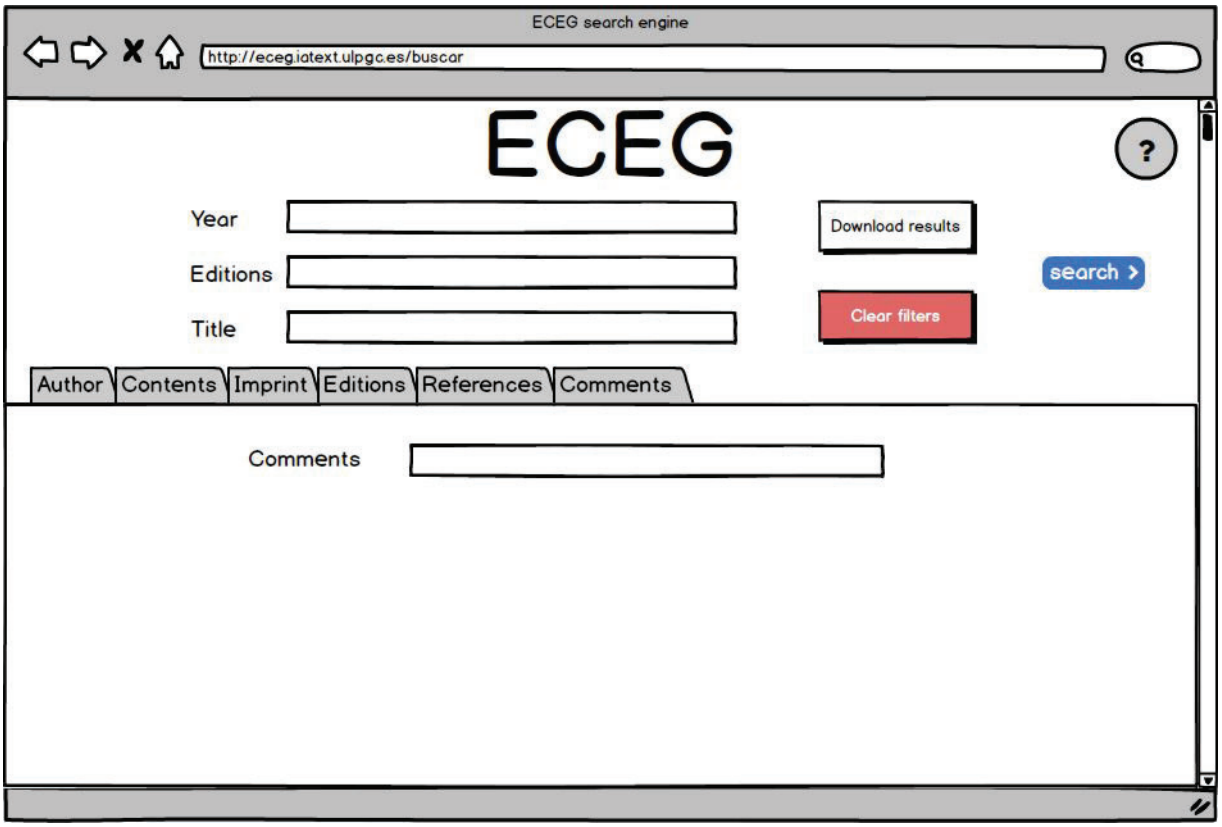


Figura 67.

La última de las pestañas se corresponde con el campo de comentarios de una gramática que queramos encontrar. Esta escueta pestaña está formada solamente por una etiqueta para identificar al campo y un campo de texto libre que el usuario puede rellenar. Este campo de texto puede también tener en cuenta caracteres comodín tales como el asterisco (*), en el caso de no saber cómo sigue la palabra o querer ocurrencias desde el inicio de la palabra hasta la posición del asterisco y a partir de ahí encontrar cualquier palabra, y el cierre de interrogación (?) en el caso de no conocer uno de los caracteres de lo que estamos buscando o simplemente darle ese grado de libertad al buscador.

