UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistema de Gestión de Corpus web versión 4: Consulta, análisis y gestión de corpus lingüísticos mediante un sistema web

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A N

Gabriela Aideé Castillo Del Valle

Marco Antonio Molina Medina

DIRECTOR DE TESIS

M. en C. Gabriel Castillo Hernández

Ciudad Universitaria, Cd. Mx., 2022

Dedicatorias

Dedico esta tesis a mi madre por darme el regalo de la vida, por ser mi fuente de fortaleza, por siempre estar a mi lado y por ser la guía que me hizo llegar a este punto de mi carrera profesional. A mi padre por el ímpetu que has inyectado en mí, por ser mi ejemplo a seguir y por enseñarme que sin importar lo difícil que sea el camino, debo hacer un esfuerzo para no quedar a treinta metros de mis metas. A mi hermano, de quien estoy muy orgullosa, por ser la luz que ilumina mis días y el que me motiva para seguir creciendo. Y a mis profesores Jorge Campos, Patricia Del Valle, Gabriel Castillo, Julio De León, Rocío Aldeco y Eduardo Ortega cuya vocación por la enseñanza me hicieron amar cada aspecto de la ingeniería en computación y por mostrarme los valores y actitudes que hacen a un profesionista destacable. Gracias a sus consejos y ayuda, hoy son los artífices de la culminación de mi licenciatura, este trabajo es para ustedes.

Gabriela Aideé Castillo Del Valle

A mi madre por el amor, esfuerzo y paciencia que me dio a lo largo de mi formación personal y académica, gracias por siempre estar conmigo. A mi padre por ser mi modelo a seguir en mi vida profesional y deportiva, gracias por tus consejos y orientación. A mis hermanos, familia, profesores, sinodales, amigos, compañeros y universidad. Finalmente, a la familia Castillo Del Valle, por su amistad y enseñanza, no solo durante el desarrollo del presente trabajo, sino de toda mi carrera universitaria. A todos les agradezco su apoyo y con cariño, dedico mi contribución al presente trabajo.

Marco Antonio Molina Medina

Índice

Capítulo 1. Introducción

En esta tesis se presenta el desarrollo de un nuevo sistema de gestión de corpus lingüísticos desarrollado en el Grupo de Ingeniería Lingüística (GIL) implementado patrones de diseño y herramientas para el desarrollo de software comúnmente utilizadas en la actualidad a través de un marco de trabajo ágil. El nuevo sistema busca solventar la necesidad de unificar las funciones de los actuales gestores de corpus en el GIL, resolviendo las deficiencias y enriqueciendo los servicios y productividad de estos, pues permitirá subir y compartir colecciones de textos digitales contribuyendo al estudio e investigación de la lingüística de los corpus, en donde se busca como parte de los objetivos a largo plazo, ser una herramienta utilizada entre diferentes colaboradores alrededor del mundo. A continuación, se describe la fundamentación del trabajo realizado. En el capítulo 1 se presenta la introducción, antecedentes, justificación, alcance del sistema y el objetivo de la tesis. En el capítulo 2 se encuentran las bases teóricas del trabajo realizado, mientras que en los capítulos 3 y 4 se muestran los resultados del proceso de análisis. En el capítulo 5 trata sobre el diseño general del sistema y el desarrollo se presentan en el capítulo 6 y 7. Las pruebas finales se muestran en el capítulo 8 y las conclusiones en el último capítulo.

1.1 Antecedentes

La Lingüística Computacional, también conocida como ingeniería lingüística, tuvo sus inicios a principios de 1950 con la introducción de la Computación y la Inteligencia Artificial. Su objeto de investigación es la representación computacional del conocimiento lingüístico y sus procesos, es decir, es el estudio del lenguaje desde una perspectiva computacional creando modelos de cómputo para el análisis de distintos fenómenos lingüísticos y en cuyo proceso participan lingüistas, computologos, psicólogos cognoscitivos, entre otros.

Esta disciplina tiene diversas especialidades, teorías y metodologías, entre las más prominentes se encuentran el reconocimiento del habla y síntesis de voz, la traducción automática del lenguaje natural, el procesamiento del lenguaje natural y la lingüística de corpus.

La lingüística de corpus se enfoca a la creación de recursos lingüísticos, agrupados bajo un conjunto de criterios mínimos, para el estudio uno o más ni-veles de representación lingüística; por ejemplo, se pueden usar con el fin de servir como muestra representativa de una lengua.

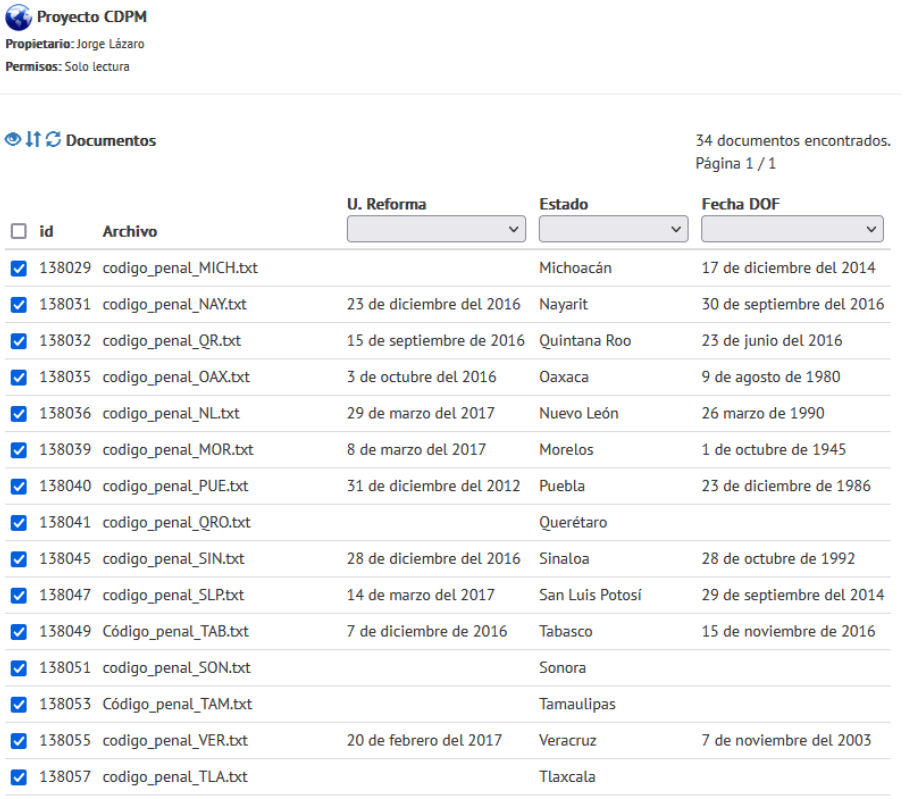
En cuanto a la definición del término corpus, autores como Lemnitzer y Zinsmeister (2006/2010, p. 40) señalan que un corpus es una recopilación de expresiones escritas u orales en una o varias lenguas. En donde los datos pueden ser digitalizados, es decir, almacenados y procesados por una computadora.

Los corpus están formados por expresiones y de metadatos. Estos últimos representan fuentes de información que permiten contextualizar aspectos del corpus o de los datos obtenidos, por ejemplo, la fecha y/o lugar de publicación, datos del autor, editorial, descripción, título, lengua, entre otros.

El desarrollo de corpus exige procedimientos y criterios rigurosos tomando en cuenta su magnitud, recolección y organización, de manera que sean confiables y apropiados para las tareas de interés. Es en este sentido, el desarrollo de herramientas computacionales como los corpus informatizados han permitido la ampliación y mejora del análisis lingüístico al generar resultados más exactos.

Actualmente en el desarrollo de los gestores de corpus se busca crear aplicaciones especializadas que permitan a los usuarios cargar archivos de texto y ejecutar consultas. Los corpus informatizados son repositorios donde se pueden añadir o eliminar textos de una colección,

anotar metadatos en los documentos, utilizar funciones y aplicaciones para analizar los textos. En la Fig. 1 se muestra como ejemplo de un corpus informatizado, el corpus del "Derecho Penal Mexicano"[1]



Proyecto CDPM
Propietario: Jorge Lázaro
Permisos: Solo lectura

Documentos

34 documentos encontrados.
Página 1 / 1

id	Archivo	U. Reforma	Estado	Fecha DOF
138029	codigo_penal_MICH.txt		Michoacán	17 de diciembre del 2014
138031	codigo_penal_NAY.txt	23 de diciembre del 2016	Nayarit	30 de septiembre del 2016
138032	codigo_penal_QR.txt	15 de septiembre de 2016	Quintana Roo	23 de junio del 2016
138035	codigo_penal_OAX.txt	3 de octubre del 2016	Oaxaca	9 de agosto de 1980
138036	codigo_penal_NL.txt	29 de marzo del 2017	Nuevo León	26 marzo de 1990
138039	codigo_penal_MOR.txt	8 de marzo del 2017	Morelos	1 de octubre de 1945
138040	codigo_penal_PUE.txt	31 de diciembre del 2012	Puebla	23 de diciembre de 1986
138041	codigo_penal_QRO.txt		Querétaro	
138045	codigo_penal_SIN.txt	28 de diciembre del 2016	Sinaloa	28 de octubre de 1992
138047	codigo_penal_SLP.txt	14 de marzo del 2017	San Luis Potosí	29 de septiembre del 2014
138049	Código_penal_TAB.txt	7 de diciembre de 2016	Tabasco	15 de noviembre de 2016
138051	codigo_penal_SON.txt		Sonora	
138053	Código_penal_TAM.txt		Tamaulipas	
138055	codigo_penal_VER.txt	20 de febrero del 2017	Veracruz	7 de noviembre del 2003
138057	codigo_penal_TLA.txt		Tlaxcala	

Figura 1: Corpus informatizado del Derecho Penal Mexicano

Algunas de las ventajas de informatizar un corpus son:

1. La manipulación de la información es más sencilla y se obtienen resultados más precisos.
2. Al ser procesados por recursos computacionales, la velocidad de procesamiento es mayor.
3. El costo de acceso disminuye, esto se refiere a que, si son almacenados en un sistema web, los textos pueden ser recuperados y manipulados al instante, facilitando la

[1] Universidad Autónoma de Baja California, Corpus del Habla de Baja California. http://www.corpus.unam.mx/corhbc, 24 de mayo de 2022.

transferencia de datos y aumentando la posibilidad de compartir información con investigadores y personas interesadas en el tema.

1.2 Justificación

El GIL de la UNAM, en el 2015 desarrolló su primer Sistema de Gestión de Corpus Lingüísticos llamado "GECO" [www.corpus.unam.mx/geco] con el fin de unificar el registro de las compilaciones de muestras documentales de corpus comparables[2] y poner a disposición de la comunidad la tecnología de búsqueda de los distintos corpus que puedan apoyar en la investigación del análisis del lenguaje. Posteriormente, se creó un segundo sistema de administración de corpus llamado "GECO3" [www.geco.unam.mx] enfocado en el procesamiento de corpus paralelos[3].

El problema reside en que ambos sistemas presentan ciertas limitantes en usabilidad, control, procesamiento y experiencia de usuario. Los sistemas no cuentan con documentación, ni el mantenimiento necesario, lo que provoca errores durante la usabilidad y mantenimiento de los sistemas.

La nueva versión de GECO, desarrollada en esta tesis, plantea la necesidad de crear un nuevo sistema considerando las funcionalidades de los dos sistemas antes mencionados, implementando una reestructuración total de la plataforma web, haciendo uso de tecnologías que permitan crear un sistema integral que sea escalable, multiplataforma, pues se requiere que el sistema sea compatible con todo tipo de dispositivos, de modo que se pueda visualizar en un smartphone, tablet o pc. Se busca que el sistema sea escalable porque el GIL tiene previsto un

[2] Instituto Cervantes. (s. f.). Tipos de corpus. CVC. Biblioteca fraseológica y paremiológica. Recuperado 24 de mayo de 2022, de https://cvc.cervantes.es/lengua/biblioteca_fraseologica/n6_conde/evaristo_02.html

[3] Hallebeek, J. (s. f.). El Corpus paralelo. Universidad de Nijmegen, Departamento de Español. Recuperado 24 de mayo de 2022, de http://www.corpus.unam.mx/cursocorpus/24-articulo5.pdf

aumento en el volumen de operaciones de análisis lingüístico y de almacenamiento de corpus, de modo que se necesita que el sistema pueda crecer fluidamente sin deteriorar su calidad.

1.3 Objetivo

Elaborar un sistema integral web amigable y usable, llamado Sistema de Gestión de corpus Web Versión 4 (GECO4.0), para unificar la funcionalidad de los sistemas de Gestión de corpus y Gestión de corpus Paralelos para ponerlo a disposición de estudiantes e investigadores.

1.4 Alcance del sistema

Con GECO4, se pretende que la manipulación de la información de los corpus se realice de manera sencilla, permitiendo la generación de estadísticas, así como una mayor precisión de los análisis haciendo uso de las aplicaciones de búsquedas de concordancias, por lema, por palabra normalizada o por coincidencia exacta.

También el usuario tendrá la posibilidad de crear metadatos personalizados asociados a cada documento, para posteriormente crear filtros de búsqueda sobre dichos metadatos. Finalmente, se tendrá la posibilidad de compartir la información con otros usuarios mediante colaboraciones entre estudiantes e investigadores.

El sistema GECO4.0 permitirá a los usuarios acceder desde diferentes dispositivos electrónicos como teléfonos móviles, computadoras o tabletas electrónicas, donde el usuario podrá visualizar y cargar distintas compilaciones de corpus y corpus paralelos y efectuar análisis del lenguaje natural con las aplicaciones embebidas en el sistema. El sistema está concebido para desarrollarse en varias etapas, la primera de ellas consiste en la sistematización y administración de los documentos y los grupos de trabajo, la segunda será la adaptación y desarrollo de algoritmos de explotación de la información y la tercera corresponde al desarrollo de gestores de corpus paralelos.

1.4.1 Alcance de la tesis

Las evaluaciones del funcionamiento de los dos sistemas anteriores (GECO Y GECO3) se hicieron previo al desarrollo de esta tesis, por lo que no se incluyen en el presente trabajo.

Esta tesis, presenta el desarrollo de la primera etapa del sistema, la cuál está conformada por:

- * Creación, visualización y administración de corpus.
- * Administración de grupos de trabajo para cada corpus.
- * Sistematización y administración de documentos para cada corpus.

Como se mostrará más adelante, se empleó Django como plataforma de desarrollo. Django es un framework ampliamente usado, compañías tales como Instagram, The New York Times, el Washingthon Post, Pintrest o national Geographic lo emplean en sus plataformas de internet. Django tiene de manera nativa integrados mecanismos de seguridad para protección de las bases de datos, formularios y código javascript (previniendo la inyección de código malicioso y los accesos no autorizados). Finalmente, Django es la plataforma oficial de desarrollo del GIL para sitios web.

En esta tesis los trabajo previos a la liberación formal (manuales de procedimientos en caso de fallas, evaluación completa de la seguridad del sistema, prevención de ataques, evaluación de las cargas y tráfico, etc) no han sido tomados en cuenta, pues está tesis solo se enfoca al desarrollo de un parte del sistema completo (faltarían los módulos de explotación, publicación de portales de los corpus -que ya se encuentra en desarrollo-, etc). Sin embargo, si se ha puesto cierto cuidado en el tema, en particular en la documentación técnica del sistema y del código, desarrollo seguro y prevención de inyección de código, accesos no autorizados, etc.

Capítulo 2. Fundamentos de Software

Para el diseño de un sistema robusto, flexible, mantenible y escalable se deben de conocer y seguir las buenas prácticas que se establecen en los principios de modelado de software, pues nos darán un punto de partida para establecer la arquitectura, el lenguaje de programación y el modelo de datos que se adapte a la solución requerida por el usuario. En este capítulo se explican brevemente los conceptos teóricos de los principios de modelado de software que se emplearon en el desarrollo del sistema “GECO4”.

2.1 Ingeniería de software

El término de "ingeniería de software" se propuso en 1969 en una conferencia de la OTAN para discutir lo que entonces se llamó la "crisis del software", en donde se plantearon los problemas de desarrollo de software en sistemas grandes y complejos, pues se observó que los sistemas no brindaban la funcionalidad que necesitaban sus usuarios, costaban más de lo esperado y no eran confiables. Derivado de este suceso, en los últimos treinta años del siglo pasado y aún durante los primeros años de este siglo, se desarrollaron nuevas técnicas y métodos de ingeniería de software, como lo fueron la programación estructurada, encapsulación de información y el análisis y diseño orientado a objetos, así mismo se establecieron herramientas y notaciones estándares que ahora se consideran buenas prácticas en el desarrollo de programas de software.

La ingeniería de software tiene por objetivo respaldar el desarrollo de software profesional, incluyendo técnicas que apoyan la especificación, el diseño y la evolución de los sistemas. En este contexto, las principales actividades de un ingeniero de software son las de toma de requerimientos o especificación de software, diseño, desarrollo, validación y supervisión durante el proceso de evolución del software.

Un buen producto de software debe de ofrecer la funcionalidad y el rendimiento requeridos por el usuario, debe implementar de forma correcta los requerimientos funcionales y no funcionales, debe ser mantenible, escalable, confiable y utilizable. Para lograrlo se sigue el "Ciclo de vida de los sistemas de información", el cuál es un proceso iterativo, es decir, conforme vayan surgiendo nuevas necesidades de negocio se incorporan mejoras siguiendo el mismo proceso. Este ciclo se conforma principalmente de dos fases:

1. Fase de desarrollo: Involucra la creación del concepto, requerimientos, diseño, implementación, pruebas y liberación o despliegue.
2. Fase de evolución: Es el mantenimiento y mejoras del producto.

2.2 Metodología Kanban

La metodología que seguimos para el desarrollo del sistema GECO4 se conoce como Kanban[4], que es una metodología ágil de gestión de proyectos que busca conseguir un proceso productivo, organizado y eficiente.

Primero se hace un análisis para obtener y entender todos los elementos que se necesitan para el desarrollo del producto de software desde su pedido hasta su entrega. Después, los diferentes módulos del sistema se muestran en un tablero con columnas que representan las distintas etapas del desarrollo como se muestra en la Fig. 2. Cada módulo se pone a modo de "tarjeta" y también se establece al responsable de la tarea, la fecha de entrega y cualquier etiqueta relevante como la prioridad o el tipo de tarea.

[4] Kanbanize. (s. f.). *Qué es Kanban: Definición, Características y Ventajas*. Kanban Software for Agile Project Management. Recuperado 15 de mayo de 2022, de https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-kanban

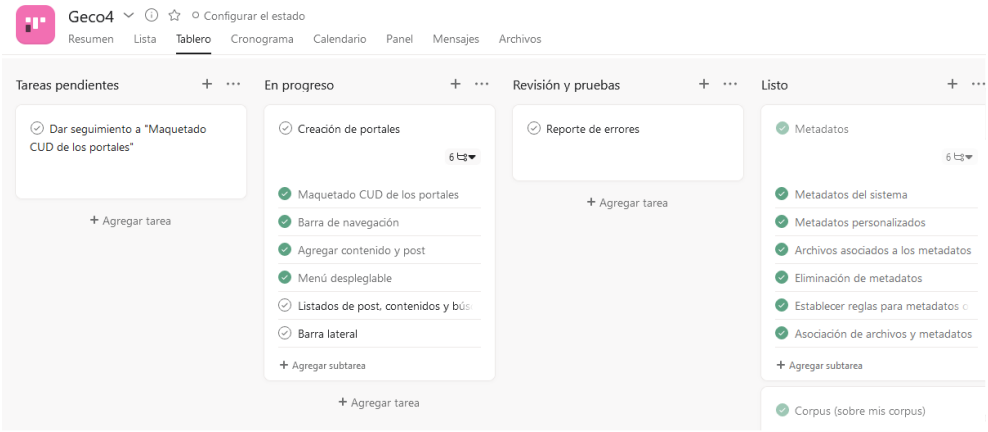


Figura 2: Muestra del tablero Kanban empleado para la gestión del desarrollo de GECO4

2.3 Paradigmas de programación

Un paradigma es una escuela de pensamiento o modelo con características, marcos, patrones y estilos distintos que sirven para resolver un problema. El campo de la programación un paradigma representa o está formado por un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas computacionales. Entender los patrones de diseño, le permite al desarrollador saber qué lenguaje es el más adecuado para cada tipo de escenario y cada problema a resolver, pues como lo dicen autores como Louden y Lambert (2011):

"*Así como la forma en que nos comunicamos influye en lo que pensamos y viceversa, la forma en que programamos influye en lo que entendemos por computación y viceversa*".

2.3.1 Paradigma orientado a objetos

Aguilar (1998, p.18) define a la programación orientada a objetos como un paradigma cuyas entidades centrales son los objetos, que son tipos de datos que encapsulan con el mismo nombre estructuras de datos y las operaciones o algoritmos que manipulan esos datos. En donde, estos objetos están ligados o se comunican mediante mensajes para la solución de problemas. Dicho

de otra forma, un objeto representa a una entidad que tiene atributos específicos, las propiedades, y unas formas de operar sobre ellos, los métodos. (Sierra, 2003, p. 24)

El paradigma orientado a objetos es muy empleado en la programación, porque por sus características permiten modelar los objetos del mundo real y por lo mismo diseñar soluciones que se necesitan en los problemas del mundo actual. Sierra (2003, p.24) considera que hay cuatro características principales que definen a la programación orientada a objetos, las cuáles son:

- Abstracción: Es la capacidad de obtener, generalizar y aislar los aspectos que le permitan al desarrollador obtener una visión global del problema para poder identificar el comportamiento fundamental o rol que tendrá cada objeto dentro de la aplicación.
- Encapsulamiento: Se refiere a ocultar todos los datos abstractos de un objeto, haciendo ver a un objeto como una caja negra. Permitiendo al programador manipularlos como unidades básicas, permaneciendo oculta su estructura interna para evitar cualquier tipo de violación de integridad de la entidad
- Herencia: Es la disposición de métodos y propiedades de una clase general a una más especializada, de esta forma se garantiza la reutilización de código y proporciona una forma más rápida y cómoda de extender la funcionalidad de una clase.
- Polimorfismo: Como su nombre lo dice es la característica de implementar de diversas formas un mismo método para que reaccione de manera diferente de acuerdo con las entradas que reciba.

2.4 Patrones de diseño

Un patrón de diseño es una solución de buenas prácticas que documenta y explica cómo una solución general se aplica para resolver un problema particular de diseño de software. Gamma, et al. (1995, p. 3) proponen que un patrón se compone de cuatro elementos esenciales:

1. El nombre del patrón, que permite identificar principales características, problemas o soluciones.
2. El problema, que permite identificar el contexto y puede representar clases, objetos o diferente tipo de estructuras.
3. La solución, que permite identificar la relación, responsabilidad y colaboración entre los diferentes elementos que componen el diseño, y
4. Las consecuencias, que permite identificar las ventajas y desventajas de los resultados de aplicar el patrón.

Un patrón de diseño reutiliza y aplica aspectos clave como los roles, colaboraciones y relaciones entre las diferentes clases e instancias participantes.

Uno de los patrones más conocidos y aplicados en el desarrollo de software es el patrón Modelo Vista Controlador. En este patrón hay una clara separación de las tareas de presentación de información (la vista), los datos (el modelo) y las operaciones que se deben realizar sobre los datos para generar las vistas (el controlador). En esta tesis emplearemos un patrón modificado, basado en este patrón.

2.4.1 Modelo Vista Template

En GECO4 se emplea como plataforma de desarrollo el framework de código abierto Django, basado en Python y desarrollado en un principio para la gestión de páginas web alrededor de 2003. Actualmente, Django es uno de los frameworks de desarrollo de sitios web más populares.

Django[5] redefine el patrón conocido como MVC (Modelo-Vista-Controlador) y crea un patrón llamado MVT [6] (Modelo-Vista-Template). En donde el modelo se encarga de gestionar la base de datos, la vista se utiliza para ejecutar la lógica de negocio e interactuar con un modelo para transportar datos y presentarlos en el template adecuado, finalmente el template gestiona completamente la parte de la interfaz de usuario.

De modo que un sistema desarrollado con framework Django funciona de la siguiente forma: Cuando el usuario accede a una URL, (1) se obtiene el mapa de URLs, que es un archivo en donde se describe la asociación que tiene cada ruta con su vista, una vez obtenida la vista correspondiente (2) se revisa si se necesita o no algún dato del modelo, en tal caso se hará una consulta a la base de datos para obtenerlo, (3) ya que se obtienen los datos necesarios la vista los manda al template (4) y finalmente, el template se encarga de renderizar e integrar los datos dinámicos recuperados del modelo a la página que solicitó el usuario como se muestra en la Fig. 3.

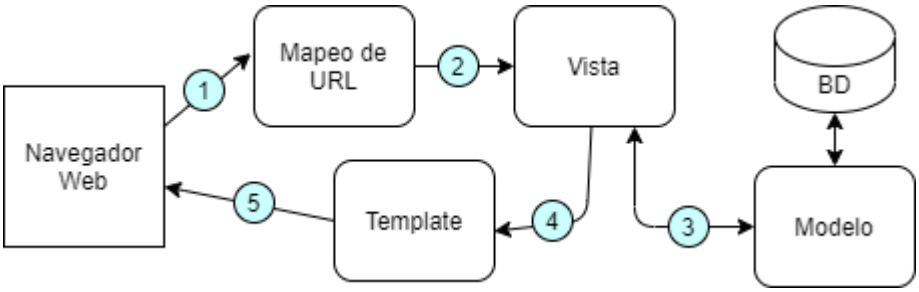


Figura 3: Diagrama del funcionamiento del patrón de diseño MVT

Implementar el patrón Modelo Vista Template de Django permite el desarrollo de una aplicación de alto rendimiento, flexible y ágil. El uso de bases de datos y modelos en el framework Django

[5] Django Project. (s. f.). *The web framework for perfectionists with deadlines | Django*. Recuperado 15 de mayo de 2022, de https://www.djangoproject.com/

[6] Javatpoint. (s. f.). *Django MVT*. Recuperado 15 de mayo de 2022, de https://www.javatpoint.com/django-mvt

ofrece una serie de herramientas que permite definir diferentes tipos de campos, relaciones y comportamientos esenciales para la información que se está almacenando, aquí podemos destacar la funcionalidad de crear métodos personalizados y predefinidos para un modelo, además de la implementación de diferentes tipos de herencia de modelos similar a la que se utiliza en las clases.

