Universidad de la República
Facultad de Ingeniería
Instituto de Computación
Uruguay

Plataforma de Análisis Documental: “Rutas de la Memoria Histórica”

Sofía Barreiro
Virginia Bacigalupe

Proyecto de Grado
Ingeniería en Computación
Universidad de la República

Montevideo, Uruguay, Mayo de 2022

Tutores: Dr. Ing. Daniel Calegari
MSc. Ing. Fernando Carpani
Universidad de la República

Resumen

El proyecto CRUZAR[1] tiene como objetivo el ordenamiento, la clasificación y el análisis de documentos de la última dictadura militar en Uruguay (1973-1985).

Una de las etapas del proceso de clasificación consiste en el etiquetado de documentos digitalizados, realizado por estudiantes y docentes de la Facultad de Información y Comunicación, Udelar (FIC), sin conocimientos informáticos. Sin embargo, la herramienta de etiquetado de documentos que utilizan al día de hoy no es intuitiva ni eficiente en diferentes aspectos.

El objetivo de este trabajo es el desarrollo de una plataforma web que remplace dicha herramienta, mejorando la experiencia de usuario e incorporando nuevas funcionalidades que aporten valor al objetivo de obtener información relevante a partir de los documentos analizados.

Palabras clave: *Análisis de documentos, Etiquetado de imágenes, Plataforma de gestión de tareas, Proyecto CRUZAR, Memoria Histórica.*

[1] https://cruzar.edu.uy/

Índice general

1 Introducción

En el presente informe se presenta el trabajo realizado como proyecto de grado de la carrera de Ingeniería en Computación de la Facultad de Ingeniería, UdelaR[1]. Dicho proyecto se denomina "Plataforma de análisis documental: Rutas de la memoria histórica".

En este capítulo se introduce el contexto y motivación del proyecto (sección 1.1), además de mencionar el objetivo principal (sección 1.2) y la idea general de la solución implementada junto con sus aportes (sección 1.3). Finalmente, en la sección 1.4 se describe la organización del documento.

1.1 Contexto y motivación

La dictadura militar que tuvo lugar entre los años 1973 y 1985 en Uruguay marcó nuestro país dejando, entre otras cosas, mucha incertidumbre sobre los hechos ocurridos durante este período. Con el paso de los años se fue recuperando el acceso a algunos conjuntos de datos que hasta entonces estaban obstaculizados.

En particular, en el presente trabajo nos focalizamos en los datos obtenidos en el denominado *Archivo Berrutti*. Dichos datos constan de imágenes escaneadas de documentos de dicha época, los cuales en muchos casos no se encuentran en óptimas condiciones de lectura como para que un programa pueda analizarlos automáticamente.

Al ser un gran volumen de datos y ser estos de gran relevancia para la sociedad, se desarrollaron varios proyectos (uno de ellos es el presente proyecto de grado) dentro del marco del proyecto CRUZAR. Dicho proyecto tiene como objetivo el ordenamiento y la clasificación del material además de la elaboración de un programa informático que permita el cruzamiento de la información contenida en esos archivos. El cruzamiento facilita la investigación y el

[1] https://www.fing.edu.uy/

análisis de la información para poder obtener la mayor cantidad de datos relevantes. Internamente, al área de ingeniería del proyecto CRUZAR se le denomina Memoria Histórica (MH). Haremos referencia a la misma a lo largo del presente informe.

El análisis manual de los documentos consta principalmente de agregar etiquetas sobre ellos, lo cual en la actualidad consta de completar una serie de datos como el origen, tipo de documento o si forma parte de un documento conformado por muchas páginas. Cada una de estas categorías cuenta con un número definido de opciones en donde el usuario podrá seleccionar a través de un checkbox la que considere más adecuada. Los usuarios no pueden agregar etiquetas que no pertenezcan a dichas categorías.

Los encargados de realizar este análisis son en su mayoría usuarios de áreas humanísticas, en particular, docentes y estudiantes de la Facultad de Información y Comunicación, UdelaR[2]. Dichas personas no cuentan necesariamente con conocimientos tecnológicos, por lo que varios de ellos presentan dificultad a la hora de realizar su tarea con la herramienta que utilizan actualmente. Por otra parte, la misma no es un gestor de tareas, por lo que la asignación de documentos a cada alumno recae en el docente, el cual lo debe hacer manualmente sin ningún tipo de software que lo ayude. A su vez no es posible que el docente visualice el estado de avance del etiquetado de los documentos asignados, ni que pueda corregir el trabajo de sus alumnos en un mismo lugar.

El presente proyecto de grado surge a partir de la necesidad de contar con una herramienta amigable para todo tipo de usuarios y que además centralice todo el proceso de creación y ejecución de tareas de etiquetado de documentos en una plataforma web de gestión de tareas.