Lista

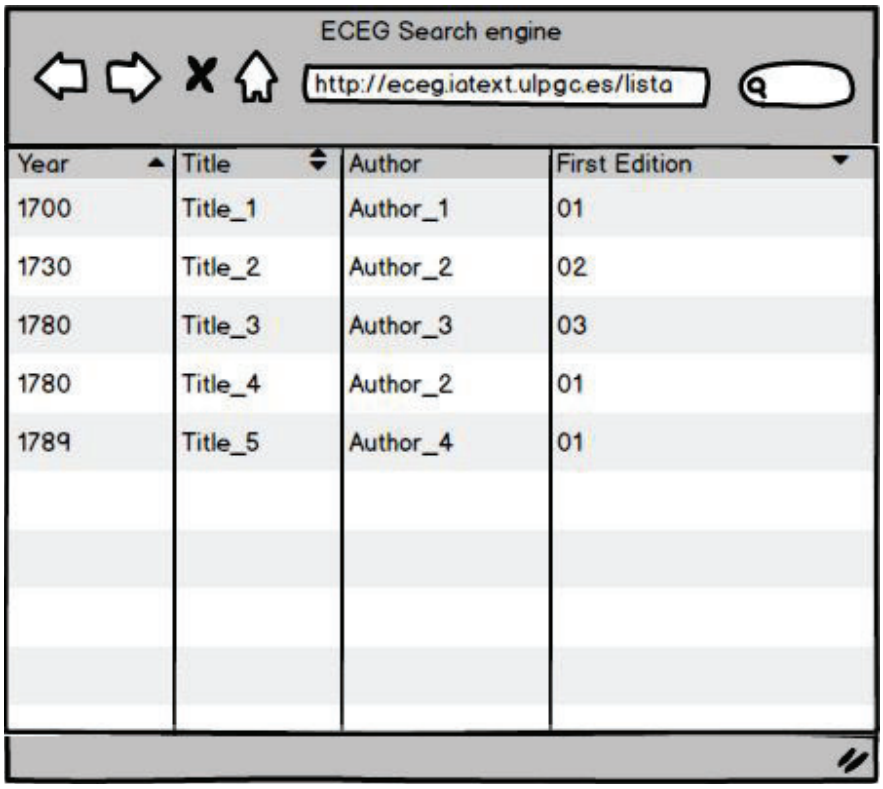


Figura 68.

Una vez terminada una búsqueda, los usuarios pueden consultar los resultados de la misma en forma de tabla con todas las gramáticas que cumplen los criterios que han introducido. En el caso de no haber ningún resultado un mensaje les notificaría.

Podemos observar en la figura que la tabla tiene cuatro propiedades: el año de publicación de la gramática, el título, el autor así como la primera edición conocida de la obra. Esto se debe a que estos atributos son considerados los más importantes debido a que es lo que más se suele consultar. En cualquier caso, al hacer click sobre cualquiera de las gramáticas el usuario es llevado directamente a la página de detalles de la gramática correspondiente.