Las vistas son las responsables de la lógica del sistema y ofrecen una extensa variedad de ventajas y servicios que van desde un amigable mapeo de URLs, devolución de errores, funciones, atajos, decoradores, funciones asíncronas, carga de archivos, entre otros servicios. Las vistas en Django pueden ser tan simples o complejas dependiendo del trabajo o función que se espere que realice, además de que se puede implementar diferentes tipos de librerías que el lenguaje de programación Python ofrece.

Finalmente, el sistema dinámico de plantillas HTML de Django ofrece ventajas como un sistema fácil de renderizar y reutilizable. Entre las herramientas que tiene el lenguaje de plantillas podemos destacar el uso de variables, etiquetas, filtros, herencia y otras funciones que facilita y optimiza el flujo de información.

Hemos decido usar Django ya que es uno de los frameworks para desarrollar aplicaciones full stack más importantes en la actualidad, ofreciendo funciones como autenticación, motor de plantillas, enrutamiento de url, mapeador relacional de objetos y migraciones de esquemas de bases de datos. Además, para tomar de decisión implementar Django como solución al desarrollo del proyecto se tomó en cuenta que las aplicaciones de análisis de textos que se incorporarán en futuras versiones del proyecto serán desarrolladas en lenguaje de programación Python, mismo que utiliza Django y que facilita su integración al sistema.

2.5 Herramientas para el desarrollo de software

Las herramientas de software cumplen el objetivo de facilitar, optimizar y mejorar el desempeño del trabajo y dependiendo de la etapa en la que se esté trabajando en el ciclo del software se usan distintas herramientas de desarrollo.

2.5.1 Control de versiones (Git y GitHub)

Se le llama control de versiones a la práctica de rastrear y gestionar los cambios en el código de software. Los sistemas de control de versiones son herramientas de software que ayudan a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo y a la distribución eficaz de las versiones de este.

Git[7] es un es un sistema de control de versiones distribuido diseñado por Linus Torvalds, que guarda diferentes versiones de uno o varios archivos y que permite que cualquier versión sea recuperable. Se utiliza en la administración de proyectos de software porque facilita el registro y comparación de diferentes versiones de un archivo, se incluye información sobre qué fue lo que cambió, quién, cuándo y porqué se hizo. El flujo de trabajo de git se divide en tres secciones como se muestra en la Fig. 4.

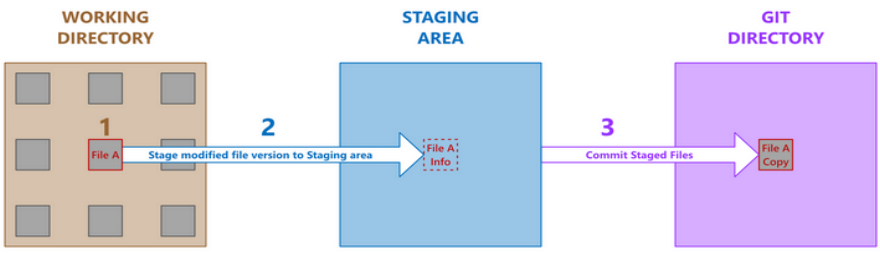


Figura 4: Flujo de trabajo básico de git

[7] Git. (s. f.). *Git*. Recuperado 15 de mayo de 2022, de https://git-scm.com/

Dentro de los sistemas de control de versiones existe una herramienta llamada GitHub que es una plataforma de alojamiento de Microsoft, basada en el sistema de control de versiones distribuida de Git, que se utiliza para crear repositorios de código y guardarlos en la nube. En la Fig. 5 se muestra el tablero de commits de github de GECO4.

Figura 5: Muestra del control de versiones en GitHub para el desarrollo de GECO4

2.5.2 Lenguaje unificado de modelado (UML)

El modelado de software es el primer paso antes de desarrollar cualquier tipo de sistema y se basa en la creación de diagramas que explican el funcionamiento del software, para ello se emplea el lenguaje UML, lenguaje de modelado creado en 1995 por la compañía Rational Software Corporation bajo la supervisión de Grady Booch, James Raumbagh e Ivar Jacobson. UML[8] permite crear modelos visuales orientados a procesos de negocio o para la arquitectura, el diseño y la implementación de sistemas de software complejos, tanto en estructura como en

[8] Lucidchart. (s. f.). *Qué es el lenguaje unificado de modelado (UML)*. Recuperado 15 de mayo de 2022, de https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml

comportamiento. UML se incluye dentro de buenas prácticas para la construcción y documentación de diferentes aspectos del modelado de sistemas de software y de negocios.

Existen tres conceptos de modelado o esquemas especificados por UML, los cuales son:

- **Funcionales:** Son los casos de uso, diagrama de actividades, diagrama de estados, entre otros, que describen la funcionalidad del sistema desde el punto de vista del usuario.
- **De objetos:** Describen la estructura del sistema en términos de objetos, atributos, asociaciones y operaciones.
- **Dinámicos:** Se emplean para describir el comportamiento interno del sistema.

Para ello el lenguaje UML utiliza dos tipos de diagramas: los estructurales, que muestran los niveles de abstracción y estructura estática del sistema, y de comportamiento, que describe el funcionamiento y la interacción de los objetos en el sistema.

2.5.3 JavaScript

JavaScript[9], basado en el estándar ECMAScript (contrario a lo que mucha gente piensa, no es un subconjunto del lenguaje Java), es un lenguaje de programación interpretado, basado en prototipos, multiparadigma y multiplataforma que se utiliza para hacer que las páginas web sean interactivas conectando a los objetos de su entorno para proporcionar control programático sobre ellos.

JavaScript se ejecuta del lado del cliente y del lado del servidor, adoptando comportamientos diferentes, en el lado del cliente extiende el núcleo del lenguaje al proporcionar objetos para

[9] Mozilla.org. (2022, 2 febrero). *JavaScript*. Developer Mozilla. Recuperado 15 de mayo de 2022, de https://developer.mozilla.org/es/docs/Web/JavaScript

controlar un navegador, su modelo de objetos de documento (DOM) y respondiendo a eventos del usuario. Mientras que en el servidor permite que una aplicación se comunique con una base de datos, brinde continuidad de información de una invocación a otra de la aplicación o realice manipulación de archivos.

2.5.4 Framework web Django

Un framework es un marco de trabajo que implementa un conjunto de técnicas y procesos estructurados que facilitan la construcción de aplicaciones. Villalobos, et al. (1995, p. 178-183) define un framework para desarrollo web como una estructura de soporte definida, mientras que en programación lo conceptualiza como un conjunto de funciones o código genérico que realiza tareas frecuentes y comunes. Django es un framework o marco de trabajo web en continuo desarrollo, de código abierto escrito en Python y lanzado en 2005. El sitio oficial web de Django lo define como un marco web de alto nivel que fomenta el desarrollo rápido, con un diseño limpio y pragmático.

Este framework sigue una filosofía de crear aplicaciones con menos código y en menor tiempo posible, aprovechando las cualidades dinámicas de Python, busca fomentar las mejores prácticas separando la lógica de la presentación, es decir el sistema de plantillas tienen el propósito de presentación y nada más, estas a su vez tienen el propósito de desacelerar la redundancia separando y almacenando elementos comunes como encabezado, pie de página, barra de navegación, entre otros, para reutilizar estos elementos. El uso de modelos para encapsular las consultas a la base de datos busca eficientar las sentencias SQL que comúnmente se ejecutan creando una sintaxis concisa que permite ejecutar la menor cantidad de veces posible para satisfacer un requerimiento de información. Las vistas que controlan la lógica son simples y de poco acoplamiento.

Las ventajas de implementar un framework o marco de trabajo como Django son minimizar el tiempo de desarrollo, contar una arquitectura bien estructurada, implementar patrones de diseño, facilitar el desarrollo de pruebas y mantenimiento, así como el desarrollo paralelo en diferentes componentes

2.5.5 Uso de Sphinx para generar documentación automática

A lo largo del tiempo en el desarrollo de software han surgido diferentes herramientas que permiten obtener un producto de calidad que garantice eficiencia, flexibilidad, confiabilidad y seguridad por mencionar algunas de las cualidades, una de estas herramientas es la documentación de código. Sphinx[10] es una herramienta que permite generar documentación de código en diferentes formatos de salida, fue escrita por Georg Brandl y está licenciada bajo la licencia BSD. Ofrece funciones como exportación en HTML, LaTeX, PDF, ePub, Texinfo, páginas de manual y texto sin formato. A través de comentarios en el código fuente, Sphinx permite documentar los módulos, clases, funciones, parámetros, tipos de parámetros, atributos, valores de retorno, entre otros tipos de valores de Python.

La documentación de código permite ayudar a entender cómo funciona el sistema para los usuarios y otros desarrolladores, además permite que el sistema sea un producto escalable, reutilizable y de fácil mantenimiento.

[10] Sphinx. (s. f.). *Overview — Sphinx documentation*. Recuperado 15 de mayo de 2022, de https://www.sphinx-doc.org/en/master/

Capítulo 3. Análisis del problema

Para generar una solución técnica que sea funcional, escalable y que resuelva la problemática del usuario, es necesario hacer un análisis de la situación actual para ser capaces de identificar las áreas de oportunidad y entender los requerimientos que plantea el usuario. Por ese motivo, a continuación, se describe el funcionamiento y la delimitación de los dos sistemas de gestión de corpus del GIL.

3.1 Gestión de Corpus en el GIL

Actualmente en el GIL del Instituto de Ingeniería de la UNAM existen dos sistemas de Gestión de corpus, el primero, llamado GECO [www.corpus.unam.mx/geco] está enfocado en la gestión y procesamiento de corpus comparables, mientras que el segundo, GECO3 [www.geco.unam.mx] está orientado en los corpus paralelos. A continuación, se describe el funcionamiento del primer sistema.

Esta descripción se basa en las Figuras 6 a 12, los números entre paréntesis hacen referencia a secciones dentro de estas Figuras. Lo primero que debe hacer el usuario es registrarse o iniciar sesión según sea el caso (1) como se muestra en la Fig. 6. Una vez iniciada la sesión se muestra un panel con varias opciones “Gestión de documentos y proyectos”, “Aplicaciones” y “Gestión de portales”, para la creación de un nuevo corpus se debe seleccionar la primera opción (2) como se muestra en la Fig. 7. Se redirige al usuario a una nueva página que despliega un pequeño menú lateral izquierdo que muestra las carpetas de documentos y los proyectos o corpus que hay en el sistema, para crear un nuevo corpus se debe de seleccionar uno ya creado (3), seleccionar uno o más documentos (4) y una vez seleccionados se actualiza un menú desplegable que mostrará la opción de crear un nuevo proyecto (5) como se muestra en la Fig. 8.

Figura 6: Página principal del sitio GECO

Figura 7: Se muestra un menú de las acciones que puede realizar el usuario en GECO

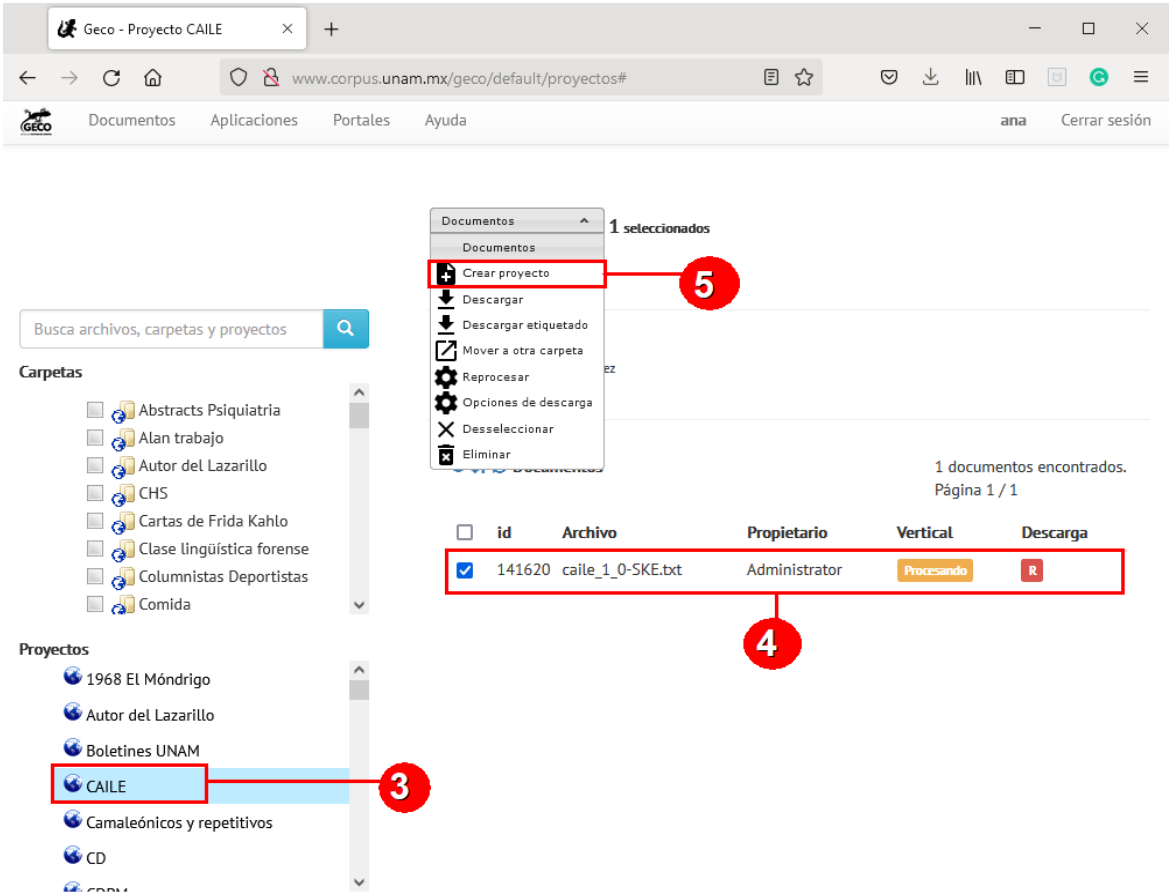


Figura 8: Se muestra la página de "Documentos" donde se puede crear un nuevo corpus en GECO

Por su parte, la funcionalidad del sistema de corpus paralelos GECO3, es de la siguiente: el usuario debe registrarse o iniciar sesión según sea el caso (1) en la Fig. 9. Una vez iniciada la sesión se muestra un menú desplegable con el nombre de "Proyectos" (2) en la Fig. 10, al abrir el menú, hay una opción llamada "Nuevo" (3) en la Fig. 11, que va a redirigir al usuario a una nueva pestaña para que pueda definir las características de su nuevo corpus (4) en la Fig. 12.

Figura 9: Página principal del sitio GECO3

Figura 10: Menú que contiene los proyectos existentes y la creación de nuevos corpus en GECO3

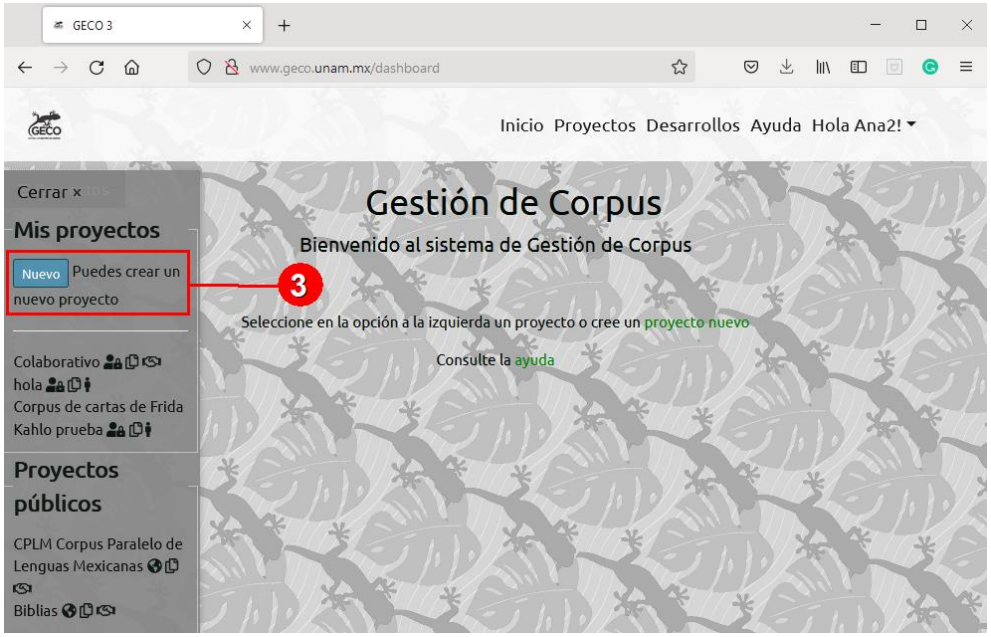


Figura 11: Se muestra la opción para la creación de nuevos corpus en GECO3

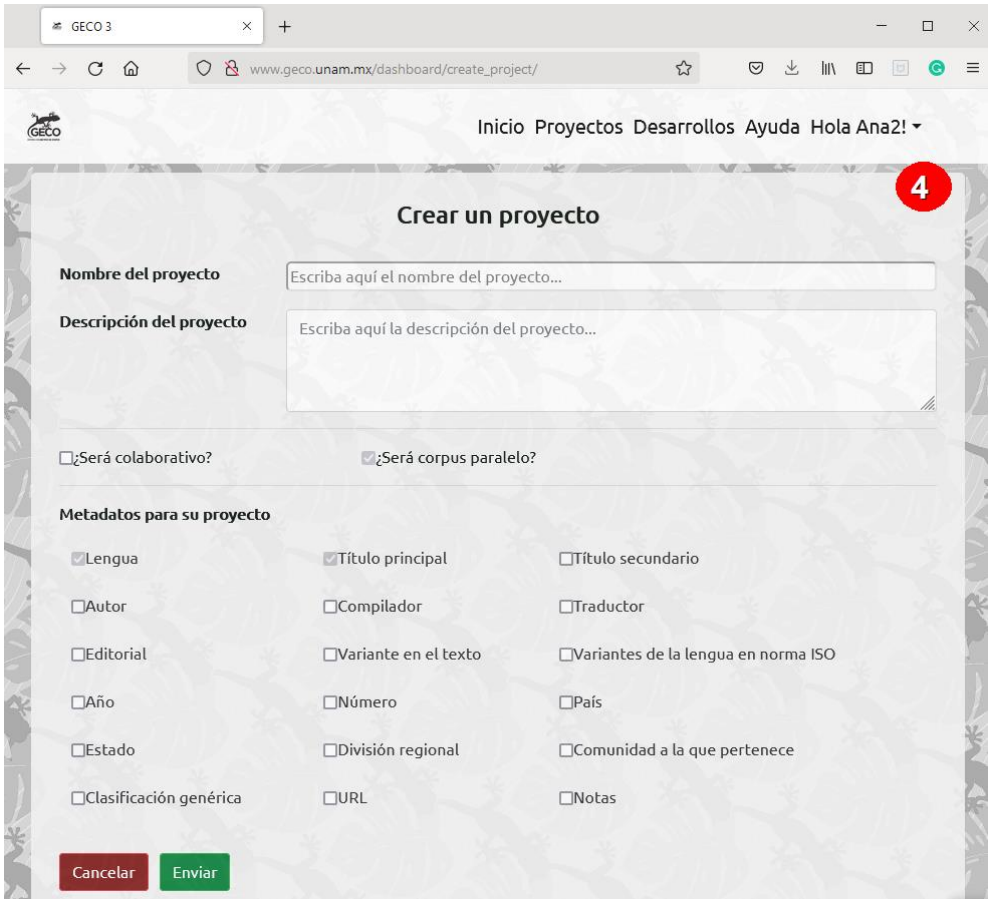


Figura 12: Página de configuración de las características del nuevo corpus en GECO3

3.2 Limitaciones de los sistemas actuales

Una vez descrito el funcionamiento de ambos sistemas, se puede hablar de sus limitaciones[11] que se obtuvieron a partir de los comentarios, experiencias y análisis del equipo de investigadores del GIL:

1. **Usabilidad y experiencia de usuario**

   Aunque ambos sistemas fueron desarrollados por la misma institución y su funcionalidad es casi la misma, no se cuenta con un estándar visual lo resulta en una experiencia confusa para el usuario y eso dificulta el uso de ambas herramientas. De la misma manera, los dos sistemas carecen de diseño centrado en el contenido, es decir, la organización del sitio y la estructura de navegación resulta para el usuario difícil de manejar y en su mayoría de las veces se necesita el apoyo de otro usuario para entender el uso del software, un claro ejemplo de esto es en la creación de un corpus en el sistema GECO donde al ingresar a la "Gestión de documentos y proyectos" no queda claro el proceso que se debe de seguir para crear un corpus, ni es posible identificar los proyectos en donde el usuario colabora o el proceso para la gestión de documentos en un corpus como se muestra en la Fig. 13.

[11] Las evaluaciones del funcionamiento de los dos sistemas anteriores (GECO Y GECO3) se hicieron previo al desarrollo de esta tesis, por lo que no se incluyen en el presente trabajo.

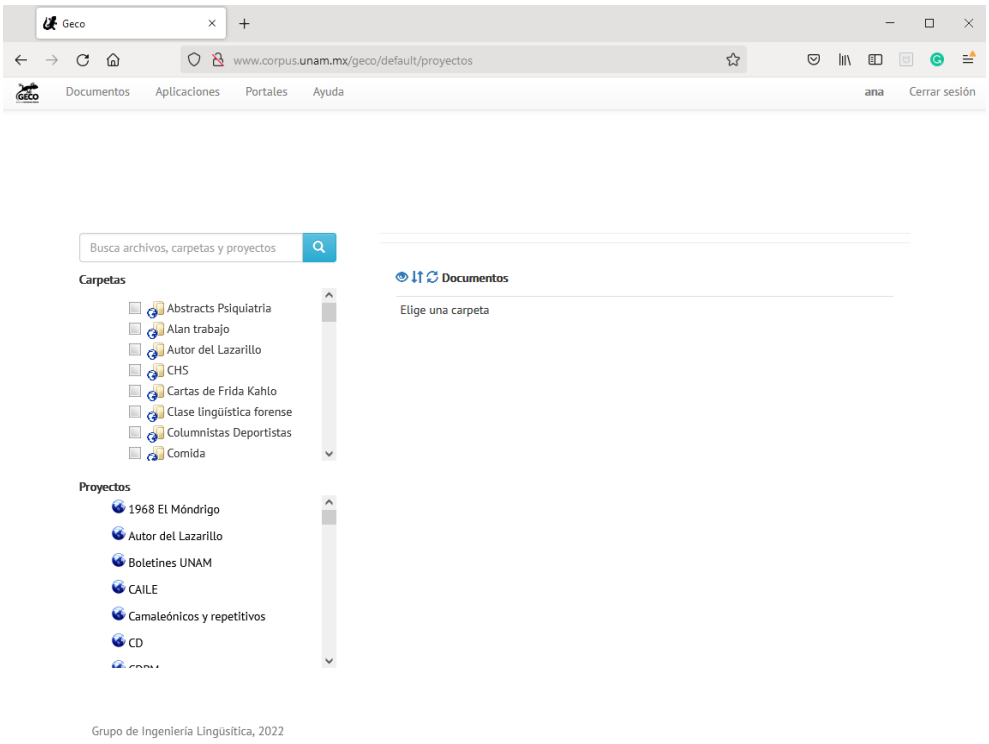


Figura 13: Se muestra la página de "Gestión de documentos y proyectos" en GECO, obsérvese que no es posible identificar fácilmente cómo crear un nuevo corpus

2. Gestión de documentos y corpus

Ninguno de los dos sistemas cuenta con roles de usuario, generando problemas en la gestión de los corpus, pues en GECO cualquier usuario puede realizar diversas acciones sobre los corpus, como editarlos, eliminarlos, agregar archivos, eliminar archivos, agregar colaboradores, entre otras. Mientras que en GECO3 no existe un rol de administrador y no hay forma de gestionar la plataforma.

3. Almacenamiento innecesario de documentos

Tanto GECO como GECO3 ofrecen la posibilidad de subir documentos en formato PDF y los sistemas convierten estos documentos a texto. Como resultado de esta

funcionalidad, se ha identificado que gran cantidad de usuarios crean corpus para hacer uso de esta funcionalidad sin ánimo de realmente conformar un corpus, generando un problema de rendimiento en la base de datos, pues se guardaban una gran cantidad de registros y de documentos que no tenían un uso o una implementación dentro de los corpus.

4. **Documentación y mantenimiento del sistema**

Ninguno de los sistemas cuenta con documentación de su desarrollo, GECO [www.geco.unam.mx], fue desarrollado con una herramienta llamada Odoo versión 8 (actualmente la última versión de este software es Odoo 15), que es una suite de aplicaciones de gestión empresarial que incluye una gama de herramientas para optimizar y rentabilizar los negocios. Su objetivo es englobar en un único software todas las herramientas que necesita una empresa para la operativa en todos los principales departamentos, de modo que se empleó un software desarrollado con un objetivo totalmente distinto para crear GECO y aunque se haya desarrollado con un software licenciado, no se cuenta con la documentación de su implementación.

Por otro lado, el sistema de GECO3 enfocado en corpus paralelos [www.ling.unam.mx] fue desarrollado con HTML no existe documentación del desarrollo ni el contacto de la persona que lo hizo. Por lo que resulta bastante complicado darles a ambos sistemas un mantenimiento constante y esto provoca errores durante la usabilidad del sistema.

5. **Obsolescencia de la plataforma donde están montados los sistemas**

Al no existir documentación de ambos sistemas, la actualización de los sistemas operativos es altamente riesgosa, pues se desconoce qué detalles deben cuidarse, incluso

la migración de los sistemas a un nuevo servidor es ya de por sí un problema que puede resultar irresoluble. GECO opera actualmente en un sistema operativo Suse 12., mientras que GECO3 está montado en un Ubuntu 16, con los consecuentes riesgos de seguridad por obsolescencia.