Al tener conocimiento sobre las necesidades planteadas tomamos conciencia que una mejora en el procesamiento de estos documentos implicaría contribuir con el descubrimiento de hechos que marcaron fuertemente la historia de Uruguay y de la vida de tantas personas. Esto fue un gran motivante a la hora de elegir el presente proyecto como culminación de nuestras carreras.

1.2 Objetivo principal

El objetivo principal de este proyecto es construir una plataforma de gestión de tareas que facilite a usuarios del área de humanidades como periodistas, historiadores, estudiantes, entre otros, el análisis de documentos conformados por imágenes escaneadas.

Para lograr cumplir con el objetivo principal, la plataforma debe contar con las siguientes características:

- Manejo de perfiles de usuario que permitan gestionar el acceso de los diferentes actores a los recursos de la plataforma (secciones de la página, documentos y tareas, entre otros)

[2] https://fic.edu.uy/

- Mecanismos de autenticación y autorización para distintas clases de actores sobre los diferentes tipos de recursos.
- Mecanismos de registro de la actividad de cada actor sobre cada tarea que permitan, por ejemplo, conocer el grado de avance en la ejecución de una tarea.
- Mecanismos para crear nuevas tareas a realizar por un actor sobre determinados recursos (documentos u otras tareas).
- Mecanismos de persistencia necesarios para el funcionamiento de la plataforma.
- Una herramienta de etiquetado flexible y completa con la cual se pueda registrar toda la información necesaria sobre los documentos.

1.3 Solución y aportes del proyecto

La solución implementada consta de una plataforma denominada "*Rutas de la Memoria Histórica*" disponible de forma online la cual se encuentra desplegada dentro del servidor de Memoria Histórica. En ella los docentes pueden crear y asignar tareas a sus alumnos para el etiquetado de documentos, mientras que los alumnos pueden llevarlas a cabo allí mismo ya que se cuenta con una herramienta de etiquetado integrada. Luego, es posible crear tareas para revisar y/o validar el trabajo realizado por los estudiantes. En estos casos posiblemente las personas asignadas a dichas tareas sean docentes o tengan conocimientos más específicos en el área.

La plataforma cuenta con el sistema de autenticación que comparten todos los proyectos contenidos dentro de MH. Cada usuario tiene acceso a las diferentes secciones y funcionalidades de la plataforma dependiendo de su/s rol/es y permisos. Además se permite la creación de nuevos roles y grupos de usuarios.

Por otro lado, se puede observar el estado y toda la información pertinente a cada tarea creada. Las etiquetas realizadas sobre cada documento dentro de cada tarea son persistidas dentro de una base de datos propia de la plataforma junto con todos los datos implicados en ella.

Un dato no menor es que la plataforma se encuentra lista para su utilización.

La plataforma desarrollada aporta un gran valor dentro del marco del proyecto CRUZAR ya que con ella se logran varios avances en comparación a la forma en la que se trabajaba previamente, en concreto se logra:

- Centralizar el flujo del etiquetado de documentos.
- Mejorar la accesibilidad a la herramienta al ser una pagina web.
- Implementar el manejo de roles y permisos.

- Fortalecer la seguridad al contar con mecanismos de autenticación y permisos.
- Crear y llevar a cavo diferentes tipos de tareas sobre los documentos.
- Crear una herramienta de etiquetado flexible y extensible.
- Registrar y poder acceder fácilmente a las etiquetas creadas sobre los documentos.

1.4 Organización del documento

El resto del documento se encuentra organizado en capítulos de la siguiente forma:

En el capítulo 2: "Análisis del problema", se describe en detalle el contexto y la definición del problema a resolver. Se presenta el análisis de los requerimientos funcionales y no funcionales, el análisis de herramientas existentes para la gestión de tareas y finalmente los conceptos clave de la solución.

En el capítulo 3: "Diseño", se presenta el diseño de la base de datos junto con el prototipo inicial de la interfaz gráfica de la plataforma.

En el capítulo 4: "Solución desarrollada", en primer lugar se realiza una descripción de todos los componentes desarrollados y externos que se encuentran implicados en la solución. Además se detallan todas las funcionalidades que posee la plataforma junto con los permisos asociados a cada una de ellas, los roles existentes, y las reglas de negocio definidas. Por otra parte, se mencionan las tecnologías utilizadas para la implementación de cada componente de la solución junto con información relevante sobre el mecanismo de autenticación, el registro de datos, la herramienta de etiquetado y los tests realizados. Por último se describen los entornos de desarrollo y cómo se llevó a cabo el despliegue de la plataforma.

En el capítulo 5: "Caso de estudio", se muestra a través de una historia con imágenes y textos descriptivos, los flujos principales de la plataforma: crear una tarea de etiquetado y ejecutarla.

Finalmente, en el capítulo 6: "Conclusiones", se presenta la evaluación de la solución desarrollada y las mejoras que podrían considerarse en un trabajo a futuro.

2 Análisis del problema

En este capítulo se introduce el contexto que impulsa la realización del presente proyecto de grado y se plantea el análisis de la problemática.