Detalles

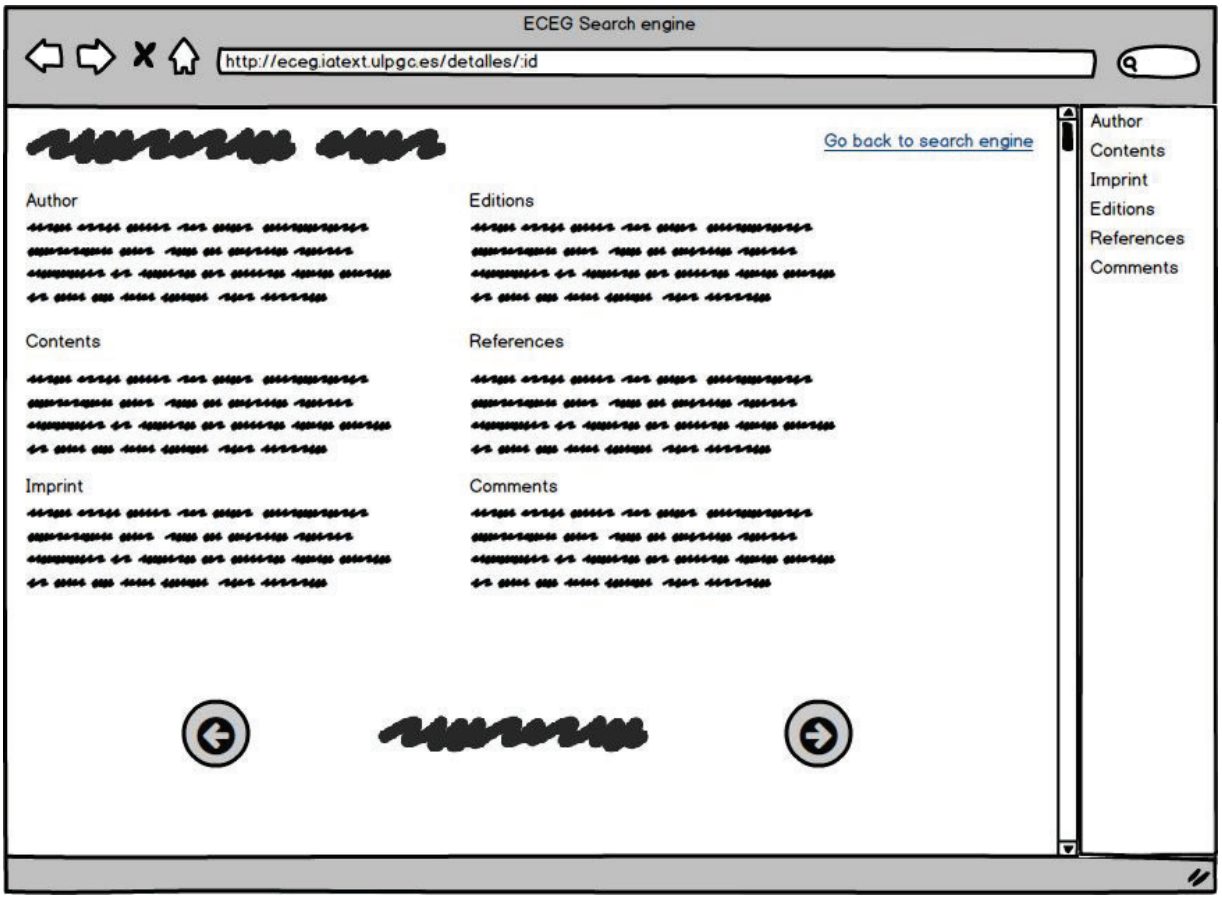


Figura 69.

Cuando el usuario lleva a cabo una búsqueda mediante el formulario y elige de entre los resultados disponibles un registro, es llevado a la página que podemos observar en la figura. Es una página donde podemos ver de manera ordenada y estructurada todos los datos de la gramática en concreto agrupada por las zonas que hemos visto en las pestañas de las figuras anteriores.

En la parte superior derecha tendremos un botón que nos devuelve a la misma zona de búsqueda para poder realizar una nueva. En el lado derecho podemos observar un menú lateral con el que podremos interactuar haciendo click en la zona que queramos resaltar y ver mejor. Por último, en la zona inferior tendremos botones para, en el caso de tener más de un resultado, ir navegando entre ellos sin tener que volver a la página anterior ni llevar a cabo acciones adicionales.

Conclusiones

Objetivos

Interfaz

Uno de los objetivos de la aplicación es que, para facilitar su uso, el diseño de la interfaz debía de ser clara, intuitiva, funcional y, lo más importante: simple. La aplicación ha sido probada por futuros usuarios de la aplicación a medida que se realizaban cambios y adaptaciones que facilitaran su uso en la medida de lo posible. En la mayoría de los casos nunca hizo falta una guía de ayuda para poder llevar a cabo las consultas.

Figura 70.

Aprendizaje

Aunque ya se conocía mínimamente el lenguaje de programación, tanto las herramientas como el entorno de desarrollo ha sido prácticamente nuevo, por lo que el proyecto ha requerido primero de cierto aprendizaje en el manejo de:

- Últimas versiones del framework ASP.NET

- Lenguaje de programación C#.
- Sistemas de gestión de bases de datos Microsoft Access 2016
- Entorno de programación de Microsoft Visual Studio 2019.
- Librerías de Microsoft utilizadas en el proyecto como LinQ o LinQKit.

Aunque se partía de ciertos conocimientos en programación web con HTML, CSS, Javascript y el framework de Bootstrap, se necesitó un pequeño tiempo de adaptación para aprender sobre las nuevas versiones de las tecnologías que no se tenía constancia, pudiendo haber retrasado la puesta en marcha de la fase de implementación en cierta medida.