Capítulo 4. Requerimientos y reglas de negocio

Antes de empezar con el diseño del sistema GECO4, es necesario analizar y establecer requerimientos específicos para saber qué es lo que se va a realizar. En este capítulo presentaremos las reglas de negocio, requerimientos funcionales, roles de usuario y sus casos de uso.

4.1 Reglas de negocio

En este apartado se enuncian las reglas de negocio que el GIL definió para crear el nuevo sistema de gestión de corpus, tomando en cuenta las características principales de GECO y GECO3, de modo que el nuevo sistema permita a los usuarios gestionar de manera eficiente sus recursos digitales, de manera que le permita publicar corpus de manera sencilla y rápida, cumpliendo las reglas de privacidad y colaboración que los usuarios necesitan.

- Dependiendo de la privacidad de los documentos, un corpus se podrá definir como público o privado.
- Siempre que un corpus sea público, los usuarios podrán visualizar sus documentos y revisar su información sin necesidad de tener una cuenta registrada en el sistema de GECO4.
- Para que los usuarios puedan crear un corpus deberán de tener una cuenta registrada en el sistema de GECO4.
- Cuando un corpus es colaborativo, será responsabilidad del propietario de dicho corpus otorgar permisos de colaboración a otros usuarios que se encuentren registrados en el sistema de GECO4.

- Cuando un usuario requiera subir un documento a un corpus, el usuario deberá iniciar sesión en el sistema de GECO4 y contar con permisos de edición en dicho corpus.
- Los usuarios que únicamente requieran convertir uno o más documentos con extensiones “.doc”, “.docx”, “.xls”, “.xlsx”, “.pdf” y “.odt” a un documento “.txt”, sin que dichos documentos se guarden en el sistema, no necesitan tener una cuenta registrada en el sistema de GECO4.

4.2 Requerimientos funcionales y módulos del sistema

En esta sección presentamos los requerimientos funcionales y la definición de los módulos de software que implementarán la funcionalidad mencionada del sistema GECO4.

Módulo de Creación de Corpus

- Para que el usuario pueda crear un nuevo corpus deberá contar con un perfil dentro del sistema GECO4 e iniciar sesión.
- Después deberá llenar un formulario donde se especifiquen las características de este, dicho formulario contiene la siguiente información:
    - Nombre único del corpus
    - Descripción detallada del contenido o del propósito de estudio.
    - Forma de citar el corpus
    - Tipo de publicación: Si es un corpus que lo puede ver el público en general o si solo lo podrán ver los usuarios registrados en el sistema. De modo que se establecen dos opciones: público o privado.
    - Forma de trabajo: Se debe de indicar si es un corpus que admite o no colaboradores.

- Metadatos: El usuario podrá elegir los metadatos predeterminados por el sistema, sin embargo, todos los corpus deberán incluir los metadatos de “Lengua” y “Título principal”. También el usuario podrá establecer si los metadatos son de carácter obligatorio u opcional.
- Metadatos personalizados: El usuario puede definir los metadatos que mejor se adapten a los documentos de su corpus, así como establecer si serán obligatorios u opcionales.
- Tipo de archivos asociados: En algunos casos hay corpus que necesitan de archivos auxiliares para su análisis, como es el caso de los corpus orales, en donde se puede o no contar con el archivo de audio y la transcripción de este. Por ello, el usuario podrá definir si el corpus necesita archivos de audio o de texto adicionales y si son obligatorios u opcionales.

Módulo de Visualización de Corpus

- El sistema deberá contar con un apartado para que cualquier persona pueda ver los corpus que sean públicos del sistema.
- En este mismo módulo se mostrarán los corpus públicos y privados del sistema, cuando el usuario haya iniciado sesión.
- El usuario podrá ver la siguiente información del corpus:
  - Nombre único del corpus
  - Descripción detallada del contenido o del propósito de estudio.
  - Propietario del corpus
  - Fecha de creación
  - Fecha de la última modificación

    - Forma de citar el corpus
    - Tipo de publicación: Si es un corpus que lo puede ver el público en general o si solo lo podrán ver los usuarios registrados en el sistema. De modo que se establecen dos opciones: público o privado.
    - Forma de trabajo: Indica si es un corpus que admite o no colaboradores.
    - Un listado de los documentos y el contenido de cada documento que conforman el corpus.
    - Los metadatos del corpus y de sus documentos
    - Los datos de los colaboradores.
- En este módulo el usuario solamente puede ver la información asociada al corpus, realizar descargas en caso de que esté permitido y utilizar sus aplicaciones para el análisis lingüístico.

Módulo de Gestión de Corpus

- El usuario podrá editar todas las características de sus corpus.
- El usuario tendrá la opción de eliminar sus propios corpus.
- El propietario del corpus puede eliminar a sus colaboradores.
- El propietario del corpus puede cambiar el tipo de colaboración, es decir, si es individual o colaborativo.
- Se pueden agregar a más usuarios a un mismo corpus
- Los usuarios que cuenten con los permisos necesarios podrán subir documentos a los corpus.
- Los usuarios que cuenten con los permisos necesarios podrán eliminar documentos de los corpus.

- Los usuarios que cuenten con los permisos necesarios podrán editar la información de los metadatos de los documentos del corpus.
- Los usuarios que cuenten con los permisos necesarios podrán editar el contenido de los documentos del corpus.
- Las actividades de eliminación, creación y edición de las propiedades de los corpus, eliminación, edición y carga de documentos deberán quedar registradas en una bitácora.

Módulo de Conversión de documentos

- El sistema deberá contar con un apartado para convertir uno o más documentos con extensiones ".doc", ".docx", ".xls", ".xlsx", ".pdf" y ".odt" a un documento ".txt", sin que dichos documentos se guarden en el sistema. Esto se hace para evitar, en la medida de lo posible, que la base de datos guarde información innecesaria.
- Para la conversión de documentos el procedimiento es el siguiente: primero el usuario carga en el sistema el documento que desea convertir y después selecciona la opción de descarga.
- Cualquier persona podrá interactuar con este módulo, es decir, no necesita tener una cuenta en el sistema para poder convertir sus documentos.

En la Fig. 14 se muestra la estructura general y la composición de cada uno de los módulos antes descritos[12].

[12] Para la tesis solamente se desarrollaron los módulos referentes a los corpus comparables, el desarrollo de los módulos de los corpus paralelos se contempla en una segunda etapa de GECO4, Por esa razón en la Fig. 14, los módulos de corpus paralelos se muestran con un color diferente.

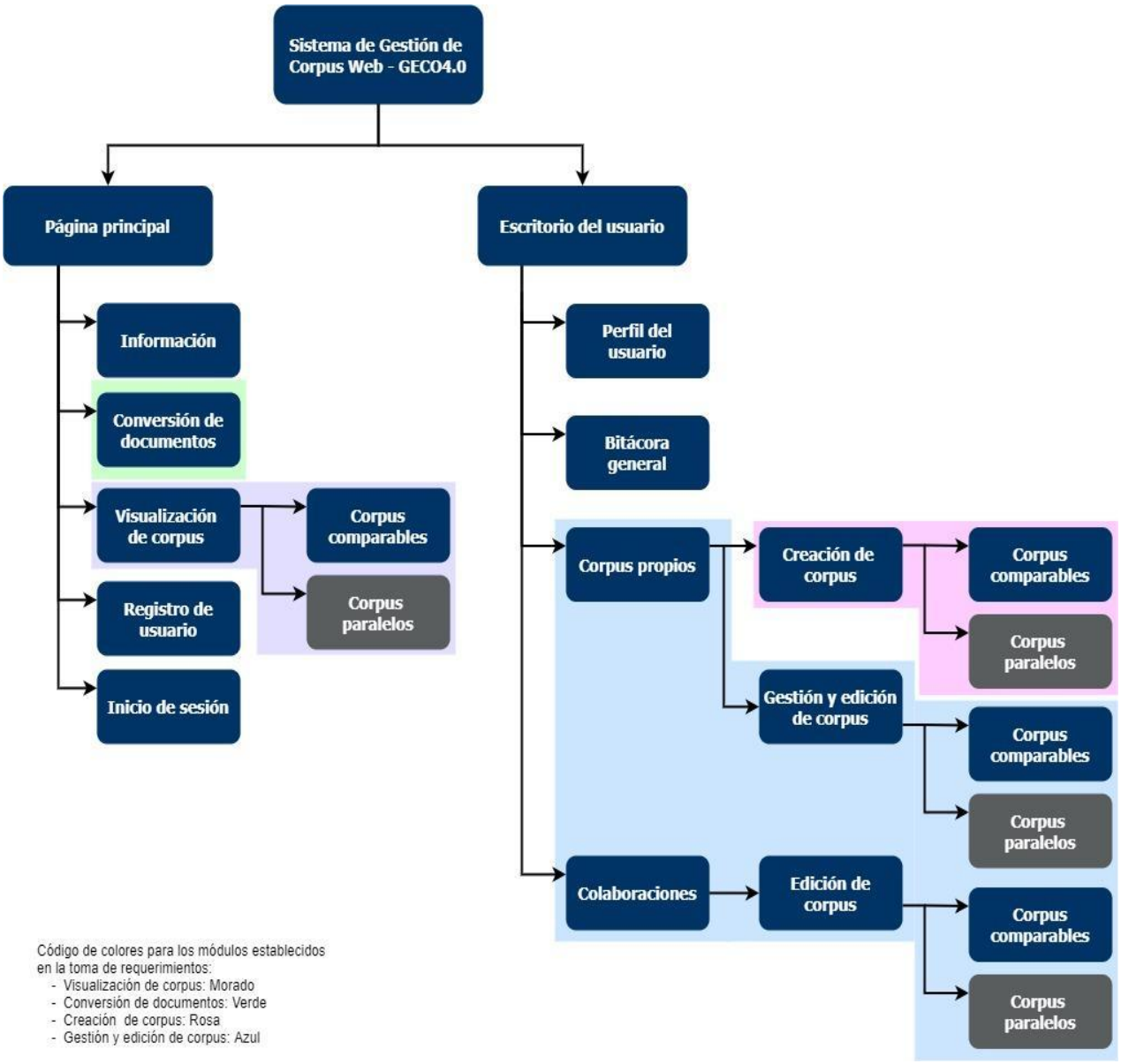


Figura 14: Estructura general del sistema GECO4

4.3 Roles de usuario

Para tener un mejor control sobre las acciones que pueden hacer los usuarios en el sistema, se definieron dos tipos principales de usuarios:

Usuario general

- Es un usuario que no está registrado en el sistema.

- Solo tiene acceso a la “Página principal del sitio” y a todos los submódulos
- Para el módulo de “Visualización de corpus” el usuario general solo podrá ver e interactuar con los corpus que estén registrados como públicos.

Usuario registrado

- Es un usuario que, como su nombre lo dice, está registrado en el sistema.
- Tiene acceso a la “Página principal del sitio”, al “Escritorio del usuario” y a todos los submódulos de ambos.
- En el “Escritorio del usuario” tendrá una bitácora donde puede ver los últimos cambios que se han realizado en todos los proyectos donde es colaborador o propietario.
- Podrá editar algunas características de su perfil como: fotografía, correo de contacto, nombre, país, nivel de estudios e instituto al que pertenece.
- Para el módulo de “Visualización de corpus” el usuario podrá ver e interactuar con los corpus que estén registrados como públicos y privados.
- Este usuario tiene acceso al “Módulo de gestión y edición de corpus” y para un mejor control del sistema se definen dos subroles:
    - **Propietario:**
        - Puede modificar la información del corpus
        - Puede eliminar el corpus
        - Tiene acceso a la bitácora del corpus
        - Puede subir, editar y eliminar documentos
        - Puede editar los metadatos de un documento
        - Puede cambiar el estado de trabajo del corpus, de individual a colaborativo y viceversa

- Puede agregar colaboradores y eliminarlos

- **Colaborador:**
  - Puede modificar la información del corpus
  - Tiene acceso a la bitácora del corpus
  - Puede subir, editar y eliminar documentos
  - Puede editar los metadatos de un documento
  - Puede agregar colaboradores

4.4 Casos de uso

En esta sección presentamos los casos de uso que se contemplan para cada uno de los módulos de GECO4. El caso de uso para el módulo de "Conversión de documentos" se muestra en la Fig. 15.

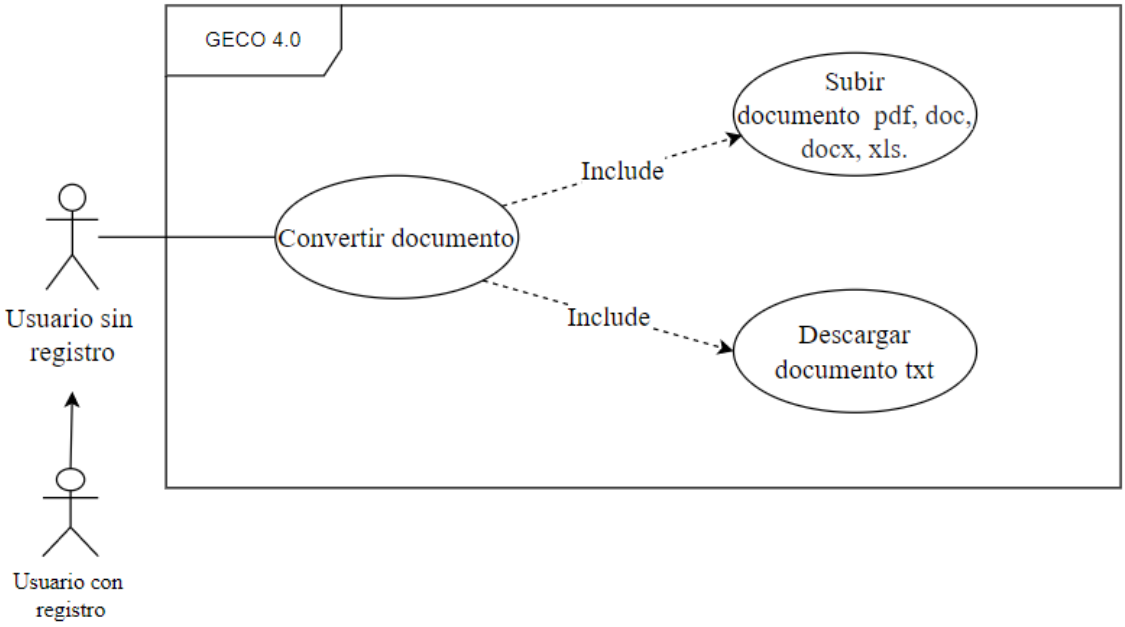


Figura 15: Diagrama de casos de uso "Conversión de documentos" del sistema GECO4

El caso de uso para la “Visualización de documentos” cuando un usuario no tiene registro se muestra en la Fig. 16.

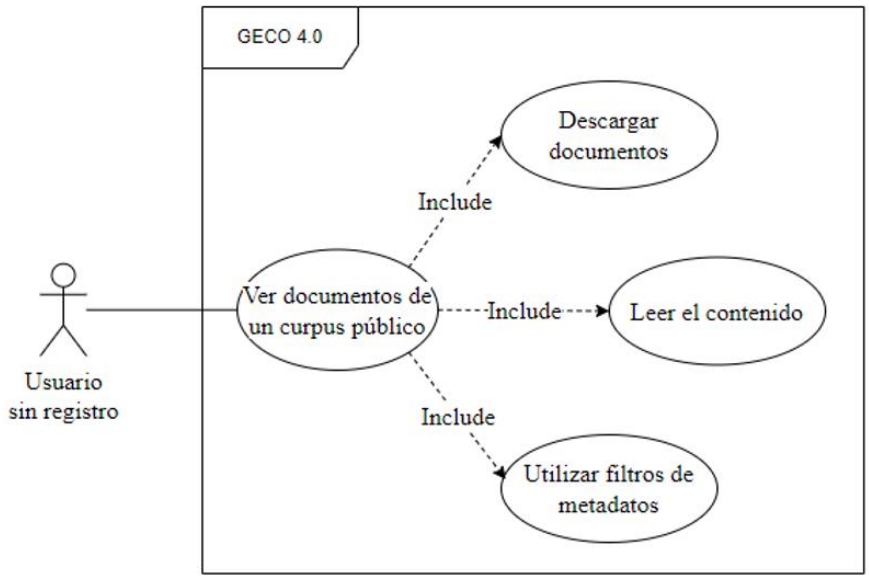


Figura 16: Diagrama de casos de uso “Visualización de documentos Usuario sin registro” del sistema GECO4

El caso de uso para la “Visualización de documentos” cuando un usuario inicia sesión se muestra en la Fig. 17.

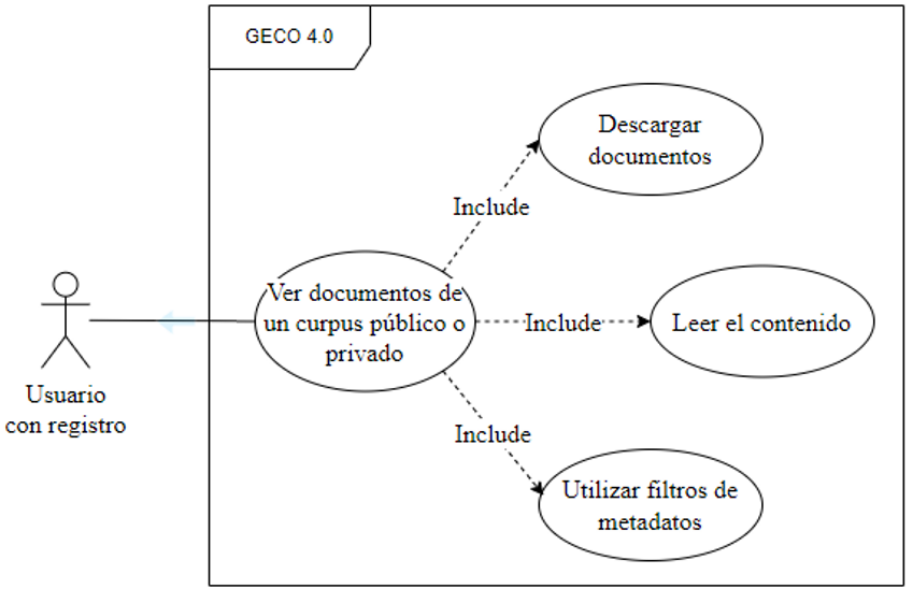


Figura 17: Diagrama de casos de uso “Visualización de documentos Usuario con registro” del sistema GECO4

El caso de uso para el módulo de “Gestión de documentos” cuando un usuario es propietario o colaborador se muestra en la Fig. 18.

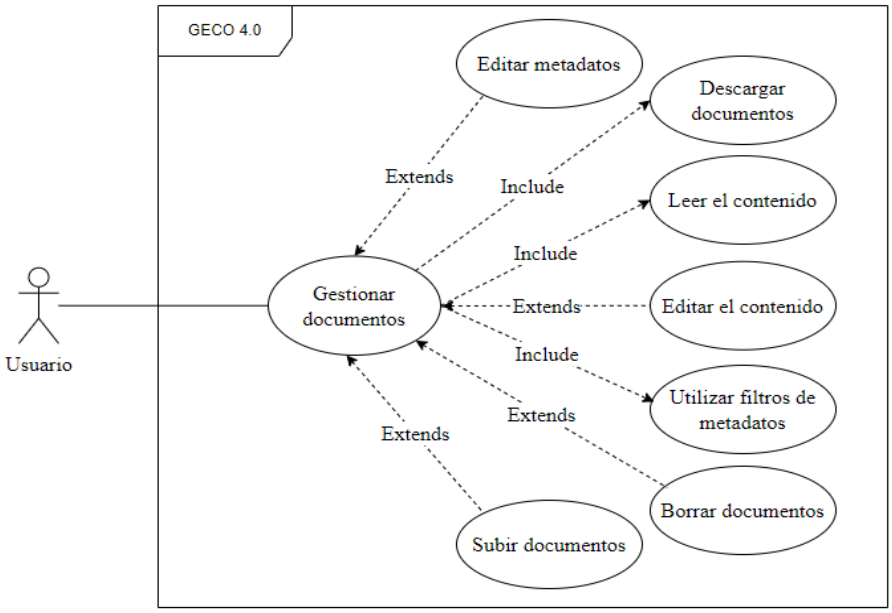


Figura 18: Diagrama de casos de uso “Gestión de documentos” del sistema GECO4

El caso de uso para el módulo de “Gestión y edición de corpus “cuando un usuario es propietario del corpus se muestra en la Fig. 19.

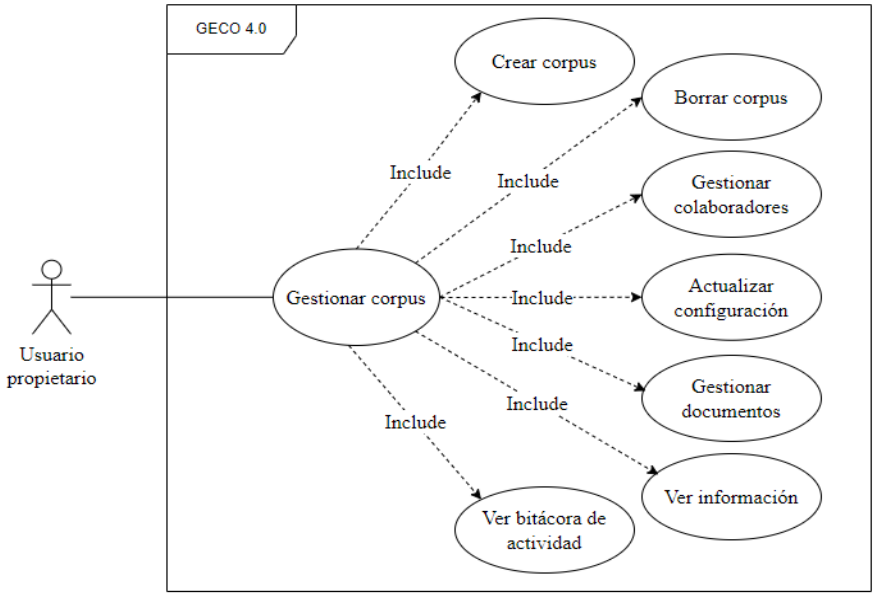


Figura 19: Diagrama de casos de uso “Gestión y edición de corpus Usuario propietario” del sistema GECO4

El caso de uso para el módulo de “Gestión y edición de corpus “cuando un usuario es colaborador del corpus se muestra en la Fig. 20.

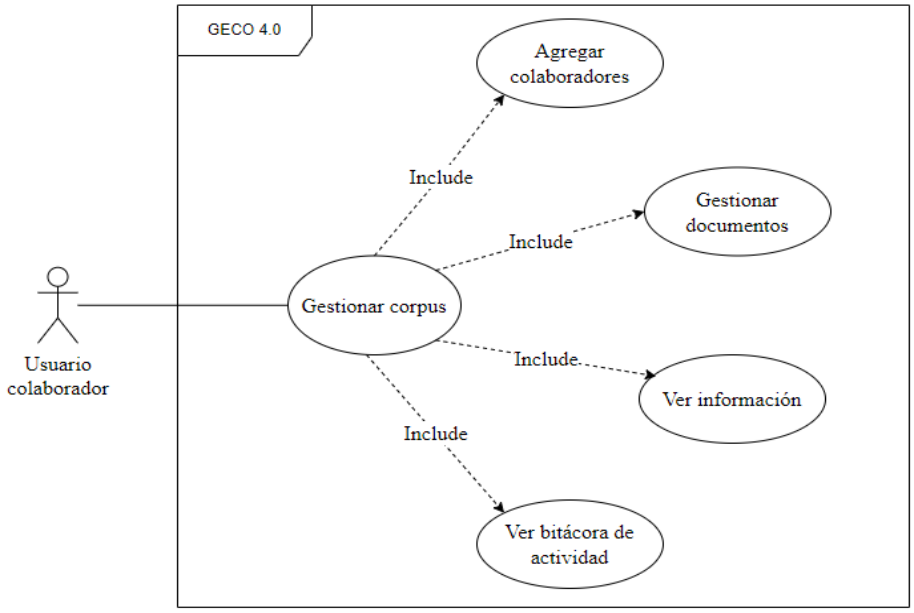


Figura 20: Diagrama de casos de uso “Gestión y edición de corpus Usuario colaborador” del sistema GECO4

Y finalmente, el caso de uso para la “Creación de corpus” se muestra en la Fig. 21.

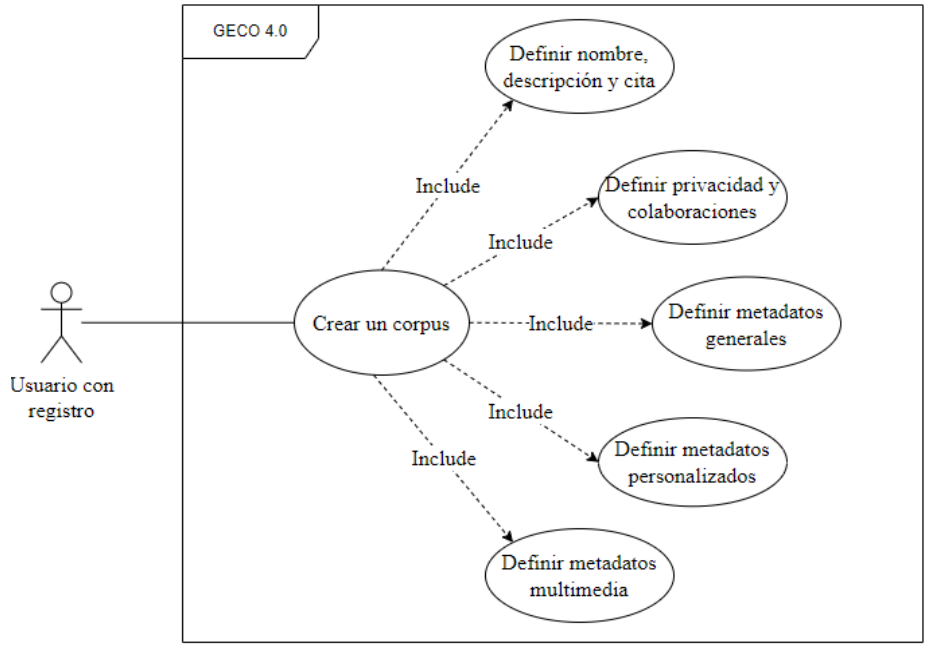


Figura 21: Diagrama de casos de uso “Creación de corpus” del sistema GECO4

Capítulo 5. Diseño del sistema

El planteamiento es crear un producto de software para el GIL de la UNAM que sea escalable y organizado en módulos que permitan publicar y gestionar de una mejor forma los corpus que se crean para el análisis lingüístico, para que su uso sea entre toda la comunidad de investigación lingüística de habla hispana, reuniendo así corpus de diversas regiones del mundo.