No obstante, pese a lo comentado anteriormente, las herramientas utilizadas para la elaboración del proyecto han satisfecho en todo momento los requisitos de usuario en la fase de análisis.

Figura 71.

Nivel de aprendizaje

Se ha aumentado la destreza en el desarrollo de proyectos desde la definición de unos requisitos hasta llegar a un producto final usable.

Por otra parte, se ha indagado en el estudio de la programación web, asegurando aquellos conocimientos que ya se tenían, además de abarcar nuevos horizontes dentro de este campo a través del estudio de herramientas software de gran utilidad, como el framework ASP.NET o el lenguaje de programación C#, lo que ha hecho de este proyecto una experiencia altamente provechosa y gratificante.

Aplicaciones web adaptativas

La aplicación web es navegable tanto para ordenadores como para dispositivos móviles como tabletas y móviles. Esto se ha hecho debido a que cada vez está más a la orden del día hacer cualquier consulta con nuestro smartphone. Para llevar a cabo esta tarea se ha hecho uso del framework Bootstrap. Dado que ya se había trabajado anteriormente con la herramienta, no surgieron problemas durante el desarrollo del proyecto ni su aplicación.

Rediseño de la base de datos

El proyecto tenía asignada ya una base de datos conformada por el cliente. Esta base de datos, a la hora de utilizarla, tenía algunos aspectos que arreglar a tener en cuenta. Podemos destacar de entre las labores de mantenimiento las siguientes acciones:

- El lugar de publicación de la gramática estaba codificado en una columna separado por punto y coma con el formato idCiudad ; idCondado ; idPais. Se optó por dividir estos valores en tres columnas diferentes e integrarlas dentro de la tabla de las gramáticas prescindiendo de esa tabla, así las consultas eran mucho más sencillas y ágiles.