De acuerdo con requerimientos del usuario y del negocio descritos en el capítulo anterior, se puede observar que cada funcionalidad puede ser modelada y organizada en diferentes módulos especializados en realizar una tarea en específico. De modo que, si se desea agregar un nuevo proceso al sistema, entonces se deberá incorporar un nuevo módulo al flujo que corresponda. Cumpliendo así, el objetivo de realizar un sistema escalable y mantenible.

Optamos por usar el patrón de diseño modelo-vista-template para la comunicación entre el front-end con el back-end, pues este nos permite mantener separados la presentación, de la funcionalidad y las conexiones con el modelo de datos, generando así una mayor manejabilidad para el mantenimiento del sistema.

El lenguaje de programación que decidimos usar fue Python, pues el uso de su framework (Django) facilita y nos permite mantener un estándar para el desarrollo web.

El gestor de base de datos elegido fue MySQL, ya que al igual que otros manejadores de bases de datos adapta correctamente al volumen transaccional que se manejaría en la aplicación y al esquema relacional, además de ser de código abierto y fácil de usar.

5.1 Modelo Relacional

Las reglas de negocio explicadas anteriormente definieron el esqueleto inicial de la base de datos y con los datos obtenidos de la toma de requerimientos se definieron las entidades y sus

relaciones. A continuación, se describe y presenta en la Fig. 22 parte del diagrama del modelo relacional de la base de datos del sistema GECO4. Por motivos de seguridad y confidencialidad del sistema, no se puede presentar completo y algunos datos fueron omitidos.

Módulo de Creación de Corpus

- Un usuario puede crear en el sistema varios corpus
- Un corpus tiene dos o más metadatos del sistema obligatorios. Los metadatos del sistema pueden estar en uno o más corpus.
- Los metadatos personalizados están relacionados solamente con un corpus. Un corpus puede tener cero o más metadatos multimedia
- Los metadatos multimedia están relacionados solamente con un corpus. Un corpus puede tener cero o más metadatos multimedia.

Módulo de Gestión y edición de Corpus

- Un corpus puede tener cero o más colaboradores
- Un corpus puede tener de cero a muchos documentos relacionados. Un documento puede estar relacionado únicamente con un corpus.
- Un documento está relacionado con dos o más metadatos del sistema. Los metadatos del sistema están relacionados con cero o más documentos.
- Un documento está relacionado con cero o más metadatos personalizados. Los metadatos personalizados están relacionados con cero o más documentos.
- Un documento está relacionado con cero o más metadatos multimedia. Los metadatos multimedia están relacionados con cero o un documento.

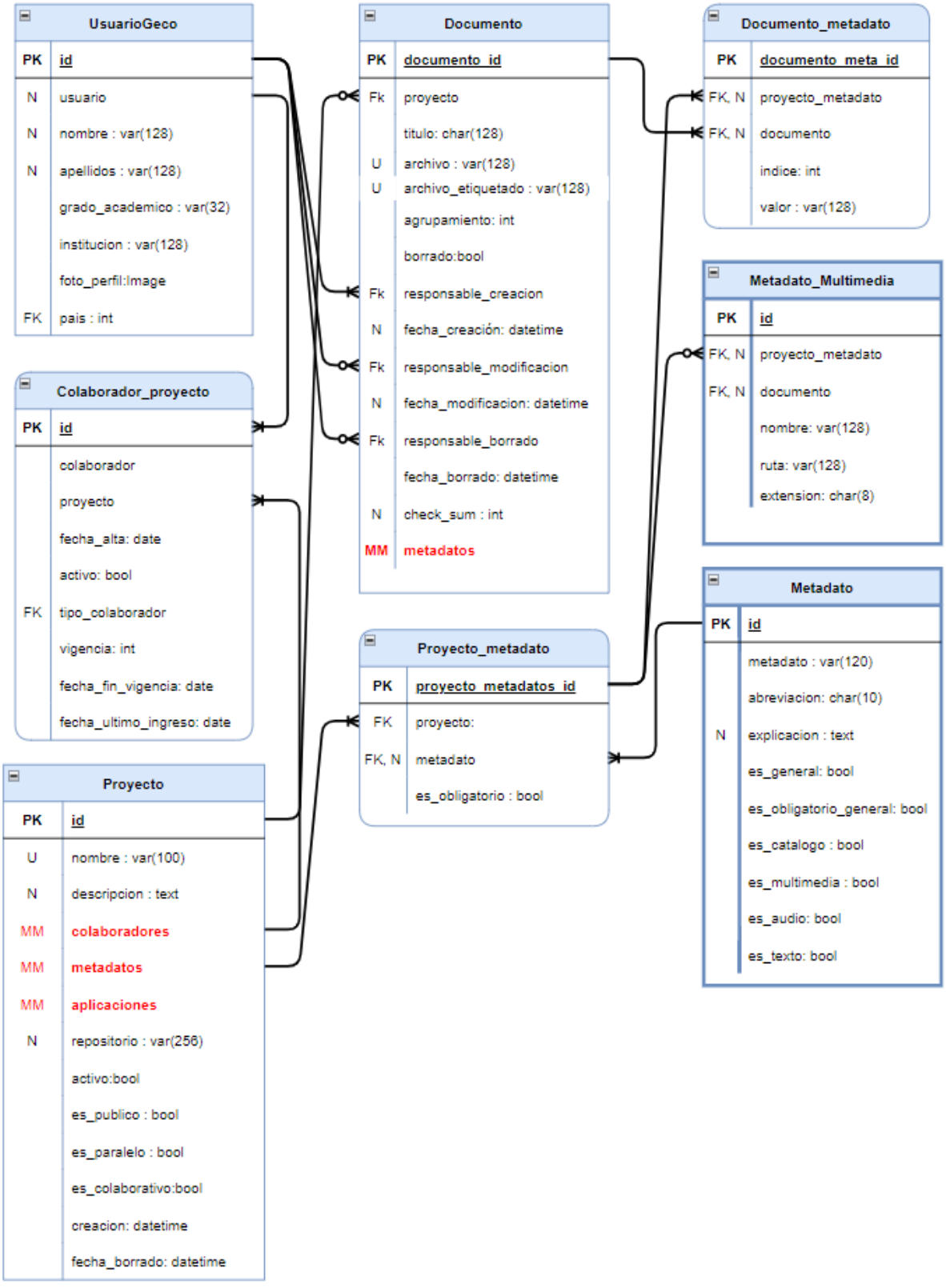


Figura 22: Diagrama parcial "entidad-relación" de la base de datos del sistema GECO4

5.2 Creación de Aplicaciones

Django fue diseñado para hacer que las tareas comunes de desarrollo web sean rápidas y sencillas y reutilizables, es por eso que en la filosofía de Django encontramos dos conceptos o estructuras importantes, los proyectos y las aplicaciones.

- **Aplicación:** es una aplicación web que hace algo. Una aplicación está compuesta usualmente por una serie de modelos (tablas de base de datos), vistas, plantillas y pruebas.
- **Proyecto:** es una colección de configuraciones y aplicaciones. Un proyecto puede estar compuesto por una o múltiples aplicaciones. Es importante resaltar que no puedes ejecutar una aplicación Django sin un proyecto.

GECO es un proyecto que se estructura con dos aplicaciones, escritorio y proyectos[13]. El "escritorio" es la aplicación que se encarga del control del sistema cuando un usuario inicia sesión y está compuesto por cuatro vistas que a su vez contienen funciones y clases que controlan la parte lógica y sus vistas son las siguientes:

- Escritorio
- Mi corpus
- Colaboraciones
- Nuevo proyecto

La aplicación llamada "proyectos" que se encarga del control del sistema cuando un usuario no está con una sesión activa, es decir no está logueado y sus vistas son las siguientes.

[13] Nota: la segunda aplicación se llama "proyectos", no hay que confundirla con el concepto de proyectos de la filosofía de Django

- Paralelos
- No paralelos

Los corpus son colecciones de documentos sobre un tema en particular, sin embargo, se hace la división de acuerdo al tipo de corpus que se va a tratar, pues las características de procesamiento de datos son distintas entre sí, según dice Jos Hallebeek en su artículo "El Corpus paralelo" las características de un corpus paralelo son las siguientes:

"*Un corpus de textos no contiene necesariamente textos en una sola lengua. Puede ser de dos (corpus bilingüe) o de más lenguas (corpus multilingüe). En tales casos los textos del corpus no son textos reunidos arbitrariamente, sino que están escogidos según idénticos criterios de selección en una y otra lengua. Por ejemplo, el Aarhus Corpus of Danish, French and English está compuesto de textos en tres lenguas que tratan todos del mismo tema: el derecho de contrato. No son traducciones de los mismos textos. Los textos son diferentes, pero coinciden en la temática. Cuando un corpus tiene los mismos textos en diferentes lenguas se habla de un parallel corpus: corpus paralelo.* "

La estructura general de la organización del proyecto y las aplicaciones de GECO se muestran en la Fig. 23.

Figura 23: Diagrama de "Organización de Aplicaciones y Proyectos" del sistema GECO4

GECO se ha diseñado con el propósito de tener escalabilidad en el sistema y contemplando la posibilidad de agregar nuevas reglas de negocio que surjan del GIL de la UNAM. La razón por la que el sistema está diseñado para tener escalabilidad es porque el GIL tiene previsto un aumento en el volumen de operaciones de análisis lingüístico y de almacenamiento de corpus, de modo que se necesita que el sistema pueda crecer fluidamente sin deteriorar su calidad. Esto puede requerir que se añadan recursos de hardware o que se realicen cambios en su configuración para ajustarse a demandas de servicio cambiantes sin necesidad de rediseñar, aprovechando recursos que se le añadan.

Capítulo 6. Metodología de Desarrollo del sistema

A manera de ejemplo, presentamos el desarrollo básico de una de las vistas del sistema, aunque sencilla, esta vista ayuda a ilustrar el proceso completo del desarrollo.

La vista que se presenta en la Fig. 24 corresponde a la pestaña información de la página principal, en ella se muestra la información general del corpus:

Figura 24: Pestaña de información del sistema GECO4

Para desarrollar esta página es necesario establecer claramente qué información vamos a presentar, y por lo tanto los modelos involucrados, luego determinar cómo nos gustaría que se viera en pantalla y finalmente cómo a partir de los modelos obtendremos la información.

Este proceso se ha seguido en mayor o menor medida a lo largo del desarrollo del sistema.

En este ejemplo, deseamos mostrar la información general del corpus: El número de usuarios registrados en el corpus, el número de corpus y el total de documentos registrados en el sistema.

6.1 Modelos involucrados

En este caso solo se necesita saber la información referente a usuarios, proyectos y documentos.

En la Fig. 25 se muestran los modelos involucrados marcados en rojo.

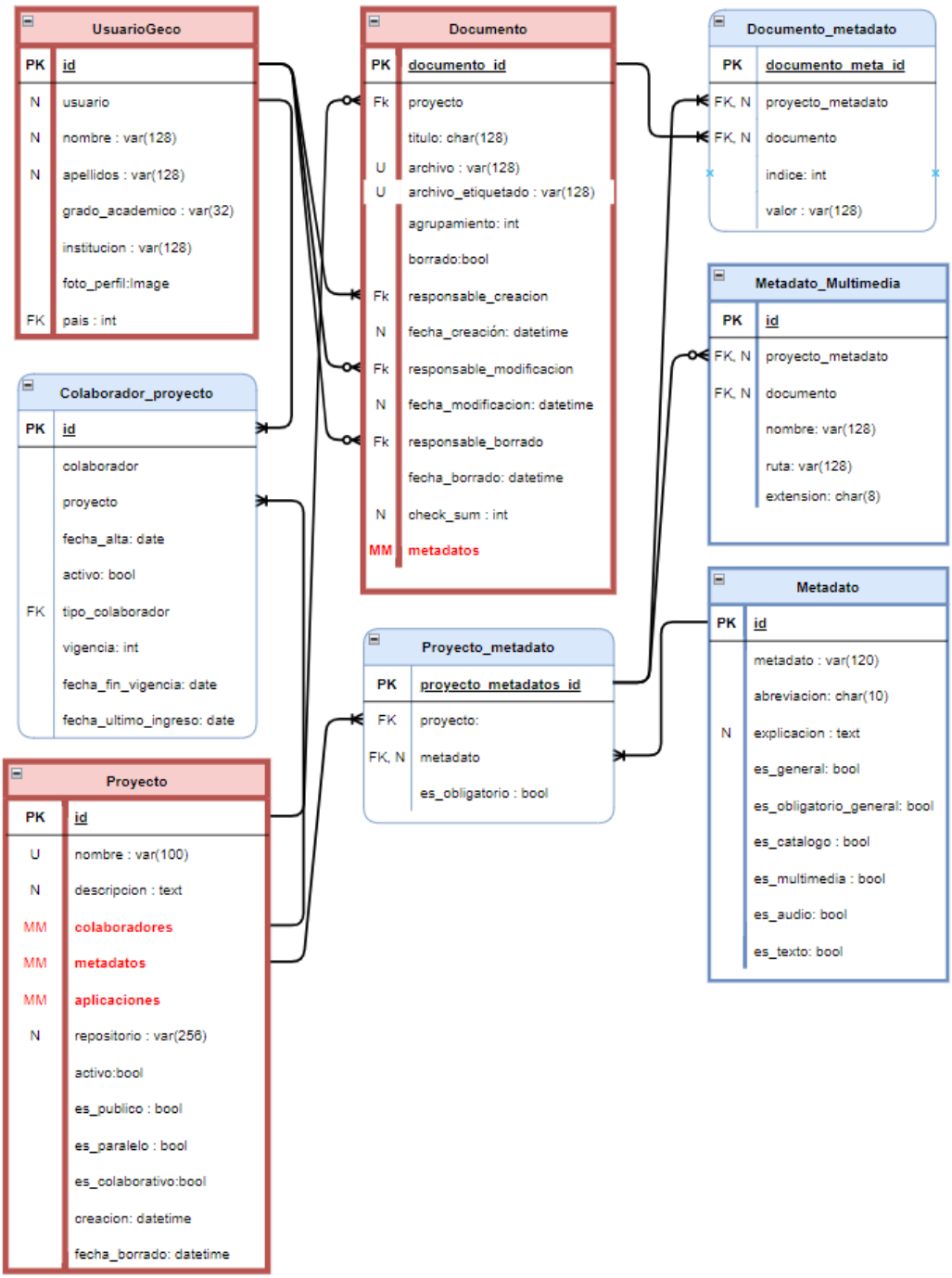


Figura 25: Diagrama parcial "entidad-relación" del sistema GECO4

Estos modelos tienen que plasmarse en el sistema dentro de un archivo que define los modelos (models.py) el contenido de este archivo, en particular para los tres modelos aquí planteados se presentan en el apéndice 1, en las Fig. 26-28 se puede observar la definición de los atributos y algunos métodos que se han definido para trabajar más cómodamente con estas entidades.

```python
class Proyecto(models.Model):
    nombre = models.CharField(max_length=128, unique=True, verbose_name='nombre del proyecto',help_text='nombre del proyecto para el corpus')
    descripcion = models.TextField(
        verbose_name='descripción', help_text='breve descripción del proyecto')
    cita = models.TextField(
        null=True, verbose_name='cita', help_text='Forma de citar el proyecto')
    colaboradores = models.ManyToManyField('UsuarioGeco', through='Colaborador_proyecto', related_name='proyectos')
    metadatos = models.ManyToManyField('Metadato', through='Proyecto_metadato', related_name='proyectos')
    aplicaciones = models.ManyToManyField('Aplicacion', through='Proyecto_aplicacion', related_name='proyectos')
    repositorio = models.CharField(max_length=256, editable=False, unique=True, default=ceateRepositorioName)
    activo = models.BooleanField(default=True, help_text='habilitar para indicar que el proyecto está activo')
    es_publico = models.BooleanField(default=False, help_text='habilitar para indicar que el proyecto es público (privado en caso contrario)')
    es_paralelo = models.BooleanField(default=False, help_text='habilitar para indicar que será un corpus paralelo')
    es_colaborativo = models.BooleanField(default=False, help_text='habilitar para indicar que será un corpus colaborativo')
    creacion = models.DateTimeField(auto_now_add=True, editable=False)
    fecha_borrado = models.DateTimeField(auto_now_add=False, editable=False, null=True, blank=True)

    class Meta: ···

    def save(self, *args, **kwargs): ···

    def __str__(self): ···

    def getColaboradores(self): ···

    def propietario(self): ···


    def getMetadatos(self): ···
```

Figura 26: Bloque de código "Clase Proyecto" del sistema GECO4

```python
class Documento(models.Model):
    proyecto = models.ForeignKey('Proyecto', on_delete = models.CASCADE,null=False)
    titulo = models.CharField(max_length= 128, unique = False, help_text = 'Titulo principal del documento')
    archivo = models.CharField(max_length = 128, help_text = 'archivo donde se encuentra el documento')
    archivo_etiquetado = models.CharField(max_length = 128, blank=True, null=True, help_text = 'archivo donde se encuentra el documento etiquetado')
    agrupamiento = models.IntegerField(default=0)
    borrado = models.BooleanField(default=False,help_text='¿El documento ha sido enviado a papelera?')
    responsable_creacion = models.ForeignKey('UsuarioGeco',related_name='responsable_creacion',on_delete=models.SET_NULL,null=True)
    fecha_creacion = models.DateTimeField(verbose_name='fecha de creacion')
    responsable_modificacion = models.ForeignKey('UsuarioGeco',related_name='responsable_modificacion',on_delete=models.SET_NULL,null=True)
    fecha_modificacion = models.DateTimeField(verbose_name='fecha de modificacion',null=True,blank=True)
    responsable_borrado = models.ForeignKey('UsuarioGeco',related_name='responsable_borrado',on_delete=models.SET_NULL,null=True)
    fecha_borrado = models.DateTimeField(verbose_name='fecha de borrado',null=True,blank=True)
    check_sum = models.IntegerField(verbose_name='código verificador',default=0,editable=False)
    metadatos = models.ManyToManyField('Proyecto_metadato',through='Documento_metadato',related_name='documentos')

    class Meta: ···
    def save(self, *args, **kwargs): ···

    def __str__(self): ···

    def getDocInfo(self): ···

    def getDocInfoRemasterizado(self): ···


    def getRengloMetadatos(self,no_disponible='valor desconocido'): ···

    def getActualizar(self): ···
```

Figura 27: Bloque de código "Clase Documento" del sistema GECO4

```python
class UsuarioGeco(models.Model):
    OPCIONES_GRADO = (('', 'Elegir...'), ('LIC','Licenciatura'), ('ESP','Especialización'), ('MBA','Mestría'), ('DOC','Docotado'), ('POS','Postdo
    usuario = models.OneToOneField(User, on_delete=models.CASCADE)
    nombre = models.CharField(max_length=128, unique=False)
    apellidos = models.CharField(max_length=128, unique=False)
    grado_academico = models.CharField(max_length=128, unique=False, choices=OPCIONES_GRADO)
    institucion = models.CharField(max_length=128, unique=False)
    foto_perfil = models.ImageField(default='fotosPerfil/generic.png',null=True, blank=True, upload_to='fotosPerfil/')
    pais = models.ForeignKey('PaisUsuario', on_delete=models.CASCADE, default=13, null=True)
    class Meta: ···
    def getNombreCompleto(self): ···
    def __str__(self): ···

    def getProyectos(self): ···

    def num_proyectos(self): ···

    def email(self): ···
```

Figura 28: Bloque de código "Clase UsuarioGeco" del sistema GECO4

6.2 Templates desarrollados

Una vez definidos los modelos es necesario determinar la información que el usuario visualizará, en nuestro caso corresponde a la ventana de información, como se muestra en la Fig. 29 (obtenido del sistema ya funcionando), se reporta el número de usuarios, el número de corpus

registrados y los documentos disponibles (el total de documentos independientemente del corpus al que pertenezcan). En la Fig. 29 se distingue claramente tres grandes zonas: el encabezado (head), marcado en amarillo, el cuerpo de la página (body), marcado en verde, y el pie de la página (footer) marcado en rojo.

Figura 29: Estructura del renderizado HTML del sitio GECO4

El encabezado del sistema corresponde a aquella sección de una página que no cambia con la navegación que el usuario hace dentro del sistema, generalmente muestra el menú general de navegación. El cuerpo constituye la información dinámica que se mostrará al usuario, en este ejemplo la información del sistema GECO4. Finalmente, el pie corresponde a la información de contactos, y en general es estática y siempre está presente en cualquier ventana.

A cada una de las secciones (encabezado, cuerpo y pie) le corresponde un template, es decir un archivo basado en HTML y Django que describen lo que se presentará. En el apéndice 1 se proporciona el contenido completo de estos tres Templates.

El archivo base_generic.html contiene el template que describe la estructura mencionada con anterioridad. En la Fig. 30 se aprecia la zona de encabezado, en la cual se definen las opciones del menú de la aplicación.

```
<!DOCTYPE html>
<html lang="en">

<head>
  {% block title %}
  <title>Geco 4.0 versión beta</title>
  {% load static %}
  <link rel="icon" type="image/x-icon" href="{% static 'img/salamander32.png' %}" />
  {% endblock %}
  ...
</head>

<body>
  {% load static %}
  <nav class="navbar navbar-expand-lg bg-secondary text-uppercase fixed-top" id="mainNav"
    ...
    Menú
      <i class="fas fa-bars"></i>
    </button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger"
            href="{% url 'informacion' %}">Información</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger"
            href="{% url 'no_paralelos' %}">Corpus</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger"
            href="{% url 'conversion-documentos' %}">Conversión de Documentos</a></li>
        {% if user.is_authenticated %}
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger"
            href="{% url 'escritorio' %}">Escritorio de {{ user.get_username }}</a></li>
        {% else %}
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger"
            href="{% url 'registro' %}">Registrar Usuario</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3 rounded js-scroll-trigger alerta"
            data-iniciosesion="{% url 'login'%}?next={% url 'escritorio' %}" href="#">Iniciar Sesión</a></li>
        {% endif %}
      </ul>
    </div>
  </nav>
```

Figura 30: Bloque de código HTML "body-navbar" del sistema GECO4

Una vez definido el encabezado, se establece el área que ocupará el cuerpo de la página, en la Fig. 31 se observa que únicamente está indicado que en ese lugar se encuentra el título y el contenido de la página, pero no se indica cuál será este.

```
{% load static %}
<header class="masthead text-white text-center" style="background-image: url(' {% static 'img/background.jpg' %} ');">
  <div class="container d-flex align-items-center flex-column">
    {% block title-content %}
    <!--- TÍTULO Y DESCRIPCIÓN DE CADA PÁGINA -->
    {% endblock %}
  </div>
</header>

<div class="margen">
  {% block content %}
  <!--- CONTENIDO DE CADA PÁGINA -->
  {% endblock %}
</div>
```

Figura 31: Bloque de código HTML "header-content" del sistema GECO4

y finalmente, en la Fig. 32 se muestra la sección que define el pie de la página, que completa el template base_generic.html.

```
...

{% block footer %}
<footer class="footer text-center">
    ...
          <h4 class="text-uppercase mb-4">Redes Sociales</h4>
        <a class="btn btn-outline-light btn-social mx-1" href="https://www.facebook.com/ingenieriaLinguistica/">
          {% load static %}
          <img class="iconFooter" src="{% static 'img/facebook.png' %}">
        </a>
        <a class="btn btn-outline-light btn-social mx-1" href="https://twitter.com/gil_unam">
          {% load static %}
          <img class="iconFooter" src="{% static 'img/twitter.png' %}">
        </a>
  ...
          gil@iingen.unam.mx
          <br> +52(55)5623-3600 ext. 8808

  ...
</footer>
{% endblock %}


<!-- Copyright Section-->

<div class="copyright py-4 text-center text-white">
  <div class="container"><small>©Todos los derechos reservados UNAM 2020. Esta página puede ser reproducida con fines
      no lucrativos, siempre y cuando no se mutile, se cite la fuente completa y su dirección electrónica. De otra forma requiere
      permiso previo por escrito de la institución.
    </small>
  </div>
  {% block copyright %}
  {% endblock %}
  </div>
</div>
```

Figura 32: Bloque de código HTML "footer" del sistema GECO4

6.3 Vista desarrollada

Ahora bien, en la sección “cuerpo” se incluye la página con base al contenido desplegar, en nuestro ejemplo corresponde a la página de información general de GECO4, el template correspondiente es información.html como se muestra en la Fig. 33.

```
{% extends "base_generic.html" %}

{% block title-content %}
<h1 class="masthead-heading mb-0">Información</h1>
<p class="masthead-subheading font-weight-light mb-0">En esta sección encontrarás estadísticas sobre todo el contenido
  del Sistema de Gestión de Corpus del Grupo de Ingeniería lingüística de la UNAM</p>
</div>
{% endblock %}

{% block content %}
<br><br>
<h1 class="page-section-heading text-left text-secondary mb-0">Actualmente en el Gestor de Corpus tenemos</h1>
<br><br>
<div class="row">
  <div class="col-md-6 col-lg-3 mb-5 ">
   ...
        Total de usuarios: {{ num_usuarios }}
   ...
     <img src="{% static 'img/usuarios.png' %}" width="100" height="100" class="align-self-center mr-3">
   ...
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    ...
        Corpus: {{num_no_paralelos}}
    ...
        independientemente de sus posibles formas o usos.</p>
    ...
        <img src="{% static 'img/no_paralelos.png' %}" width="70" height="70" class="align-self-center mr-3">
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    ...
        Documentos: {{num_documentos}}
    ...
     <img src="{% static 'img/documentos.png' %}" width="70" height="70" class="align-self-center mr-3">
    ...
    </div>
</div>

<br><br>
{% endblock %}
```

Figura 33: Bloque de código HTML “información” del sistema GECO4

Como puede apreciarse el número de usuarios (num_usuarios), el número de corpus (num_no_paralelos) y el número de documentos (num_documentos) es información que se

proporciona al template y que es generada por la vista correspondiente (definida dentro de proyectos_view.py) que se muestra en la Fig. 34.

```
def informacion(request):

    numeroUsuarios = UsuarioGeco.objects.count()
    numeroDocumentos = Documento.objects.filter(borrado=False).count()
    numeroParalelos = Proyecto.objects.filter(es_paralelo=True, activo = True).count()
    numeroNoParalelos = Proyecto.objects.filter(es_paralelo=False, activo = True).count()


    # Number of visits to this view, as counted in the session variable.
    numeroVisitas = request.session.get('num_visitas', 0)

    request.session['num_visitas'] = numeroVisitas+1
    context = {
        'num_usuarios' : numeroUsuarios,
        'num_documentos' : numeroDocumentos,
        'num_paralelos' : numeroParalelos,
        'num_no_paralelos': numeroNoParalelos,
        'num_visitas' : numeroVisitas
    }

    setSessionProyectoActual(request,context)

    return render(request, 'informacion.html',context=context)
```

Figura 34: Bloque de código "Función información" del sistema GECO4

Con esto se completa la definición del esquema modelo-vista-template. La ejecución de la vista, que pone en funcionamiento todo el esquema se lleva a cabo en el archivo urls.py que se muestra en la Fig. 35, en donde se indica que al momento de solicitar la liga información/ se entregue el template creado a partir de la vista información:

```python
from django.urls import path
from . import views
from django.conf import settings
from django.urls import re_path
from django.views.static import serve

urlpatterns = [
    path('',views.index, name ='index'),

    path('informacion/',views.informacion, name ='informacion'),

    ...

]
```

Figura 35: Bloque de código "Urlpatterns información" del sistema GECO4

Este es el proceso general que se ha seguido en todo el desarrollo del sistema, en total se desarrollaron 35 vistas, 29 modelos y 33 templates. A continuación, nos enfocaremos en la descripción de la implementación de los módulos desarrollados.

Capítulo 7. Implementación

Los módulos diseñados cubren la serie de actividades que el usuario ha planteado como requerimientos funcionales, de tal forma que el sistema será integral y adaptado a las necesidades del análisis lingüístico. En este apartado presentamos su implementación.

Cuando el usuario entra al sistema de GECO4 aparece una pantalla de bienvenida y en la parte superior podrá encontrar un menú (1) como se muestra en la Fig. 36 en el que podrá acceder a distintas partes del sistema. Este menú incorpora los módulos que se han descrito en capítulos anteriores del presente trabajo, por lo que a continuación se presenta la implementación y funcionalidad de cada uno.

Figura 36: Página principal del sitio GECO4

7.1 Módulo de Visualización de Corpus

Para la explicación de este módulo primero se ilustrará el funcionamiento que tiene con el rol de un "usuario no registrado" y después se complementará con el rol de "usuario registrado".

Cuando el usuario no registrado accede a la pestaña de "Corpus" (1) aparecerá un listado de los corpus que hay registrados en el sistema como públicos y para ver el contenido de cada uno de

ellos deberá de presionar sobre el nombre del corpus que desee (2) como se muestra en la Fig. 37.

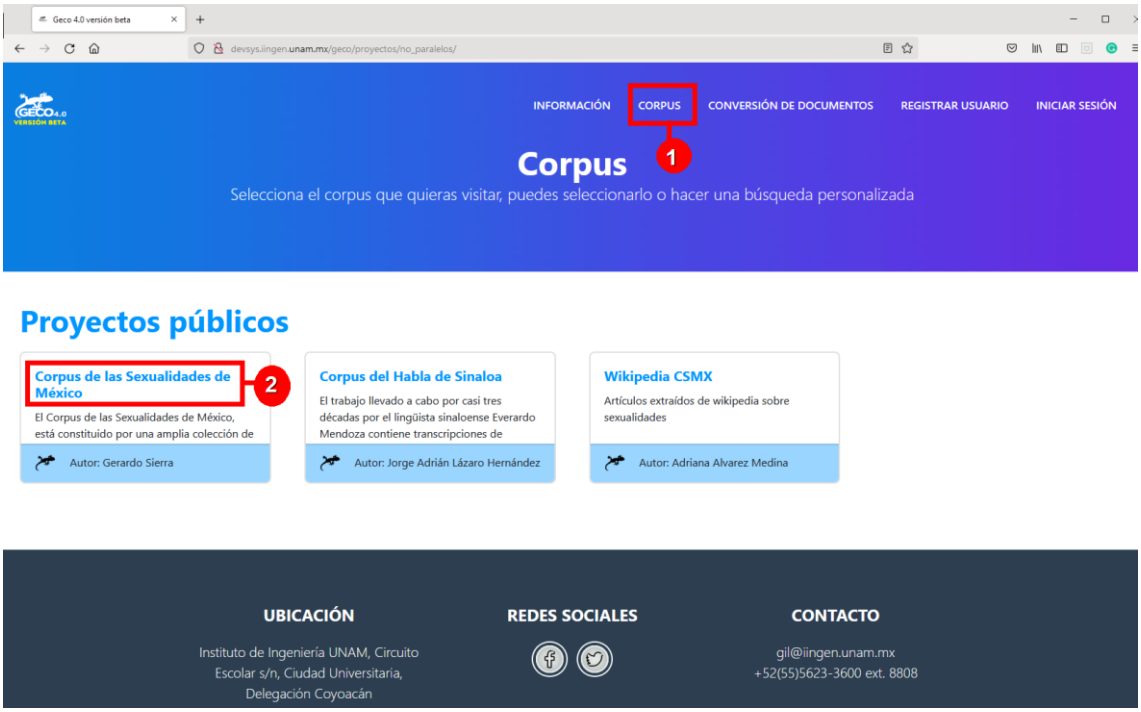


Figura 37: Visualización de corpus para el "usuario no registrado" en el sistema GECO4

Esta acción lo lleva a la vista individual de dicho corpus, donde debajo del nombre del corpus aparece un pequeño menú de navegación y su primera pestaña es la "Descripción" (3) que se muestra en la Fig. 38 y que muestra su información general.

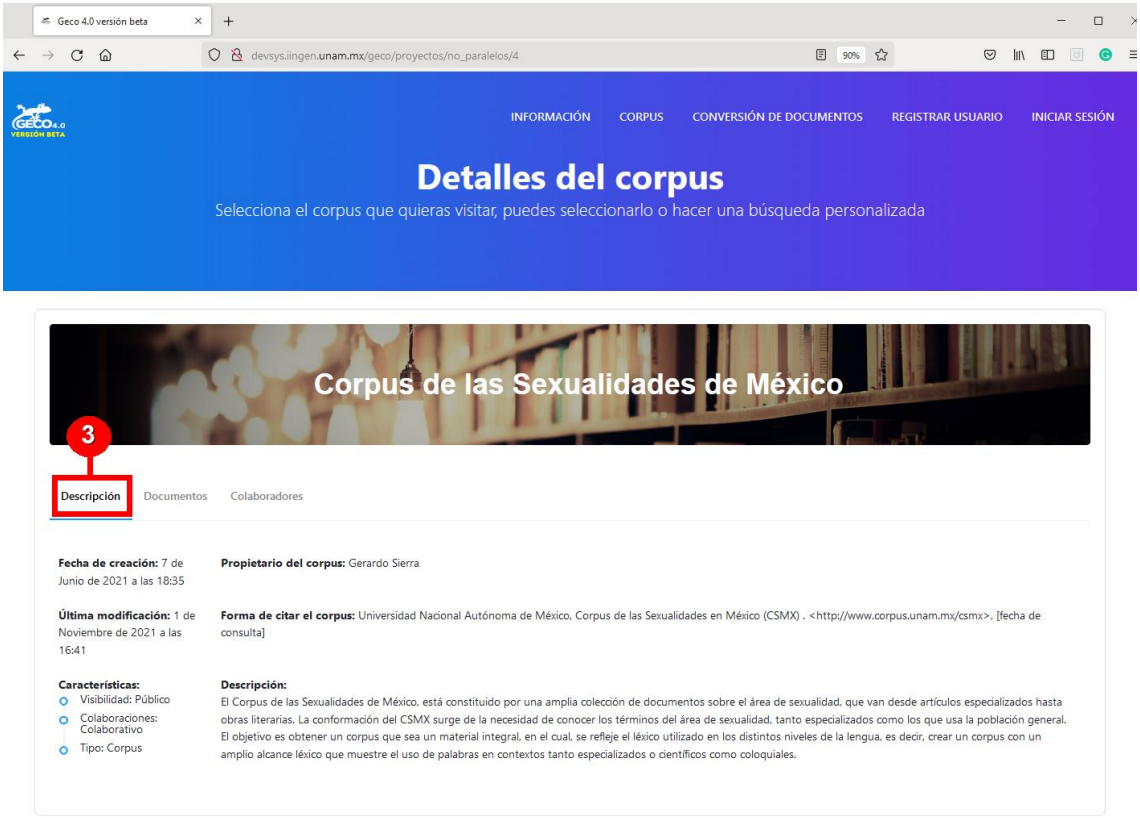


Figura 38: Pestaña de descripción en la visualización de un corpus en el sistema GECO4

Después está la pestaña de “Documentos” (4) en la que aparece un listado de todos los documentos que conforman a dicho corpus, así como sus metadatos (5) y a la izquierda la opción para descargar los documentos que el usuario desee (6). Si es el deseo del usuario, también podrá ver el contenido de los documentos sin tener que descargarlos, para eso el usuario debe dar clic sobre el nombre del documento (7) como se muestra en la Fig. 39 y será redirigido a una pestaña donde podrá ver su contenido (8) como se muestra en la Fig. 340.

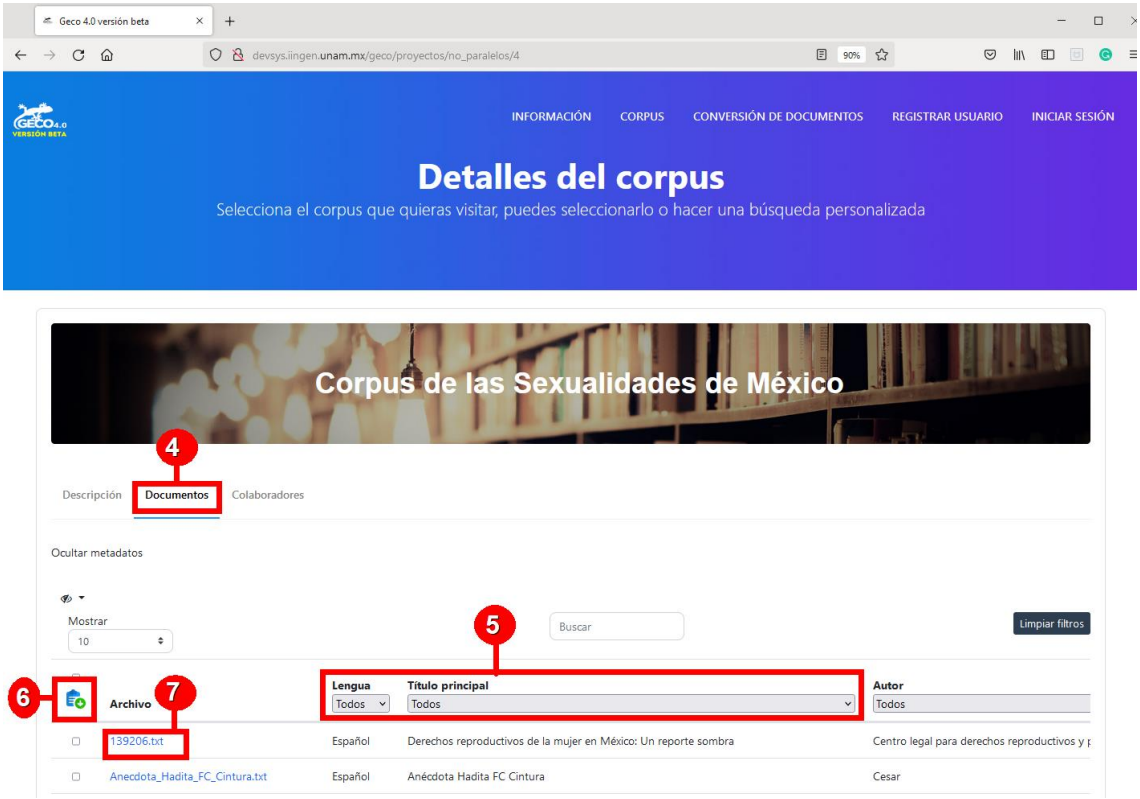


Figura 39: Pestaña de documentos en la visualización de un corpus en el sistema GECO4

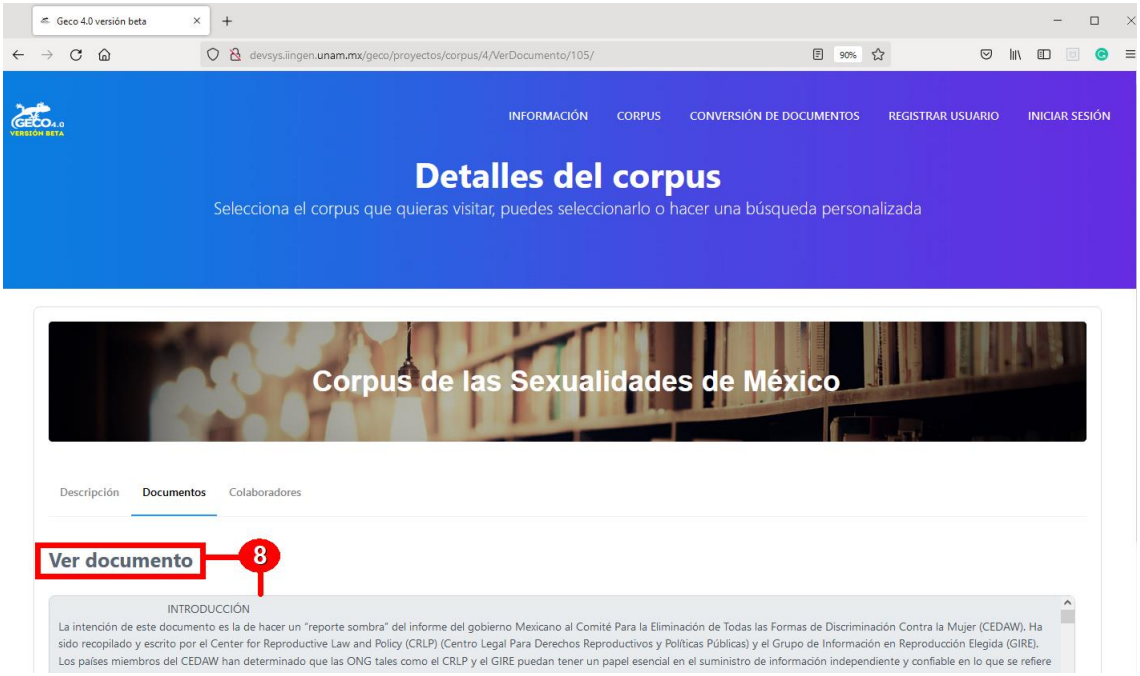


Figura 40: Contenido de un documento en la visualización de un corpus en el sistema GECO4

Y finalmente, en la Fig. 41 se encuentra la pestaña de “Colaboradores” (9) que muestra la información de los usuarios que participan en ese corpus.

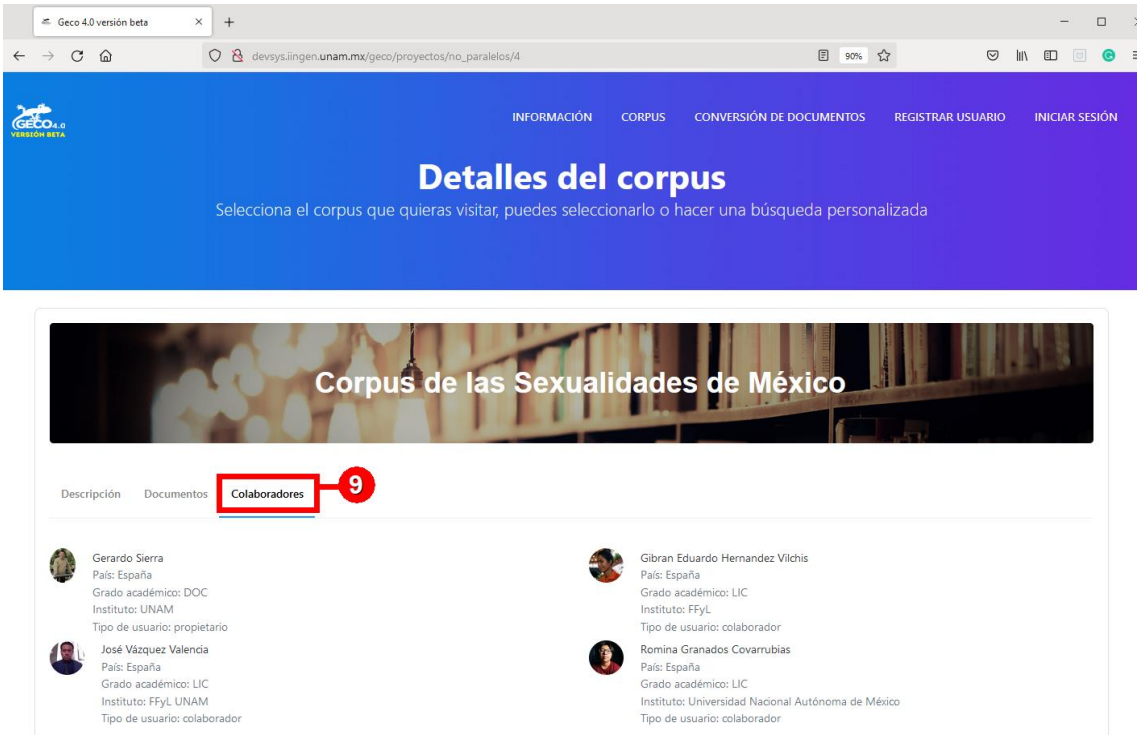


Figura 41: Pestaña de colaboradores en la visualización de un corpus en el sistema GECO4

Para el “usuario registrado” el funcionamiento de este módulo es exactamente el mismo, para esto, deberá iniciar sesión en el sistema (10) y la diferencia con el “usuario no registrado” es que también podrá interactuar con los corpus que estén registrados como “privados” (11) como se muestra en la Fig. 42.

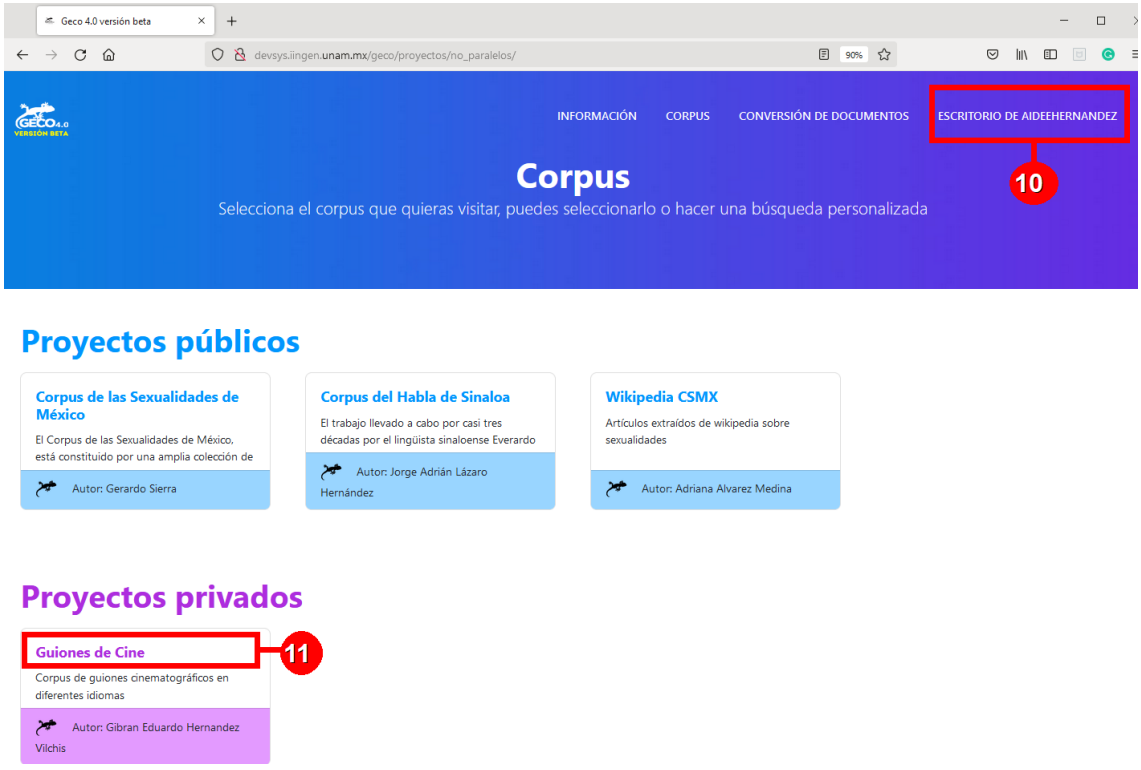


Figura 42: Visualización de corpus para el "usuario registrado" en el sistema GECO4

7.2 Módulo de Conversión de documentos

El funcionamiento de este módulo es indistinto al tipo de usuario que lo quiera utilizar. Para acceder a él, se debe de seleccionar la pestaña de "Conversión de documentos" (1) que se encuentra en el menú superior del sistema. Después deberá seleccionar un documento (2) ".doc", ".docx", ".xls", ".xlsx", ".pdf" y ".odt" a un documento ".txt", una vez que se haya cargado el documento, se debe de presionar el botón de "convertir" (3) como se muestra en la Fig. 43. Finalmente, el sistema va a hacer la conversión y cuando esté listo el usuario deberá de presionar el botón de "descargar" (4) de la Fig. 44.

Figura 43: Visualización de conversión de documentos en el sistema GECO4

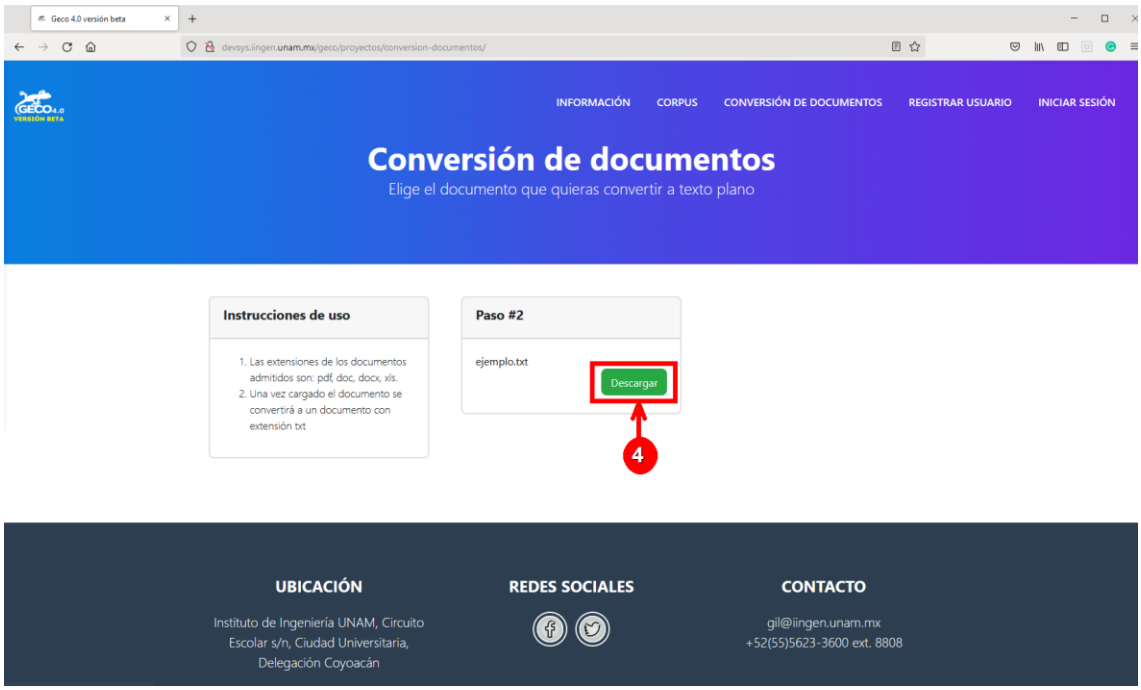


Figura 44: Descarga de documento convertido a "txt" en el sistema GECO4

7.3 Módulo de Creación de Corpus

Para que un usuario pueda interactuar con este módulo deberá de iniciar sesión en el sistema GECO4, después se debe de dirigir a la pestaña de “Mis corpus” (1) que se encuentra en el menú lateral izquierdo. En esa pestaña, hay un botón llamado “Nuevo” (2) que le permite al usuario crear un nuevo corpus como se muestra en la Fig. 45.