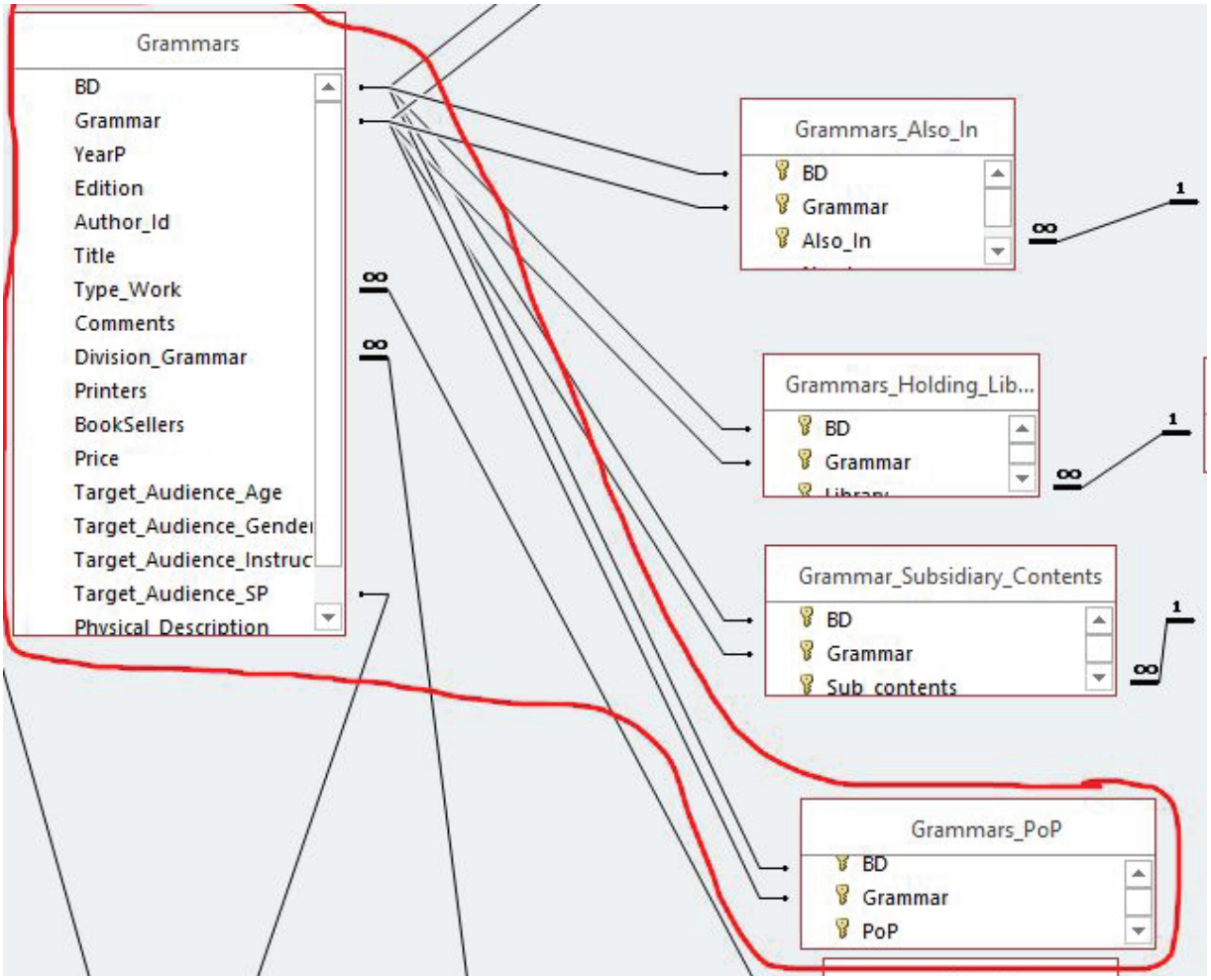


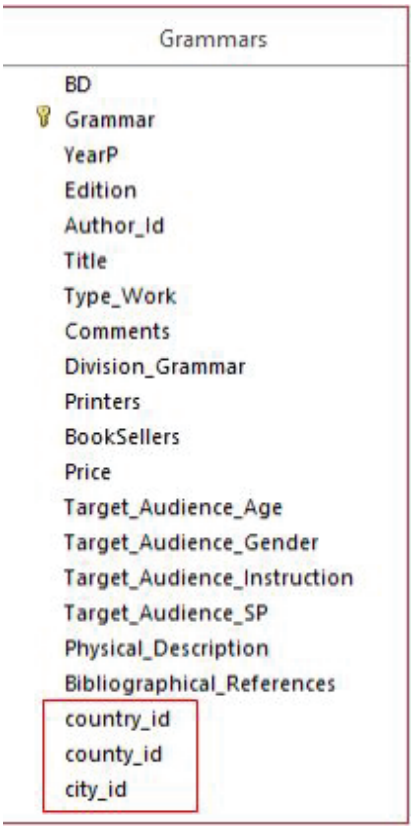
Figura 72.

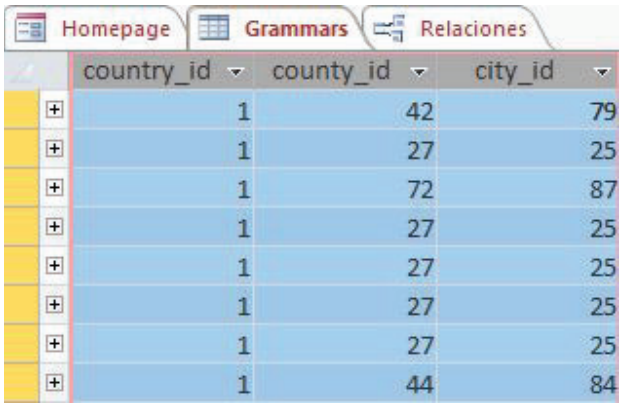
Homepage | Grammars | Relaciones | Grammars_Pop

BD	Grammar	PoP	Haga clic para agregar
2	186	1;15;62	
2	206	1;18;63	
2	31	1;18;64	
2	263	1;21;65	
2	224	1;21;65	
2	95	1;21;65	

Figura 73.

Teniendo en cuenta esto, se transformó en la siguiente organización:

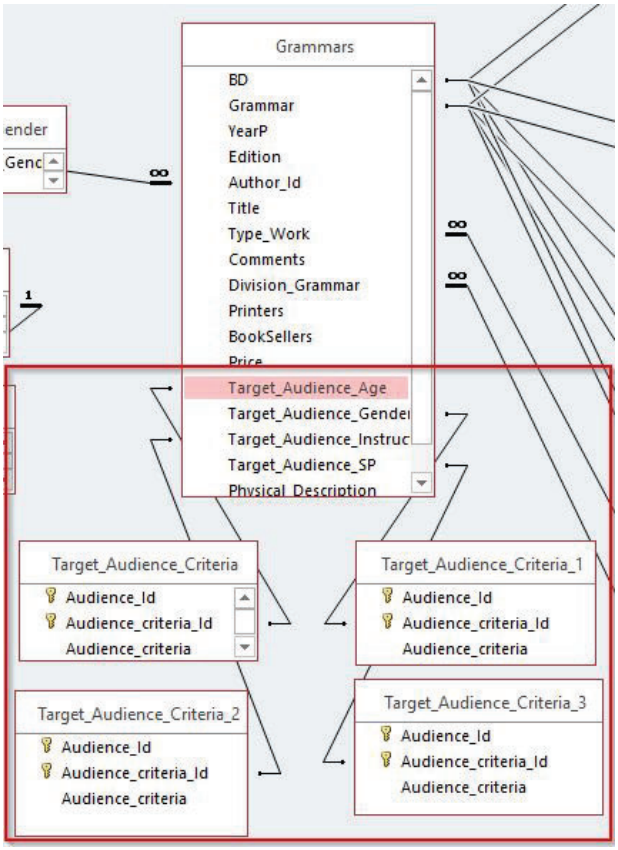




Figuras 74 (izquierda) y 75 (derecha).

De esta manera ahorramos espacio quitando una tabla de la base de datos y queda mucho más clara la relación de los campos.

- Los campos de audiencias objetivo en la tabla de gramáticas hacían todas referencias a la misma tabla. La cual estaba codificada como una combinación de una clave compuesta que no estaba definida. Para evitar las confusiones a la hora de hacer consultas e interpretar en el futuro la base de datos, se optó por dividir la tabla en cuatro tablas con algo menos de información pero mucho mejor compartimentada. Pudiendo pasar de la siguiente organización:



Audience_Id	Audience_criteria_Id	Audience_criteria
1	0	unknown / no copy located
1	1	children & adults
1	2	children
1	3	children & youth
1	4	youth
1	5	unknown / non-specified
1	6	adults
1	7	unknown / not consulted
2	0	unknown / no copy located

Figuras 76 (izquierda) y 77 (derecha).

A poder visualizar las relaciones de la siguiente manera:

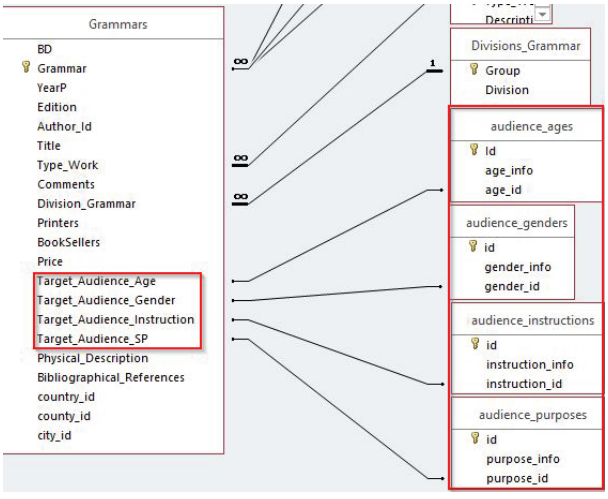


Figura 78.

Estos han sido los cambios más llamativos dentro de la base de datos, a los que hay que añadir otras modificaciones menores como es la creación de índices y la

eliminación de algunas columnas y tablas que no afectan nada en el desarrollo de la aplicación y por eficiencia se prescindió de ellas.

Con los cambios llevados a cabo y cambiando la organización de la base de datos, pudimos pasar del diseño original al siguiente sin perder ninguna información relevante para el proyecto.