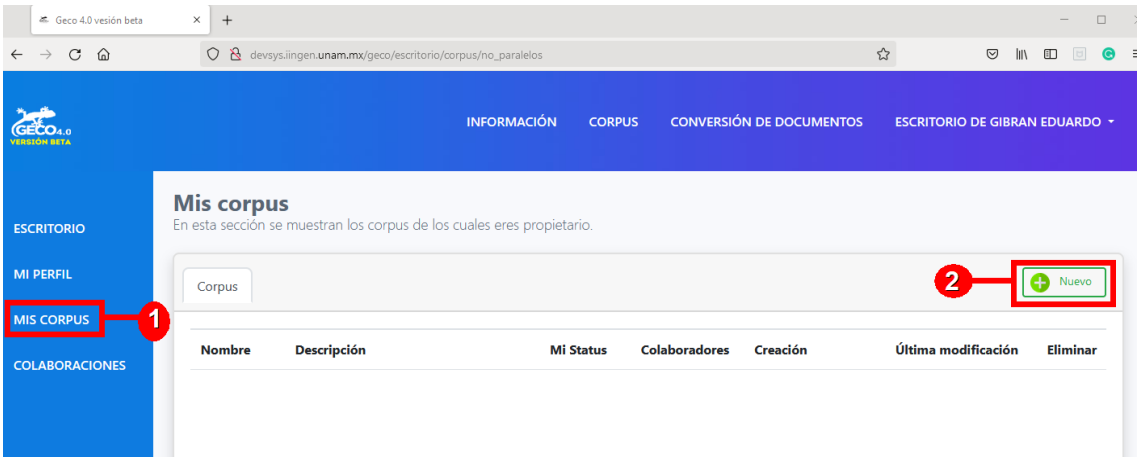


Figura 45: Creación de corpus en el sistema GECO4

Cuando se presiona ese botón, el usuario es redireccionado a un formulario que deberá llenar para crear un corpus. El formulario de la Fig. 46 cuenta con varias pestañas que clasifican el tipo de información que se debe de registrar, empezando por la “Descripción” (3) donde se debe de poner toda la información general del corpus, como su nombre, descripción, forma de citar el corpus, tipo de publicación y forma de trabajo.

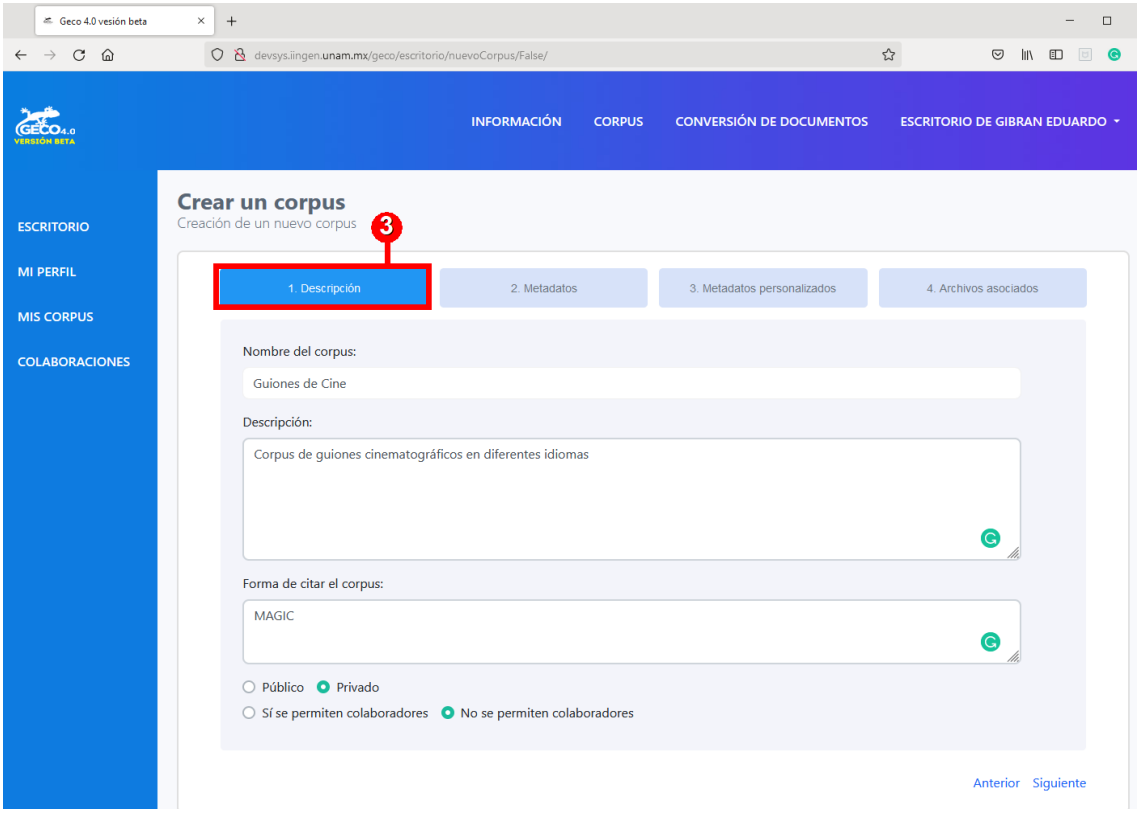


Figura 46: Vista del formulario para la creación de corpus en el sistema GECO4

La siguiente información que debe de llenar el usuario es la de los “Metadatos” (4) que son los que se el sistema ofrece al usuario, los metadatos de “Lengua” y “Título principal” (5) siempre se van a incluir en todos los corpus y su llenado es de forma obligatoria. Adicionalmente el usuario puede elegir más de los metadatos que aparecen en el sistema y dependiendo de su corpus, los podrá marcar como obligatorios (6) u opcionales (7) como se muestra en la Fig. 47.

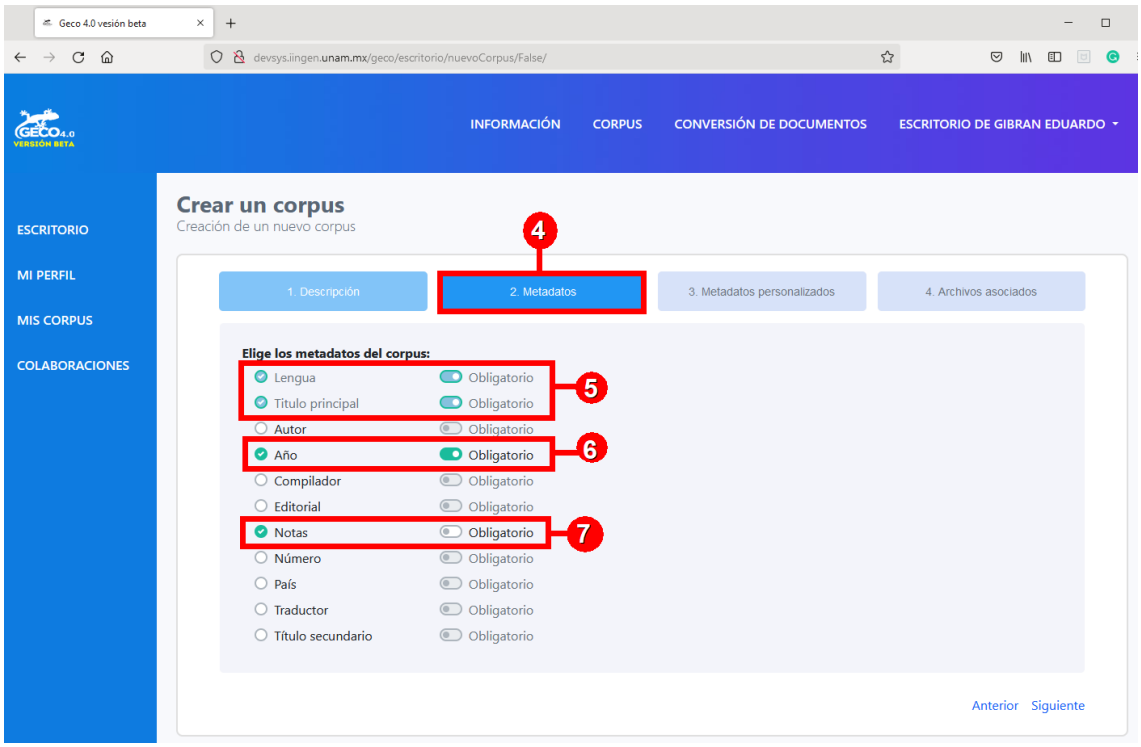


Figura 47: Pestaña de "Metadatos" del formulario para la creación de corpus en el sistema GECO4

La pestaña de "Metadatos personalizados" (8) permite que el usuario capture los metadatos que mejor se adapten a las necesidades de análisis lingüístico de su corpus, para agregar un metadato deberá poner el nombre y presionar el botón de "Agregar" (9). El usuario podrá establecer si su obligatoriedad, eliminarlo o editarlo si así lo requiere como se muestra en la Fig. 48.

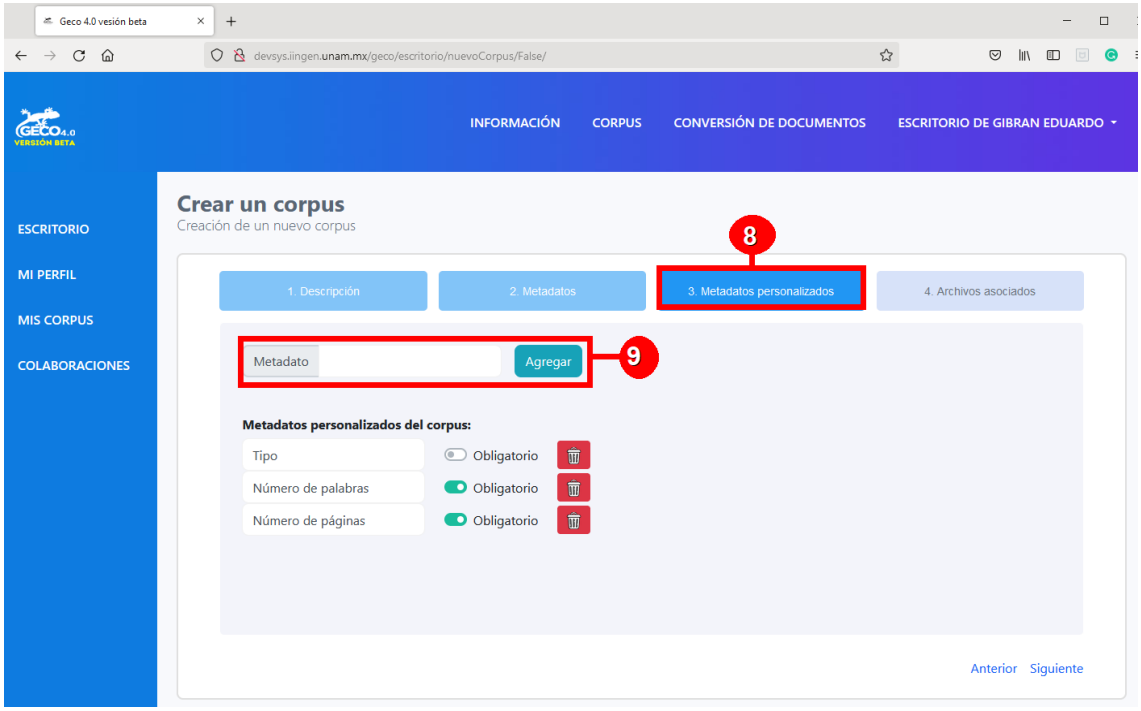


Figura 48: Pestaña de "Metadatos personalizados" del formulario para la creación de corpus en el sistema GECO4

Finalmente, la pestaña de "Archivos asociados" (10) se refiere a los metadatos multimedia, que son recursos de apoyo que clasifican en archivos de texto o de audio, el usuario podrá capturar los metadatos que mejor se adapten a las necesidades de análisis lingüístico de su corpus, incluso, podría decidir no utilizar este tipo de metadatos. Al igual que en la pestaña anterior, deberá poner el nombre y presionar el botón de "Agregar" (11). El usuario debe especificar el tipo de recurso que se va a cargar (12) y podrá establecer si su obligatoriedad, eliminarlo o editarlo si así lo requiere. Y para la creación del corpus debe de presionar el botón de "Crear corpus" (13) como se muestra en la Fig. 49.

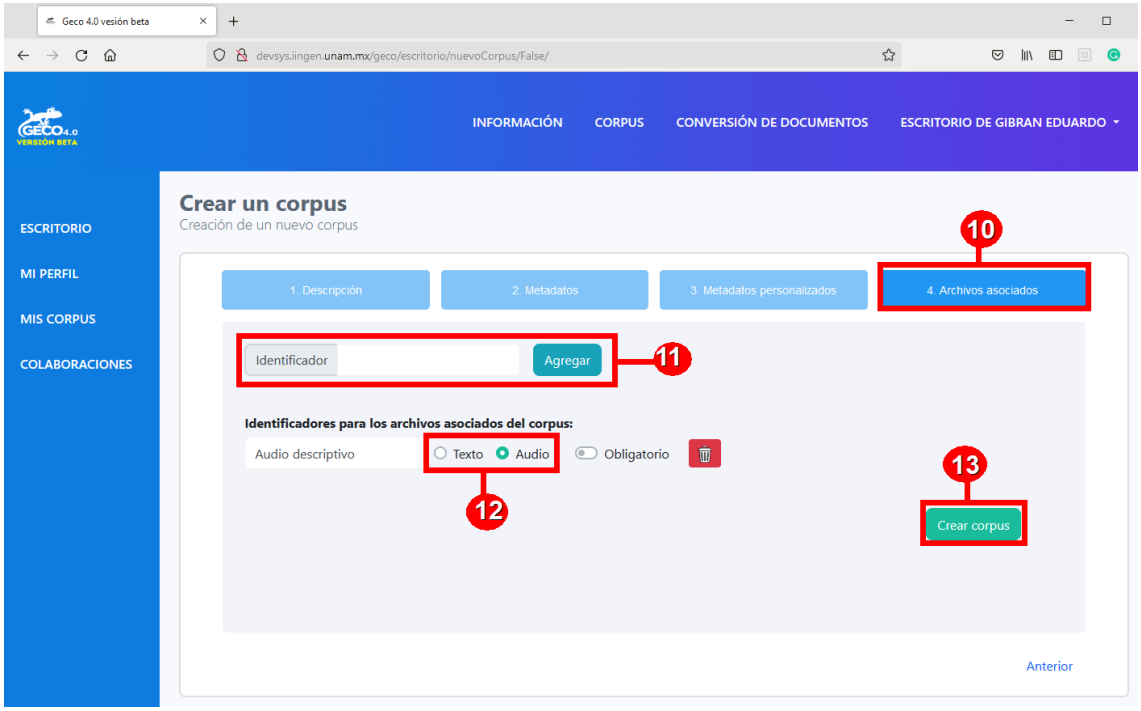


Figura 49: Pestaña de "Archivos asociados" del formulario para la creación de corpus en el sistema GECO4

7.4 Módulo de Gestión y edición de Corpus

Para la gestión de corpus hay dos pestañas en el menú lateral izquierdo, que ayudan a separar y a identificar el tipo de rol que tiene el usuario en cada uno. Primero se explicará este módulo desde el rol de propietario.

7.4.1 Gestión de corpus desde el rol de propietario

Para gestionar los corpus donde el usuario es propietario se debe de dirigir a la pestaña de "Mis corpus" (1) que se encuentra en el menú lateral izquierdo. En esta pestaña, aparecerá una lista con los corpus de los que el usuario es propietario, por cada corpus se muestran algunos detalles de este y un botón de eliminar (2) cuya función es eliminar todo el corpus del sistema, incluidos sus documentos y metadatos. Para acceder a cada uno de los corpus, el usuario deberá de dar clic sobre el nombre (3) que desee como se muestra en la Fig. 50.

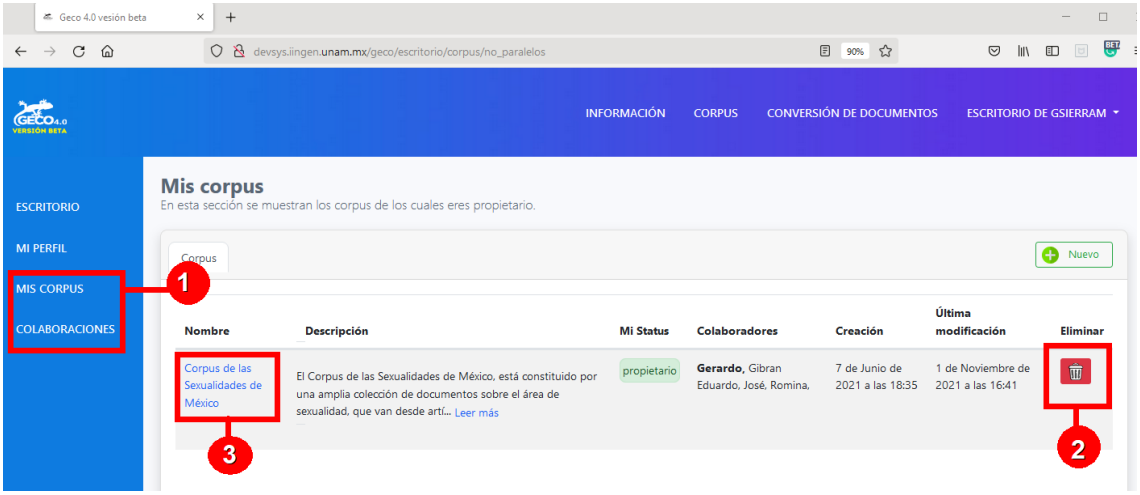


Figura 50: Se muestran los corpus que le pertenecen al usuario en el sistema GECO4

7.4.1.1 Información general

La primera pestaña que se muestra del corpus seleccionado es la de "Descripción" (4) que se muestra en la Fig. 51, cuya finalidad es dar la información general del corpus, así como las últimas modificaciones de este.

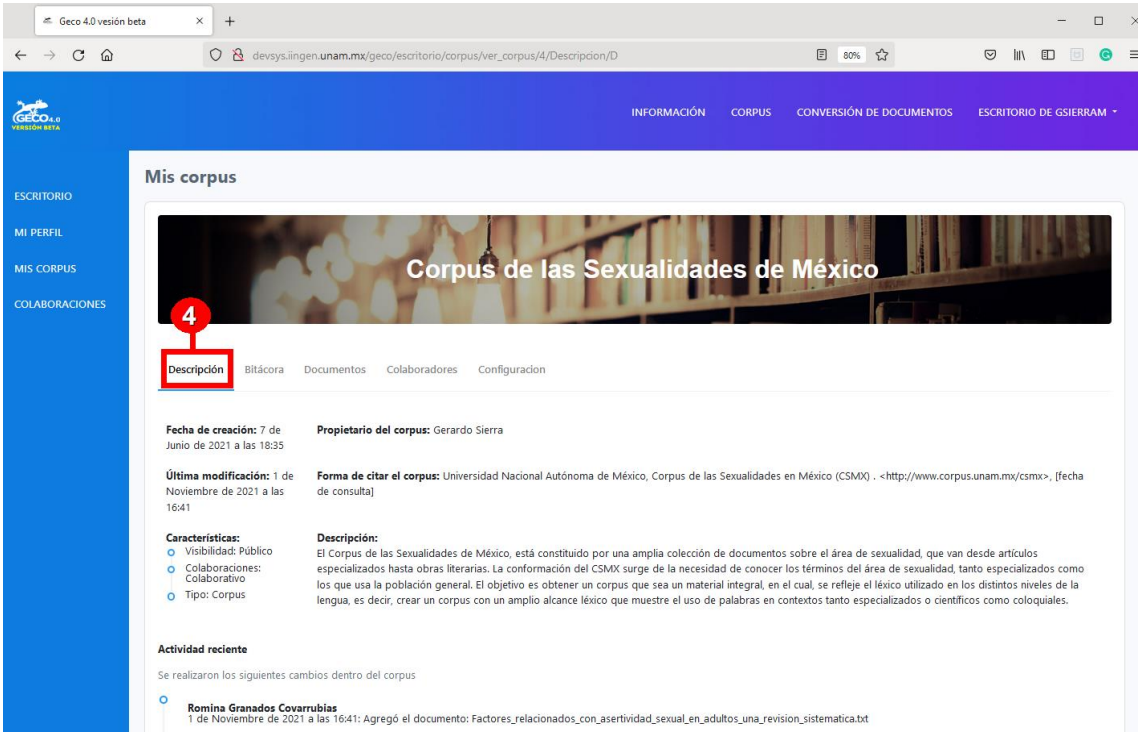


Figura 51: Pestaña de "Descripción" del corpus seleccionado en el sistema GECO4

Sin embargo, si el usuario quisiera ver todas las modificaciones que se han realizado en el corpus desde el momento de su creación deberá ir a la pestaña de “Bitácora” (5) que se muestra en la Fig. 52, ahí aparece la fecha, el tipo de modificación o actividad y el usuario que lo realizó. Esto le permite al propietario o al equipo de trabajo llevar un mejor control, pues se puede identificar rápidamente si un usuario ha eliminado, subido o modificado un documento, si se han editado los metadatos de los documentos, si se han agregado o eliminado colaboradores, entre otros.

Figura 52: Pestaña de “Bitácora” del corpus seleccionado en el sistema GECO4

7.4.1.2 Gestión de documentos

La pestaña de “Documentos” (6), le muestra al usuario una lista de todos los documentos y metadatos que conforman el corpus. También le permite al usuario gestionar documentos al eliminarlos (7), descargarlos (8) y editarlos como se muestra en la Fig. 53.

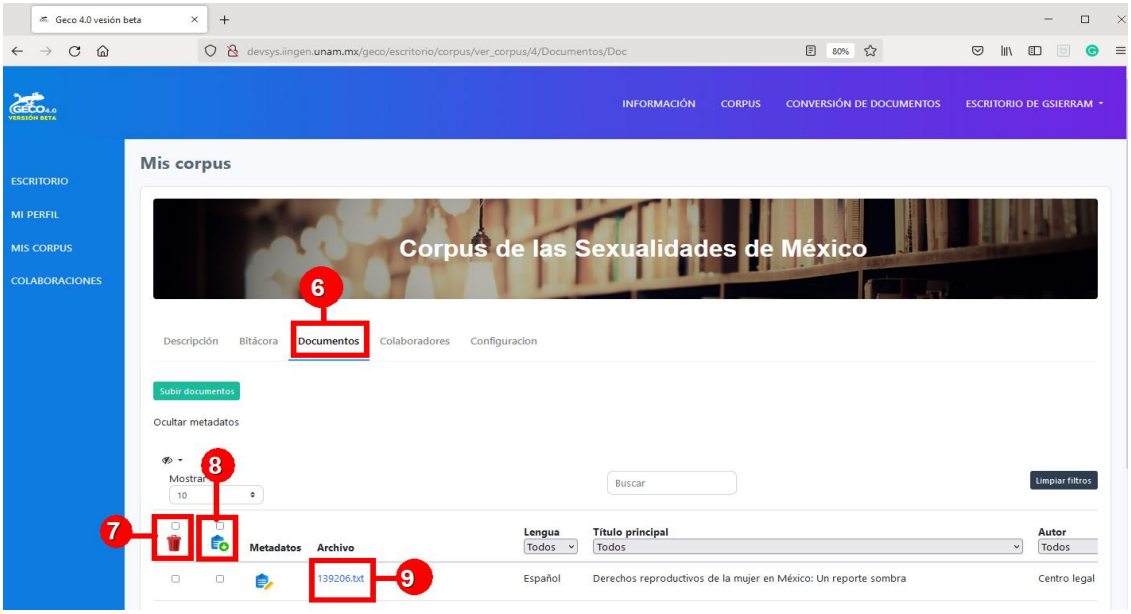


Figura 53: Pestaña de "Documentos" del corpus seleccionado en el sistema GECO4

El usuario puede ver y editar un documento al hacer clic sobre el nombre del archivo (9), en primera instancia será redirigido a una ventana de visualización del documento y del lado derecho aparecerá un botón para su edición (10) como se muestra en la Fig. 54.

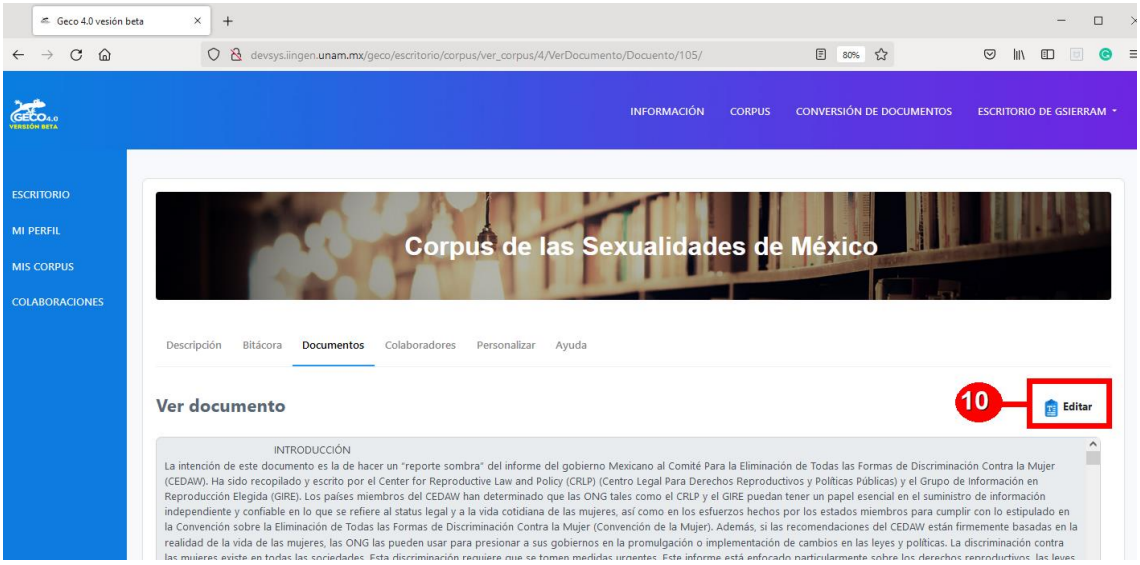


Figura 54: Visualización de un documento del corpus seleccionado en el sistema GECO4

Donde el usuario tendrá un espacio para editar el documento (11) o podrá hacer uso de un buscador de palabras para el reemplazo masivo de caracteres (12) como se muestra en la Fig. 55.

Figura 55: Edición del contenido del documento seleccionado en el sistema GECO4

Parte de la gestión de documentos también implica la carga de nuevos documentos al corpus, para eso, en la pestaña de "Documentos" el usuario encontrará un botón llamado "Subir documentos" (13) como se muestra en la Fig. 56, que se encargará de abrir una nueva ventana donde seleccionará el documento que desea agregar y los metadatos que deberá de llenar. Como se mencionó anteriormente, en la creación del corpus el propietario estableció los metadatos que serían obligatorios y los que serían opcionales, y en la ventana de subida de documentos se pueden diferenciar los metadatos obligatorios (14) de los opcionales (15), pues los primeros por convención se encuentran en una tipografía "bold" y cuentan con un asterisco previo al metadato como se muestra en la Fig. 57. El sistema de GECO4 permite la carga masiva de documentos, sin embargo, los metadatos que se establezcan tendrán los mismos valores en cada uno de ellos, a excepción del "Título principal" que tendrá como valor el mismo nombre que el documento,

también cabe mencionar que no importa el tipo de documento, pues el sistema los convierte a archivos de texto, es decir, con extensión ".txt".

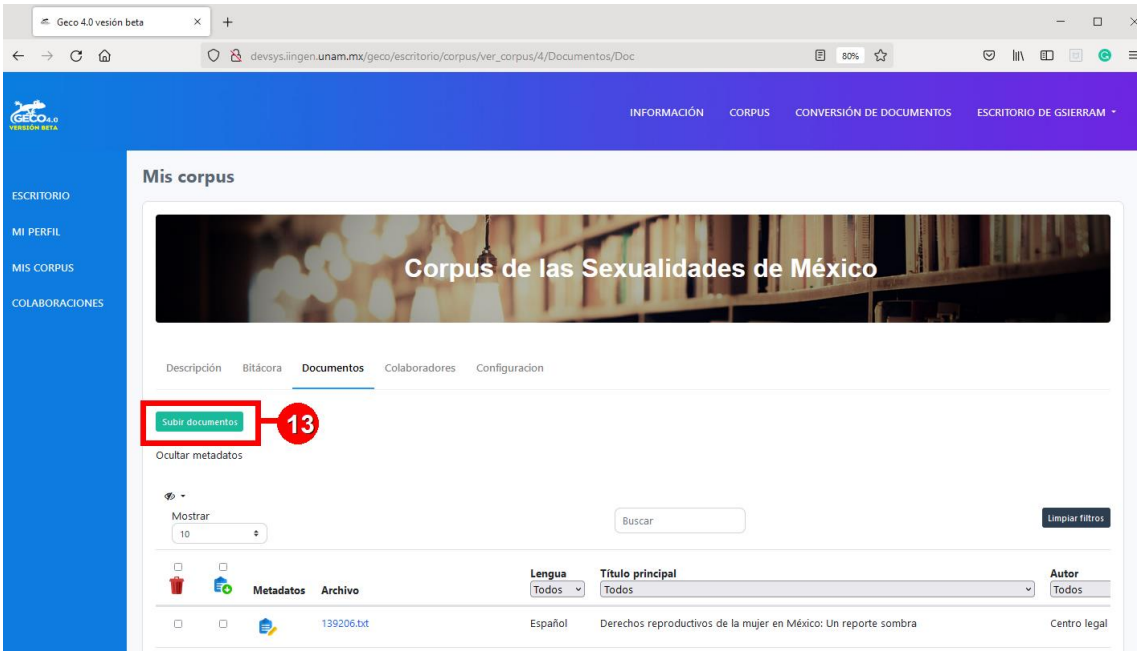


Figura 56: Proceso de la subida de documentos en el sistema GECO4

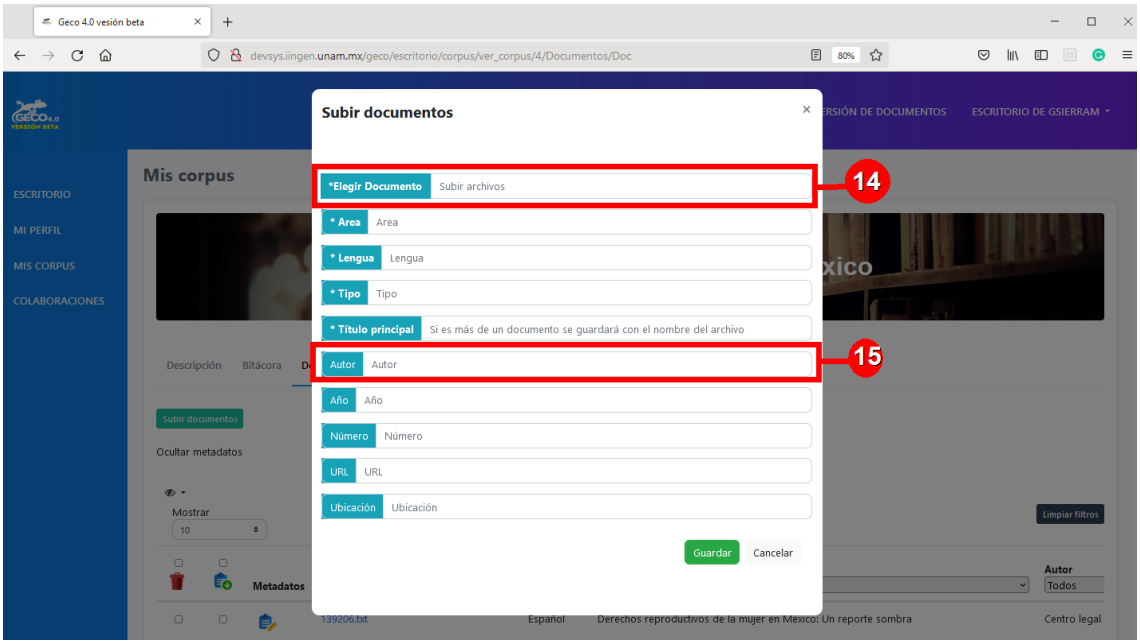


Figura 57: Formulario de subida de un documento en el sistema GECO4

7.4.1.3 Gestión de los metadatos de cada documento

El usuario también tendrá la posibilidad de editar el valor de los metadatos de cada documento del corpus, para eso deberá de presionar el símbolo de edición (16) como se muestra en la Fig. 58 y aparecerá una pestaña para su edición. Al igual que en la carga de los documentos, se marcan los metadatos obligatorios (17) con una tipografía "bold" y con un asterisco previo al metadato, para diferenciarlos de los opcionales (18) como se muestra en la Fig. 59.

Figura 58: Proceso de edición del valor de los metadatos de un documento en el sistema GECO4

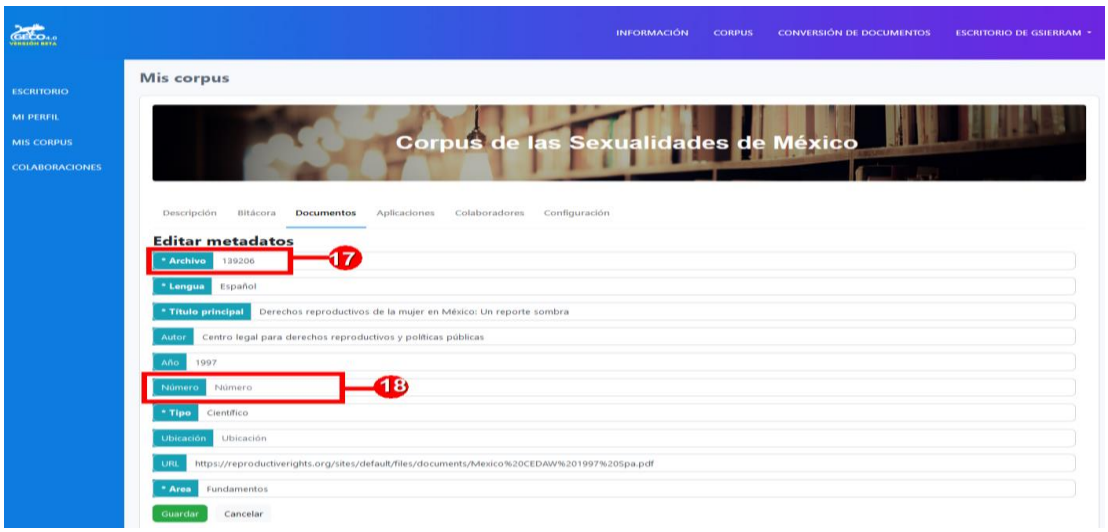


Figura 59: Formulario para la edición del valor de los metadatos de un documento

7.4.1.4 Gestión de colaboradores

En la sección de “Colaboradores” (19), el usuario tendrá la posibilidad de cambiar el tipo de trabajo del corpus (20), es decir, de colaborativo a individual y viceversa. Además, si el modo de trabajo es colaborativo podrá agregar a todos los usuarios que desee (21) o también podrá eliminarlos del flujo de trabajo (22) como se muestra en la Fig. 60.

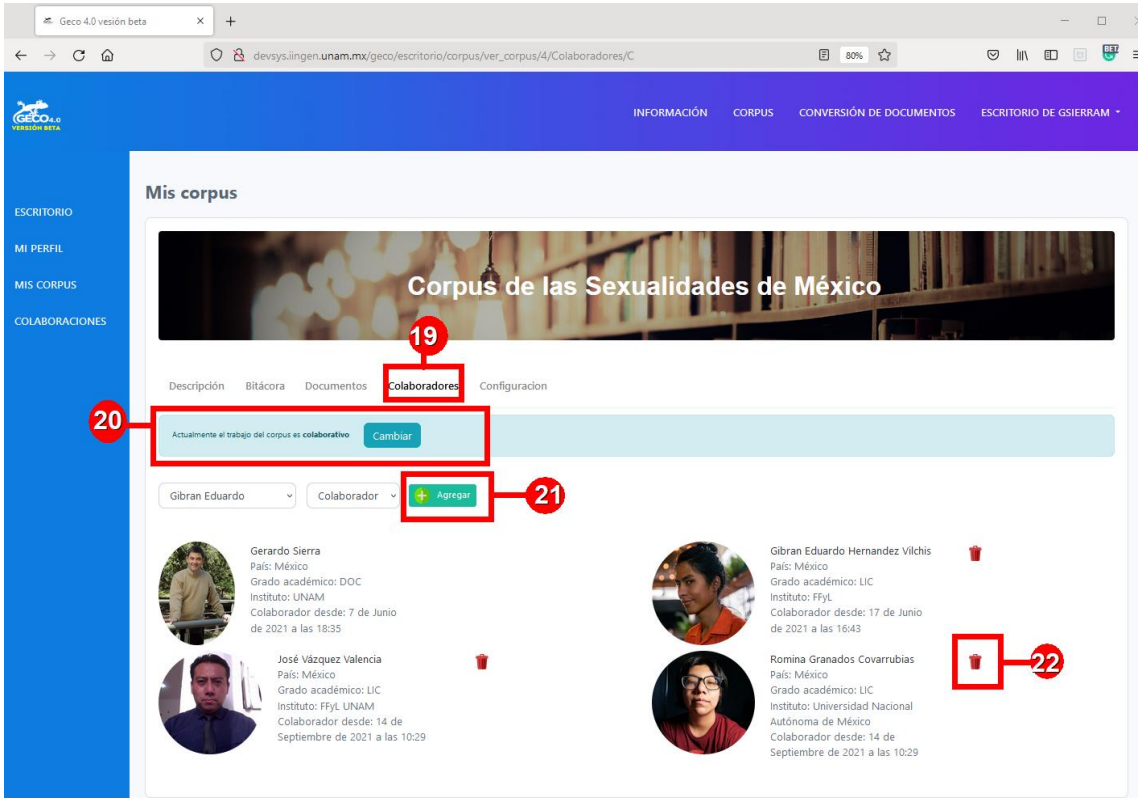


Figura 60: Proceso de gestión de colaboradores de un corpus en el sistema GECO4

7.4.1.5 Cambios en la configuración del corpus

El propietario del corpus podrá editar en cualquier momento la configuración del corpus, para ello deberá acceder a la pestaña de “Configuración” (23) en donde aparecerá un formulario igual al que se utilizó para la creación de este, por lo que el usuario podrá editar la información general (nombre, descripción, forma de citar, tipo de trabajo y tipo de publicación), agregar, quitar o modificar los metadatos del corpus, así como su obligatoriedad como se muestra en la Fig. 61.

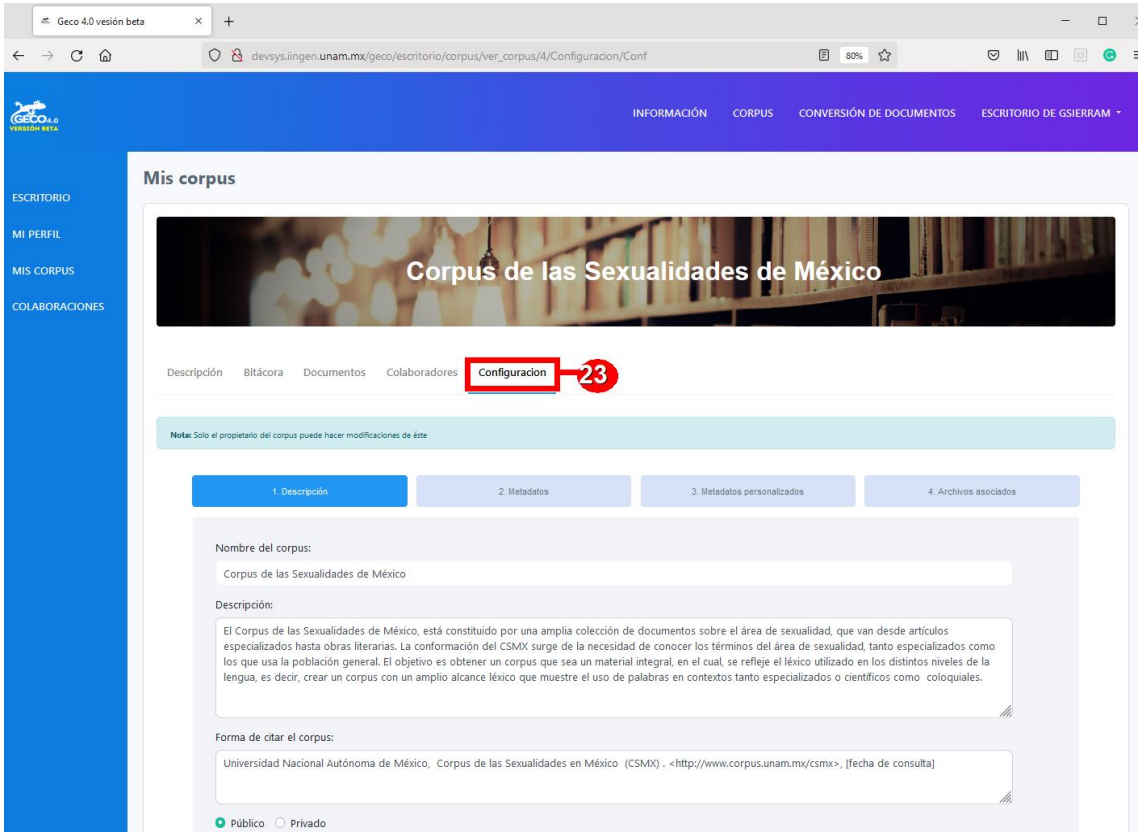


Figura 61: Formulario para la edición de la información de un corpus en el sistema GECO4

7.4.2 Gestión de corpus desde el rol de colaborador

Para gestionar los corpus donde el usuario es colaborador se debe de dirigir a la pestaña de "Colaboraciones" (24) que se encuentra en el menú lateral izquierdo como se muestra en la Fig. 62. En esta pestaña, aparecerá una lista con los corpus de los que el usuario es propietario, por cada corpus se muestran algunos detalles de este. El funcionamiento es casi el mismo que el que tiene el rol de propietario, sin embargo, el colaborador tiene algunas limitaciones, por ejemplo, no puede eliminar los corpus de donde colabora, no podrá cambiar el tipo de trabajo o eliminar colaboradores (25) y tampoco cuenta con la pestaña de configuración del corpus como se muestra en la Fig. 63.

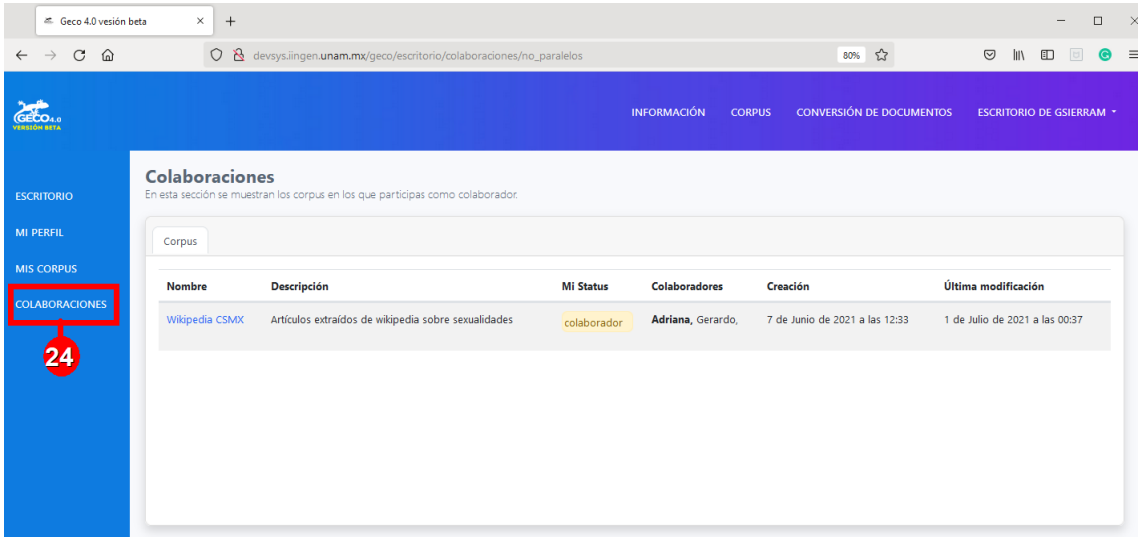


Figura 62: Se muestran los corpus donde el usuario colabora en el sistema GECO4

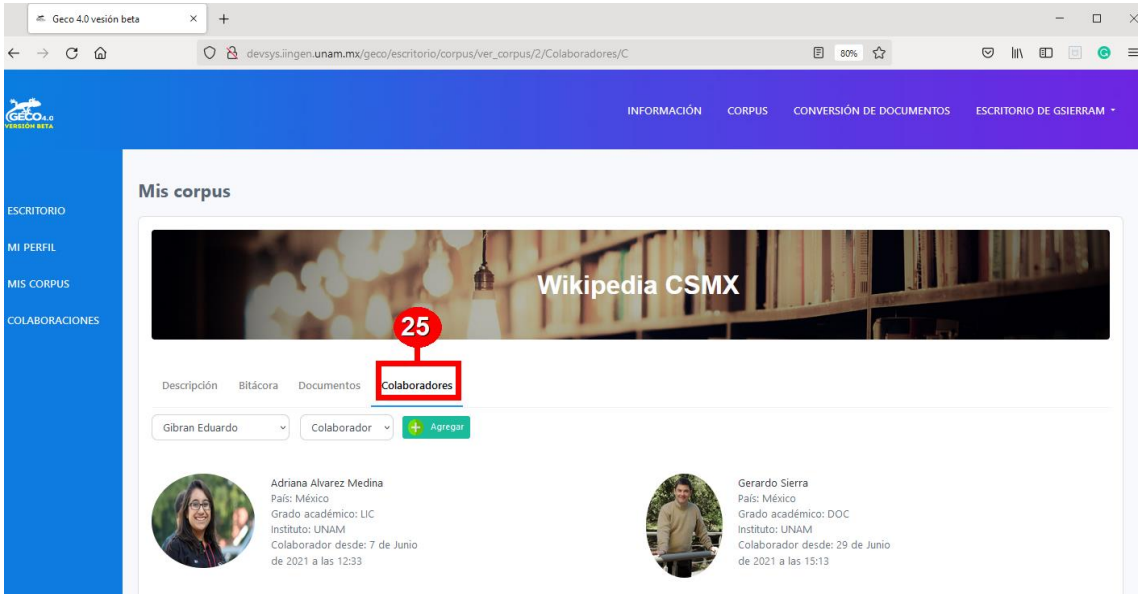


Figura 63: Proceso de gestión de colaboradores con el rol de colaborador de un corpus

Capítulo 8. Verificación y pruebas

Las pruebas en los sistemas de software son actividades que contribuyen a prevenir pérdidas y defectos de un proceso, son útiles para eliminar errores en el desarrollo, como manejo de solicitudes HTTP o validación de formularios, es por esta razón que en los diferentes módulos del sistema se realizaron pruebas para validar que los componentes funcionen correctamente.

Al término de desarrollar un nuevo elemento o función del sistema, en el tablero de Kanban se avanzaba la "tarjeta" de la columna "En progreso" a la etapa de "Revisión y pruebas" para validar que el elemento cumpla con la tarea requerida.

Las pruebas unitarias se realizan en los componentes más pequeños y validan que cada unidad de software funcione según lo esperado. Para las pruebas de unidad se utilizó el módulo de pruebas automatizadas unittest de Python y se realizaron en las clases y funciones del sistema como se muestra en la Fig. 64.

```
class testDocumento(TestCase):
    def validaDocInfo(self):
        dicDocuemntoOri = Documento.objects.filter(proyecto_id = 1)
        self.assertEqual(dicDocuemntoOri.titulo, "Corpus Literatura Mexicana")
```

Figura 64: Ejemplo de prueba unitario en modelo Documento

Las pruebas de integración validan que los componentes o funciones del software operen juntos. Después de validar los componentes unitarios se realizaron pruebas a nivel de aplicaciones, es decir validando el flujo completo de los servicios y herramientas, por ejemplo, la correcta creación de un corpus, para posteriormente agregar un documento o agregar un nuevo colaborador, por mencionar algunos ejemplos.

Las pruebas funcionales consisten en verificar los requisitos funcionales emulando un escenario de negocio, para GECO4 se realizaron pruebas al integrar el sistema al servidor de ambientes previos y principalmente se verificó el manejo correcto de solicitudes HTTP, la configuración,

conexión con la base de datos, y el resto de los servicios para comprobar el correcto funcionamiento del sistema en un ambiente semi productivo.

En las pruebas de usuario validan que tan bien un cliente puede usar un sistema o una aplicación para realizar una tarea. Las pruebas de usabilidad de GECO4 se realizó con apoyo de investigadores y colaboradores del GIL, y tuvo el objetivo de verificar que el sistema cumpliera con las reglas de negocio y necesidades de los usuarios finales. Consistieron en realizar tareas como llenar los formularios para crear un corpus, agregar documentos, agregar metadatos o compartir sus corpus con otros colaboradores. Entre los resultados el grupo de colaboradores aprobó aspectos del sistema como el control, la libertad de uso, el reconocimiento de iconos, la prevención de acciones cómo eliminar y menara de general el diseño y estética del sistema.

El proceso de pruebas que realizaron para el sistema GECO4 se muestra en la Fig. 65 y en cada etapa se validó el correcto funcionamiento, en caso contrario cuando se presentaba un error, se analizaba e identificaba el motivo de la incidencia para ser atendida en la etapa anterior correspondiente.

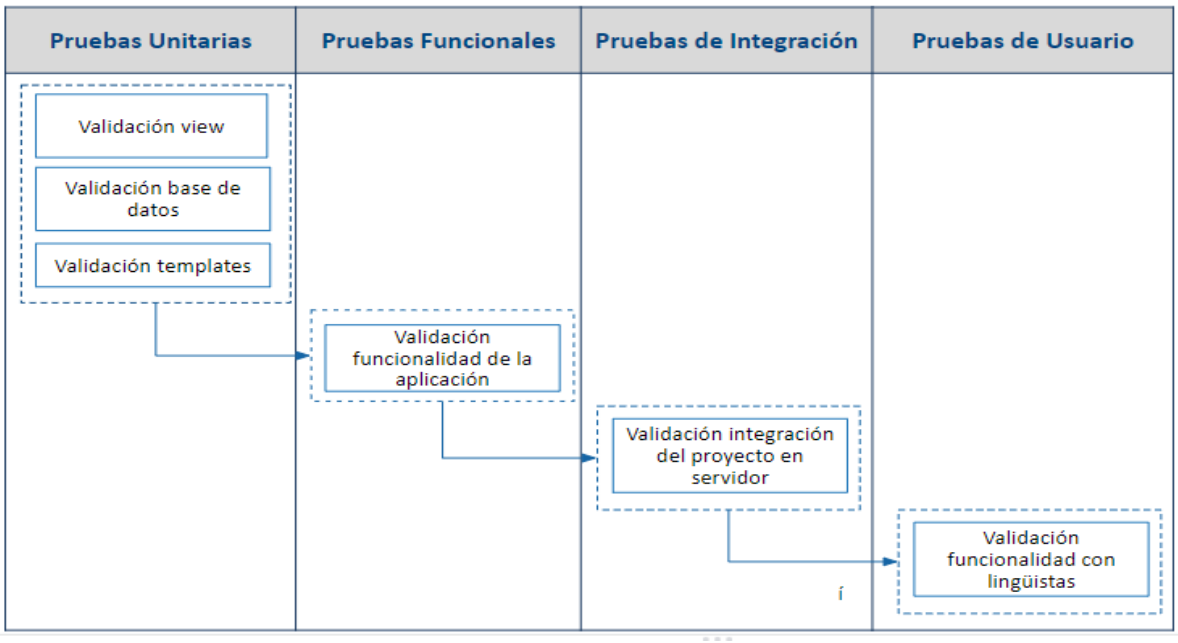


Figura 65: Diagrama de "Flujo de pruebas" del sistema GECO4

Capítulo 9. Conclusiones

El nuevo sistema GECO4 es un sistema usable de simple diseño que permite a los usuarios navegar de manera sencilla y cómoda, además es un sistema al que se puede acceder desde diferentes dispositivos electrónicos. En el presente trabajo se describió el análisis, los requerimientos, reglas de negocio, diseño e implementación de la lógica necesaria para desarrollar un sistema moderno y versátil, que unió en una sola plataforma los dos gestores de corpus que se tenían en función en el GIL de la UNAM, y dejando así una arquitectura escalable que permita la adaptabilidad de nuevas funciones.