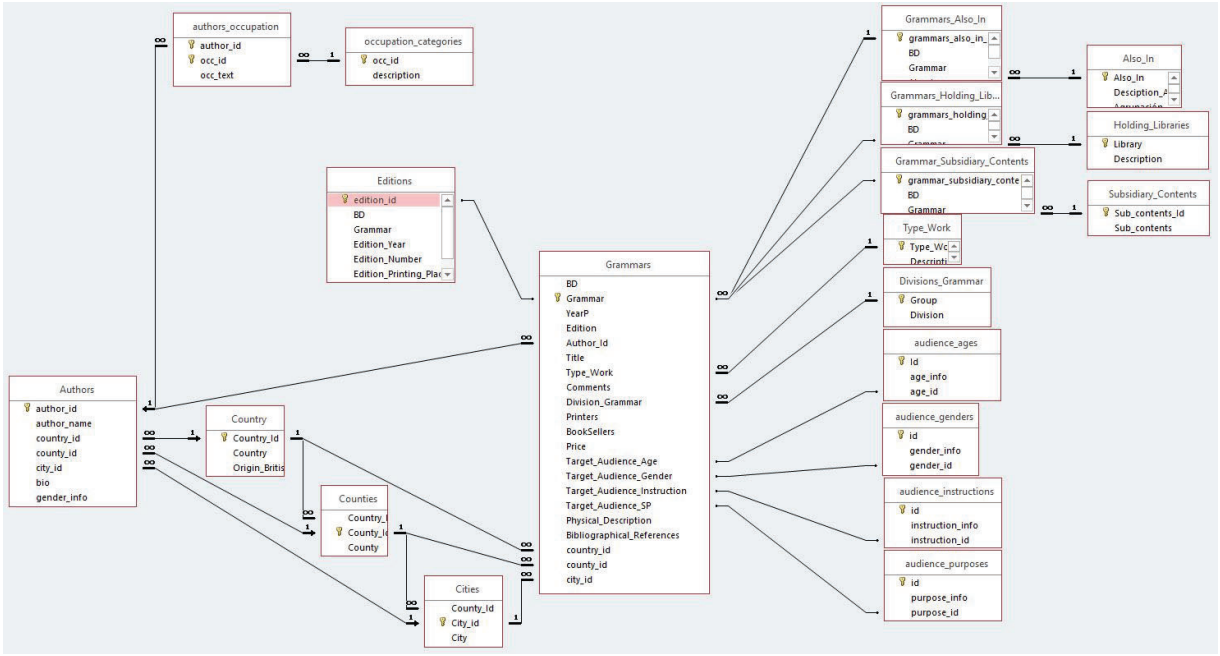


Figura 79.

Valoración personal

Cuando me lancé a tomar parte de este proyecto pensé que no estaba preparado para completarlo debido a que no conocía muy en profundidad algunas de las herramientas especificadas que se necesitaban, pero a medida que iba pasando el tiempo, fui aprendiendo poco a poco lo necesario para ir haciéndolo y lo que en un principio fue motivo de agobio o estrés, terminó por transformarse en algo satisfactorio y tomado como reto.

Este aprendizaje me acompañó durante todo el desarrollo del proyecto. Ha sido un proceso continuo y constante. Esa inseguridad se fue convirtiendo en confianza e ilusión por empezar a introducirme en el mundo laboral.

Con todo, he tenido la gran suerte de haber participado en un proyecto que me sirve como carta de presentación en futuras entrevistas de trabajo por dos motivos: los conocimientos necesarios para haberlo podido llevar a cabo están a la orden del día y

es una aplicación publicada y utilizada por el Instituto Universitario de Análisis y Aplicaciones Textuales (IATEXT).

Es por ello que mediante la realización de este proyecto esperamos mejorar el panorama de la metodología de investigación en el estudio de la educación, literatura y la sociedad inglesa del siglo XVIII.

Trabajos futuros

Cliente multilingüe

Un punto que pensamos que sería llamativo es la implementación de un módulo que pueda proveer a la interfaz web que se ha desarrollado un método por el cual podamos ver la información que nos presenta en las etiquetas, pestañas y demás en varios idiomas como el español y demás. Esto no es una necesidad de primera mano y se decidió dejarlo como un posible añadido debido a que el usuario potencial de esta aplicación es angloparlante.

Control de acceso

El servicio es accesible para cualquier persona en estos momentos y puede realizar cualquier acción de las que se ha hablado a lo largo del contenido de este documento. No obstante, pensamos que en un futuro se podría bien restringir la aplicación de filtros de búsqueda y dejar que las consultas a la base de datos sin parámetros, es decir, a la base de datos completa, sea sin necesidad de registro.