Durante el desarrollo de GECO4 se aplicaron diferentes técnicas, procesos y conocimientos que obtuvimos a lo largo de nuestra formación académica, pues en la forma de trabajo se implementaron metodologías ágiles que optimizaron la organización y flexibilidad del proceso. La implementación de algoritmos, paradigmas de programación y estructuras de datos, nos permitió generar un código eficiente y modular que permite el escalamiento del sistema, haciendo más transparente la modificación y creación de nuevos módulos, sin que esto impacte en su desarrollo previo o posterior.

El diseño y gestión de bases de datos, nos brindó el conocimiento necesario para la reestructuración de su base de datos, lo que permitió que el sistema tuviera mejor velocidad de procesamiento y transferencia de datos de los corpus gestionados, pues su nuevo esquema definió la implementación de las estructuras de programación empleadas, lo cual tiene un impacto directo en la complejidad de la manipulación de los datos. Resolviendo así, las limitaciones de control y procesamiento que existían en los sistemas anteriores. Este nuevo sistema de gestión de corpus mejora la usabilidad y experiencia del usuario al llevar un control más fino y eficiente a través del flujo de pantallas, de modo que, al ser más natural, ayuda a

evitar errores de captura y persistencia de la información, permitiendo a los usuarios navegar de manera sencilla y cómoda, además es un sistema al que se puede acceder desde diferentes dispositivos electrónicos.

Finalmente, la nueva función de crear metadatos permite agregar archivos multimedia o recursos auxiliares de texto y audio al corpus, lo que brinda mayores facilidades al estudio e investigación de uno o más ni-veles de representación lingüística.

GECO4 sigue siendo enriquecido con nuevas funcionalidades, actualmente se encuentra en desarrollo la incorporación de corpus paralelos, así como de las herramientas de análisis léxico y terminológico. Con lo cual se pretende que este nuevo gestor de corpus sea un recurso de gran importancia en la práctica de la terminología, pues integra la información almacenada de los corpus, permitiendo observar el cambio semántico de las palabras, y así llevar a cabo un análisis más exacto y minucioso de los nuevos términos que se van acuñando y de sus significados inherentes.

Referencias

- Bolaños, S. (2015). La lingüística de corpus: perspectivas para la investigación lingüística contemporánea. *Forma y Función*, *28*(1), 31-54. doi: 10.15446/fyf.v28n1.51970
- Sommerville, I. (2010). Software Engineering (9.a ed.). Addison-Wesley.
- Aguilar, L. J. (1998). Programación orientada a objetos (2.a ed.). McGraw-Hill Education.
- Sierra, F. J. C. (2003). Java TM 2 (2.a ed.). Alfaomega.
- Villalobos, G. M., Sánchez, G. D. C., & Gutiérrez, D. A. B. (2010). Diseño de framework web para el desarrollo dinámico de aplicaciones. *Scientia et technica*, *16*(44), 178-183.
- Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). *Elements of reusable object-oriented software* (Vol. 99). Reading, Massachusetts: Addison-Wesley.
- Louden, K. C., & Lambert, K. A. (2011). Programming Languages: Principles and Practices (3rd ed.). Cengage Learning.
- Hari, S. (2020). *Programming Paradigms: A must know for all Programmers*. Hackr.io. https://hackr.io/blog/programming-paradigms
- Vincent, W. S. (2018). Django for Beginners: Build websites with Python and Django (English Edition). WelcomeToCode.
- Chacon, S. (2009). Pro Git (1.a ed.). Apress.
- Haverbeke, M. (2018). Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming (3.a ed.). No Starch Press.
- Guzman, H. C. (2018, 25 octubre). Patrón MVT: Modelo-Vista-Template | Curso de Django | Hektor Profe. HektorDocs. Recuperado 7 de enero de 2022, de https://docs.hektorprofe.net/django/web-personal/patron-mvt-modelo-vista-template/
- Lucidchart. (s. f.). Qué es el lenguaje unificado de modelado (UML). Recuperado 10 de enero de 2022, de https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml/

- Camacho, D. (2021, 25 agosto). Qué es GitHub y cómo usarlo para aprovechar sus beneficios. Platzi. Recuperado 10 de enero de 2022, de https://platzi.com/blog/que-es-github-como-funciona/
- Departamento de Ciencias de la Computación, Universidad de Chile. (2008). Cómo funciona la web (1.a ed) [Libro electrónico]. Recuperado el 15 de junio de 2021
- Vincent, W. S. (2020). Django for Beginners: Build websites with Python and Django. WelcomeToCode.
- Hallebeek, J. (s. f.). *El Corpus paralelo*. Universidad de Nijmegen, Departamento de Español. Recuperado 24 de mayo de 2022, de http://www.corpus.unam.mx/cursocorpus/24-articulo5.pdf

Apéndice A.

A continuación, a manera de ejemplo, se presentan los modelos, el template y la vista desarrollada para la sección de información en GECO4

A.1 Modelos

```
class Proyecto(models.Model):
    nombre = models.CharField(max_length=128, unique=True,
                    verbose_name='nombre del proyecto',
                    help_text='nombre del proyecto para el corpus')
    descripcion = models.TextField(verbose_name='descripción',
                    help_text='breve descripción del proyecto')
    cita = models.TextField(null=True, verbose_name='cita',
                    help_text='Forma de citar el proyecto')
    colaboradores = models.ManyToManyField('UsuarioGeco',
                        through='Colaborador_proyecto',
                    related_name='proyectos')
    metadatos = models.ManyToManyField('Metadato', through='Proyecto_metadato',
                    related_name='proyectos')
    aplicaciones = models.ManyToManyField('Aplicacion',
                    through='Proyecto_aplicacion', related_name='proyectos')
    repositorio = models.CharField(max_length=256, editable=False, unique=True,
                    default=ceateRepositorioName)
    activo = models.BooleanField(default=True,
                    help_text='habilitar para indicar que el proyecto está activo')
    es_publico = models.BooleanField(default=False,
                   help_text='habilitar para indicar que el proyecto es público
                    (privado en caso contrario)')
    es_paralelo = models.BooleanField(default=False, help_text='habilitar para
                    indicar que será un corpus paralelo')
    es_colaborativo = models.BooleanField(default=False, help_text='habilitar para
                    indicar que será un corpus colaborativo')
    creacion = models.DateTimeField(auto_now_add=True, editable=False)
    fecha_borrado = models.DateTimeField(auto_now_add=False, editable=False,
                    null=True, blank=True)

    class Meta:
        ordering = ["nombre"]
        verbose_name = 'proyecto'
        verbose_name_plural = 'proyectos'
        indexes = [
            models.Index(fields=['nombre']),
        ]

    def save(self, *args, **kwargs):
```

```
        self.nombre = self.nombre.strip()
        super().save(*args, **kwargs)

    def __str__(self):
        return self.nombre

    def getColaboradores(self):
        res = []
        for ca in self.colaborador_atributos.all():
            if ca.activo == 1:
                res += [(ca.tipo_colaborador.id,
                     ca.tipo_colaborador.tipo, ca.colaborador, ca)]
        return res

    def propietario(self):
        tipo_propietario = Tipo_colaboracion.objects.get(tipo='propietario')
        for c in Colaborador_proyecto.objects.filter(proyecto=self):
             if c.tipo_colaborador == tipo_propietario:
                 return c.colaborador.usuario
        return '¿desconocido?'


    def getMetadatos(self):
        res = []
        for m in self.metadatos_proyecto.all():
            res += [m.metadato]
        return res

class Documento(models.Model):
    proyecto = models.ForeignKey('Proyecto', on_delete = models.CASCADE,null=False)
    titulo = models.CharField(max_length= 128, unique = False, help_text = 'Titulo
             principal del documento')
    archivo = models.CharField(max_length = 128, help_text = 'archivo donde se
             encuentra el documento')
    archivo_etiquetado = models.CharField(max_length = 128, blank=True, null=True,
             help_text = 'archivo donde se encuentra el documento etiquetado')
    agrupamiento = models.IntegerField(default=0)
    borrado = models.BooleanField(default=False,help_text='¿El documento ha sido
             enviado a papelera?')
    responsable_creacion =
             models.ForeignKey('UsuarioGeco',related_name='responsable_creacion',on_
             delete=models.SET_NULL,null=True)
    fecha_creacion = models.DateTimeField(verbose_name='fecha de creacion')
    responsable_modificacion =
models.ForeignKey('UsuarioGeco',related_name='responsable_modificacion',on_delete=mo
dels.SET_NULL,null=True)
    fecha_modificacion = models.DateTimeField(verbose_name='fecha de
modificacion',null=True,blank=True)
```

```
    responsable_borrado =
models.ForeignKey('UsuarioGeco',related_name='responsable_borrado',on_delete=models.
SET_NULL,null=True)
    fecha_borrado = models.DateTimeField(verbose_name='fecha de
borrado',null=True,blank=True)
    check_sum = models.IntegerField(verbose_name='código
verificador',default=0,editable=False)
    metadatos =
models.ManyToManyField('Proyecto_metadato',through='Documento_metadato',related_name
='documentos')

    class Meta:
        ordering = ["proyecto","titulo"]
        verbose_name = 'documento'
        verbose_name_plural = 'documentos'
        constraints = [models.UniqueConstraint(fields=["titulo", "proyecto"],
name='unique_file')]
        indexes = [
            models.Index(fields=['proyecto','titulo']),
        ]
    def save(self, *args, **kwargs):
        self.titulo = self.titulo.strip()
        super().save(*args, **kwargs)

    def __str__(self):
        return f"{self.proyecto} {self.titulo}"

    def getDocInfo(self):
        datosMetadatos = []
        archivosAsociadosAudio = []
        archivosAsociadosTXT = []
        for md in self.metadatos_documentos.all():
            consulta = Metadato.objects.get(id = md.proyecto_metadato.metadato.id)
            if consulta.es_multimedia:
                archivoAsociado = Metadato_multimedia.objects.get(documento = self,
proyecto_metadato = md.proyecto_metadato, nombre = md.valor)
                if consulta.es_texto:
                    archivosAsociadosTXT.append({'ruta':archivoAsociado.ruta,
'nombre':md.valor})
                elif consulta.es_audio:
                    archivosAsociadosAudio.append({'ruta':archivoAsociado.ruta,
'nombre':md.valor})

        for md in self.metadatos_documentos.all():
            if md.valor == '0': #Para los que son catalogo
                consulta = Metadato_opcion.objects.get(id=md.indice)
                datosMetadatos.append(consulta.opcion)
            else:
```

```
                consulta = Metadato.objects.get(id =
md.proyecto_metadato.metadato.id)
                if consulta.es_multimedia == 0: #Para los que no son catalogo y
tampoco son multimedia/archivos asociados
                    datosMetadatos.append(md.valor)

        dic = {
            "documentoPrincipal": self.titulo,
            "archivosAsociadosTXT": archivosAsociadosTXT,
            "archivosAsociadosAudio": archivosAsociadosAudio,
            "datosMetadatos": datosMetadatos,
            "link": self.archivo,
            "idDoc": self.id,
            "idCorpus": self.proyecto.pk
        }

        return dic

    def getDocInfoRemasterizado(self):
        datosMetadatos = []
        archivosAsociadosTXT = []
        archivosAsociadosAudio = []
        metaTextoArr = Documento_metadato.objects.filter(documento = self,
proyecto_metadato__metadato__es_texto = 1).order_by('proyecto_metadato_id')
        for metaTexto in metaTextoArr:
            rutaTexto = Metadato_multimedia.objects.get(proyecto_metadato =
metaTexto.proyecto_metadato, documento = self)
            archivosAsociadosTXT.append({
                'ruta': rutaTexto.ruta,
                'valor': metaTexto.valor,
                'proyecto_metadato_id': metaTexto.proyecto_metadato.id
            })
        metaAudioArr = Documento_metadato.objects.filter(documento = self,
proyecto_metadato__metadato__es_audio = 1).order_by('proyecto_metadato_id')
        for metaAudio in metaAudioArr:
            rutaAudio = Metadato_multimedia.objects.get(proyecto_metadato =
metaAudio.proyecto_metadato, documento = self)
            archivosAsociadosAudio.append({
                'ruta': rutaAudio.ruta,
                'valor': metaAudio.valor,
                'proyecto_metadato_id': metaAudio.proyecto_metadato.id
            })
        datosMetadatos = self.getRengloMetadatos()
        diccionarioMetadatos = {
            "documentoPrincipal": self.titulo,
            "archivosAsociadosTXT": archivosAsociadosTXT,
            "archivosAsociadosAudio": archivosAsociadosAudio,
            "datosMetadatos": datosMetadatos,
```

```
            "link": self.archivo,
            "idDoc": self.id,
            "idCorpus": self.proyecto.pk
        }
        return diccionarioMetadatos

    def getRengloMetadatos(self,no_disponible='valor desconocido'):
        p = self.proyecto
        metadatos = p.metadatos.all().order_by('id')
        metas = []
        for m in metadatos:
            if m.es_texto:
                metas.append(m)
            elif m.es_audio:
                metas.append(m)
            elif m.es_general:
                metas.append(m)
            elif not m.es_general:
                metas.append(m)
        dic = {}
        for x in Documento_metadato.objects.filter(documento=self):
            dic[x.proyecto_metadato.metadato] =
{'valor':x.valor,'proyecto_metadato_id':x.proyecto_metadato.id}
        ren = []
        for m in metas:
            dc = {}
            if m in dic.keys():
                dc['proyecto_metadato_id'] = dic[m]['proyecto_metadato_id']
                dc['valor'] = dic[m]['valor']
            else:
                dc['proyecto_metadato_id'] =
Proyecto_metadato.objects.get(metadato=m, proyecto=p).id
                dc['valor'] = no_disponible
            ren.append(dc)
        return ren

    def getActualizar(self):
        res = []
        for md in self.metadatos_documentos.all().order_by('proyecto_metadato_id'):
            if md.valor == '0':
                consulta = Metadato_opcion.objects.get(id=md.indice)
                res.append(consulta.opcion)
            else:
                res.append(md.valor)

        dic = {
            "meta": res,
            "archivo": self.titulo
```

```
        }

        return dic

class UsuarioGeco(models.Model):
    OPCIONES_GRADO = (('', 'Elegir...'), ('LIC','Licenciatura'),
('ESP','Especialización'), ('MBA','Mestría'), ('DOC','Docotado'),
('POS','Postdoctorado'), ('NONE','OTRO'),)
    usuario = models.OneToOneField(User, on_delete=models.CASCADE)
    nombre = models.CharField(max_length=128, unique=False)
    apellidos = models.CharField(max_length=128, unique=False)
    grado_academico = models.CharField(max_length=128, unique=False,
choices=OPCIONES_GRADO)
    institucion = models.CharField(max_length=128, unique=False)
    foto_perfil = models.ImageField(default='fotosPerfil/generic.png',null=True,
blank=True, upload_to='fotosPerfil/')
    pais = models.ForeignKey('PaisUsuario', on_delete=models.CASCADE, default=13,
null=True)
    class Meta:
        ordering = ["usuario"]
        verbose_name = 'Usuario GeCo'
        verbose_name_plural = 'Usuarios GeCo'
        constraints = [
            models.UniqueConstraint(
                fields=["nombre", "apellidos"], name='unique_nombre_and_apellido'),
        ]
    def getNombreCompleto(self):
        return '{} {} {}'.format(self.grado_academico,self.nombre,self.apellidos)
    def __str__(self):
        return '{} ({})'.format(self.usuario,self.getNombreCompleto())

    def getProyectos(self):
        res = []
        for ca in self.colaborador_atributos.all():
            res += [{'tipo_id': ca.tipo_colaborador.id,
                     'tipo_colaboracion': ca.tipo_colaborador.tipo, 'proyecto':
ca.proyecto}]
        return res

    def num_proyectos(self):
        return len(list(self.colaborador_atributos.all()))
    def email(self):
        return self.usuario.email
```

A.2 Templates

A.2.1 Información.html

```
{% extends "base_generic.html" %}

{% block title-content %}
<!-- Masthead Heading-->
<h1 class="masthead-heading mb-0">Información</h1>
<!-- Masthead Subheading-->
<p class="masthead-subheading font-weight-light mb-0">En esta sección encontrarás
estadísticas sobre todo el contenido
  del Sistema de Gestión de Corpus del Grupo de Ingeniería lingüística de la
UNAM</p>
</div>
{% endblock %}

{% block content %}
<br><br>
<h1 class="page-section-heading text-left text-secondary mb-0">Actualmente en el
Gestor de Corpus tenemos</h1>
<br><br>
<div class="row">
  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Total de usuarios: {{ num_usuarios }}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Número total de
usuarios registrados en GECO4.0</p>
          </div>
        </div>
        <div class="col-md-4 media">
          {% load static %}
          <img src="{% static 'img/usuarios.png' %}" width="100" height="100"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Corpus: {{num_no_paralelos}}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Conjunto amplio y
estructurado de textos, creado
              independientemente de sus posibles formas o usos.</p>
          </div>
        </div>
        <div class="col-md-4 media">
```

```
          {% load static %}
          <img src="{% static 'img/no_paralelos.png' %}" width="70" height="70"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Documentos: {{num_documentos}}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Número total de
documentos registrados en GECO4.0</p>
          </div>
        </div>
        <div class="col-md-4 media">
          {% load static %}
          <img src="{% static 'img/documentos.png' %}" width="70" height="70"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>
</div>

<br><br>
{% endblock %}
```

A.2.2 base_generic.html

```
<!DOCTYPE html>
<html lang="en">

<head>
  {% block title %}
  <title>Geco 4.0 versión beta</title>
  {% load static %}
  <link rel="icon" type="image/x-icon" href="{% static 'img/salamander32.png' %}" />
  {% endblock %}

  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/4.7.0/css/font-awesome.min.css">
  <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
    integrity="sha384-
TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2"
crossorigin="anonymous">
  {% load static %}
  <link rel="stylesheet" href="{% static 'css/styles.css' %}">
  <link rel="stylesheet" href="{% static 'css/vendor.bundle.addons.css' %}">
```

```
  <link rel="stylesheet" href="{% static 'css/vendor.bundle.base.css' %}">
</head>

<body>
  {% load static %}
  <nav class="navbar navbar-expand-lg bg-secondary text-uppercase fixed-top"
id="mainNav"
    style="background-image: url(' {% static 'img/background.jpg' %} ');">
    <a class="navbar-brand js-scroll-trigger" href="{% url 'index' %}">
      {% load static %}
      <img src="{% static 'img/salamander43.png' %}" width="75rm" height="65rm">
    </a>
    <button class="navbar-toggler navbar-toggler-right text-uppercase font-weight-
bold bg-primary text-white rounded"
      type="button" data-toggle="collapse" data-target="#navbarResponsive" aria-
controls="navbarResponsive"
      aria-expanded="false" aria-label="Toggle navigation">
      Menú
      <i class="fas fa-bars"></i>
    </button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger"
            href="{% url 'informacion' %}">Información</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger"
            href="{% url 'no_paralelos' %}">Corpus</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger"
            href="{% url 'conversion-documentos' %}">Conversión de
Documentos</a></li>
        {% if user.is_authenticated %}
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger"
            href="{% url 'escritorio' %}">Escritorio de {{ user.get_username
}}</a></li>
        {% else %}
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger"
            href="{% url 'registro' %}">Registrar Usuario</a></li>
        <li class="nav-item mx-0 mx-lg-1"><a class="nav-link py-3 px-0 px-lg-3
rounded js-scroll-trigger alerta"
            data-iniciosesion="{% url 'login'%}?next={% url 'escritorio' %}"
href="#">Iniciar Sesión</a></li>
        {% endif %}
      </ul>
    </div>
  </nav>


  {% load static %}
  <header class="masthead text-white text-center" style="background-image: url(' {%
static 'img/background.jpg' %} ');">
    <div class="container d-flex align-items-center flex-column">
      {% block title-content %}
      <!--- TÍTULO Y DESCRIPCIÓN DE CADA PÁGINA --> {% endblock %}
    </div>
  </header>

  <div class="margen">
    {% block content %}
```

```
    <!--- CONTENIDO DE CADA PÁGINA --> {% endblock %}
  </div>
  {% block pagination %}
  {% if is_paginated %}
  <div class="pagination">
    <span class="page-links">
      {% if page_obj.has_previous %}
      <a href="{{ request.path }}?page={{ page_obj.previous_page_number
}}">anterior</a>
      {% endif %}
      <span class="page-current">
        Página {{ page_obj.number }} de {{ page_obj.paginator.num_pages }}.
      </span>
      {% if page_obj.has_next %}
      <a href="{{ request.path }}?page={{ page_obj.next_page_number
}}">siguiente</a>
      {% endif %}
    </span>
  </div>
  {% endif %}
  {% endblock %}

  {% block footer %}
  <footer class="footer text-center">
    <div class="container">
      <div class="row">
        <!-- Footer Location-->
        <div class="col-lg-4 mb-5 mb-lg-0">
          <h4 class="text-uppercase mb-4">Ubicación</h4>
          <p class="lead mb-0">
            Instituto de Ingeniería UNAM, Circuito Escolar s/n,
            Ciudad Universitaria, Delegación Coyoacán
          </p>
        </div>
        <!-- Footer Social Icons-->
        <div class="col-lg-4 mb-5 mb-lg-0">
          <h4 class="text-uppercase mb-4">Redes Sociales</h4>
          <a class="btn btn-outline-light btn-social mx-1"
href="https://www.facebook.com/ingenieriaLinguistica/">
            {% load static %}
            <img class="iconFooter" src="{% static 'img/facebook.png' %}">
          </a>
          <a class="btn btn-outline-light btn-social mx-1"
href="https://twitter.com/gil_unam">
            {% load static %}
            <img class="iconFooter" src="{% static 'img/twitter.png' %}">
          </a>
        </div>
        <!-- Footer About Text-->
        <div class="col-lg-4">
          <h4 class="text-uppercase mb-4">Contacto</h4>
          <p class="lead mb-0">
            gil@iingen.unam.mx
            <br> +52(55)5623-3600 ext. 8808
          </p>
        </div>
      </div>
    </div>
  </footer>
  {% endblock %}
```

```
  <!-- Copyright Section-->

  <div class="copyright py-4 text-center text-white">
    <div class="container"><small>©Todos los derechos reservados UNAM 2020. Esta
página puede ser reproducida con fines
        no lucrativos, siempre y cuando no se mutile, se cite la fuente completa y
su dirección electrónica. De otra forma requiere
        permiso previo por escrito de la institución.
      </small>
    </div>
    {% block copyright %}
    {% endblock %}
    </div>
  </div>

  <!-- Scroll to Top Button (Only visible on small and extra-small screen sizes)-->
  <div class="scroll-to-top d-lg-none position-fixed">
    <a class="js-scroll-trigger d-block text-center text-white rounded" href="#page-
top"><i
        class="fa fa-chevron-up"></i></a>
  </div>


  {% block js %}
  <!-- Bootstrap core JS-->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.bundle.min.js">
</script>
  <!-- Third party plugin JS-->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
easing/1.4.1/jquery.easing.min.js"></script>
  <!-- Core theme JS-->
  {% load static %}
  <script src="{% static 'js/scripts.js' %}"></script>
  <script src="https://cdn.jsdelivr.net/npm/sweetalert2@11"></script>
  {% endblock %}

</body>
</html>
```

A.2.3 información.html

```
{% extends "base_generic.html" %}

{% block title-content %}
<!-- Masthead Heading-->
<h1 class="masthead-heading mb-0">Información</h1>
<!-- Masthead Subheading-->
<p class="masthead-subheading font-weight-light mb-0">En esta sección encontrarás
estadísticas sobre todo el contenido
  del Sistema de Gestión de Corpus del Grupo de Ingeniería lingüística de la
UNAM</p>
```

```
</div>
{% endblock %}

{% block content %}
<br><br>
<h1 class="page-section-heading text-left text-secondary mb-0">Actualmente en el
Gestor de Corpus tenemos</h1>
<br><br>
<div class="row">
  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Total de usuarios: {{ num_usuarios }}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Número total de
usuarios registrados en GECO4.0</p>
          </div>
        </div>
        <div class="col-md-4 media">
          {% load static %}
          <img src="{% static 'img/usuarios.png' %}" width="100" height="100"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Corpus: {{num_no_paralelos}}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Conjunto amplio y
estructurado de textos, creado
              independientemente de sus posibles formas o usos.</p>
          </div>
        </div>
        <div class="col-md-4 media">
          {% load static %}
```

```
          <img src="{% static 'img/no_paralelos.png' %}" width="70" height="70"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>

  <div class="col-md-6 col-lg-3 mb-5 ">
    <div class="card" style="max-width: 25rem; min-height: 11rem;">
      <div class="row no-gutters">
        <div class="col-md-8">
          <div class="card-body">
            <h5 class="card-title">
              Documentos: {{num_documentos}}
            </h5>
            <p class="masthead-subheading font-weight-light mb-0">Número total de
documentos registrados en GECO4.0</p>
          </div>
        </div>
        <div class="col-md-4 media">
          {% load static %}
          <img src="{% static 'img/documentos.png' %}" width="70" height="70"
class="align-self-center mr-3">
        </div>
      </div>
    </div>
  </div>
</div>

<br><br>
{% endblock %}
```

A.3 Vista

A.3.1 proyectos_views.py

```
…
def informacion(request):

    numeroUsuarios = UsuarioGeco.objects.count()
    numeroDocumentos = Documento.objects.filter(borrado=False).count()
    numeroParalelos = Proyecto.objects.filter(es_paralelo=True, activo =
True).count()
    numeroNoParalelos = Proyecto.objects.filter(es_paralelo=False, activo =
True).count()
```

```
    # Number of visits to this view, as counted in the session variable.
    numeroVisitas = request.session.get('num_visitas', 0)

    request.session['num_visitas'] = numeroVisitas+1
    context = {
        'num_usuarios' : numeroUsuarios,
        'num_documentos' : numeroDocumentos,
        'num_paralelos' : numeroParalelos,
        'num_no_paralelos': numeroNoParalelos,
        'num_visitas' : numeroVisitas
    }

    setSessionProyectoActual(request,context)

    return render(request, 'informacion.html',context=context)
…
```