Controlando la existencia de los usuarios también podríamos saber qué personas y de qué manera hacen uso de la aplicación no solo gracias a la información que nos da el fichero log que ya está implementado, sino a lo mejor podríamos intentar también conocer el tipo de usuario (profesor, alumno, investigador, etc) que accede al servicio y desde dónde.

Panel de administración

Como último posible añadido futuro está la implementación de un segundo servicio complementario al que trata este documento. Un panel de administración mediante el cual los usuarios permitidos (mediante control de acceso con usuario y contraseña) puedan modificar algunos de los registros de la base de datos de manera

cómoda y rápidamente. Esto se ha planteado como último trabajo futuro debido al volumen de trabajo que esto requiere y la imposibilidad de llevar a cabo ambos proyectos en el tiempo que puede durar un Trabajo de Fin de Título.

Bibliografía

Recursos consultados sobre LinQ

- Documentación sobre Dynamic LinQ
    - https://github.com/kahanu/System.Linq.Dynamic/wiki
- Repositorio de Dynamic LinQ
    - https://github.com/kahanu/System.Linq.Dynamic
- Documentación sobre expresiones dinámicas de LinQ
    - https://github.com/StefH/System.Linq.Dynamic.Core/wiki/Dynamic-Expressions
- Cuestiones de funcionamiento de Predicate Builder
    - https://stackoverflow.com/questions/11490893/how-does-predicatebuilder-work
- Consultas con Dynamic LinQ mediante Predicate Builder
    - https://www.c-sharpcorner.com/UploadFile/c42694/dynamic-query-in-linq-using-predicate-builder/
- Información práctica sobre LinQ
    - http://dotnetpattern.com/LINQ-tutorials
- Documentación sobre LinQKit
    - http://www.albahari.com/nutshell/linqkit.aspx
- Repositorio de LinQKit
    - https://github.com/scottksmith95/LINQKit

Descarga de ficheros

- Consulta para la descarga de ficheros PDF en el cliente

    - https://stackoverflow.com/questions/4644506/code-to-download-pdf-file-in-c-sharp

- Formas de descargar un objeto como fichero CSV en el sistema cliente
    - https://social.msdn.microsoft.com/Forums/vstudio/en-US/9c412fb2-a352-43b2-9443-e699aa1ed28e/how-to-save-arraylist-of-object-class-to-csv-file?forum=csharpgeneral

Información sobre ficheros CSV

- Documentación de la librería utilizada para la generación de ficheros CSV
    - https://joshclose.github.io/CsvHelper/getting-started
- Repositorio de la librería
    - https://github.com/JoshClose/CsvHelper
- Estándar de ficheros CSV para generar los ficheros
    - https://tools.ietf.org/html/rfc4180
    - http://www.creativyst.com/Doc/Articles/CSV/CSV01.htm

Información sobre la librería de generación de PDF

- Librería usada para la generación de PDFs
    - https://itextpdf.com/es/products/itext-5-legacy
- Creación de listas dentro de la librería iText
    - https://kodejava.org/how-do-i-create-a-list-in-itext/
- Gestión de tablas con iText
    - https://www.aspforums.net/Threads/778493/Place-Align-two-tables-side-by-side-using-iTextSharp-in-C-and-VBNet/

Consultas realizadas sobre el lenguaje de programación

- Cambiar FAVICON de una aplicación de webforms

    - https://mariusschulz.com/blog/working-with-favicons-in-asp-net-mvc-applications-and-visual-studio

- Convertir listas a tablas de datos
    - https://social.msdn.microsoft.com/Forums/vstudio/en-US/6ffcb247-77fb-40b4-bcba-08ba377ab9db/converting-a-list-to-datatable?forum=csharpgeneral
- Mecanismos para ordenar tablas de datos y listas
    - https://techbrij.com/anonymous-type-linq-gridview-sorting-asp-net

Creación y gestión de la interfaz

- Librería de selectores múltiples
    - https://developer.snapappointments.com/bootstrap-select/
- Librería de interruptores estilizados
    - https://www.bootstraptoggle.com/
- Dejar marcados valores en un listbox de c#
    - https://asp-net-example.blogspot.com/2013/12/aspnet-listbox-set-selected-items.html