En la sección 2.1 se explica el contexto del proyecto. Se menciona el motivo por el cual surge y el proyecto mayor del cual es parte. Además se mencionan las herramientas que se usan actualmente, sus características, falencias y las necesidades de los usuarios y de las personas involucradas en el tema que motivaron la realización de este proyecto. A continuación, en la sección 2.2 se define concretamente el problema a tratar en este proyecto y en la sección 2.3 se muestra el análisis de los requerimientos funcionales y no funcionales a trabajar. En la sección 2.4 se presenta un análisis sobre la posibilidad de utilizar herramientas para el soporte de gestión de tareas. Finalmente, en la sección 2.5 se definen los conceptos que constituyen la solución y se presenta el modelo de dominio.

2.1 Contexto

En Uruguay entre los años 1973 y 1985 gobernó una dictadura cívico-militar [1]. Durante esos años ocurrieron violaciones a los derechos humanos como torturas, violaciones, secuestros y desapariciones forzadas de cientos de uruguayos. Estos actos marcaron fuertemente la historia del país dejando, entre otras cosas, una gran incertidumbre entre las personas.

El acceso a las distintas fuentes de información que aún existen de ese período fue obstaculizada durante mucho tiempo. Sin embargo, a lo largo de los años, se ha logrado el acceso a algunos conjuntos de datos. Uno (el utilizado en este proyecto), es el llamado *Archivo Berrutti*, que contiene aproximadamente 3 millones de páginas de material diverso producido por las agencias de seguridad durante la dictadura y años posteriores (entre los años 1965 y 1999). Los documentos de esta colección consisten en imágenes escaneadas de rollos de microfilm de los documentos en papel originales, que ya no se encuentran disponibles. Den-

tro de cada rollo los documentos no se encuentran organizados de ninguna forma particular excepto por un orden cronológico aproximado relacionado con cada uno. Los documentos originales incluyen texto escrito a mano, páginas escritas a máquina, imágenes y partes de material impreso (como periódicos).

El contenido de estos documentos es de gran valor para comprender mejor este período de la historia uruguaya, ayudar en la búsqueda de personas desaparecidas, aportar elementos a los juicios en curso y construir conciencia sobre este período histórico.

Desde hace aproximadamente cuatro años, un grupo de docentes de la Universidad de la República con la colaboración de actores de la sociedad civil, se encuentran trabajando en el desarrollo de herramientas informáticas para colaborar en el procesamiento de los archivos de la última dictadura militar en Uruguay en el contexto del proyecto CRUZAR.

La tarea principal del proyecto CRUZAR es la realización de investigaciones diversas sobre el contenido de estos documentos. En concreto, sistematizar y organizar los documentos históricos antes mencionados para maximizar la cantidad y calidad del conocimiento extraído. Es de esperar que dicha información permita comprender los mecanismos y el funcionamiento del sistema represivo, identificar los procesos que llevaron a la desaparición de los presos (que podrían, a su vez, permitir encontrar sus restos y brindar consuelo a sus familiares), y ayudar en el proceso de hacer justicia.

Algunas características fundamentales de estas investigaciones son:

- Son realizadas, en su mayoría, por usuarios de las áreas humanísticas (historiadores, periodistas, abogados, etc).
- Se realizan en equipos de investigación de personas con diferentes roles dentro del equipo: unos organizan el trabajo como secuencias de actividades a nivel general, otros asignan y revisan los resultados de esas actividades, otros las realizan, etc.
- Estas asignaciones de tareas podrían funcionar entre grupos, es decir, alguna persona de algún grupo podría dejar tareas para otro grupo determinado.
- Las actividades que se realizan sobre los documentos pueden ser muy diversas. Al día de hoy, una de las actividades fundamentales es la clasificación y lectura de documentos. Eso no quita que puedan aparecer otras actividades como la realización de transcripciones, la revisión de transcripciones automáticas, tareas automáticas como el procesamiento con *Optical Character Recognition* (OCR) para el reconocimiento automático de texto a partir de una imagen o la construcción de transcripciones en función de datos de sistemas externos.

En particular, una de las herramientas utilizadas dentro de este contexto en la Facultad de Información y Comunicación, UdelaR es *LabelMe* [2]. Dicha herramienta permite visualizar imágenes escaneadas de documentos de la última dictadura militar en Uruguay y realizar etiquetas sobre los mismos. Estas etiquetas se pueden clasificar en cuatro grupos:

- **Flags**: Booleanos que tienen la información que se necesita codificada en el nombre. Reflejan un tipo de documento. Ejemplo: “Acta”,“Informe”, “Memo de antecedentes”.
- **Origins**: Se corresponden con los nombres de las oficinas que generaban los documentos. Ejemplo: “Armada Nacional”,“Fuerza Aérea”.
- **Multipage**: Sirve para identificar si el documento que se está analizando es una página de inicio, de fin o está en el medio de un documento. Ejemplo: “inicio”,“continua”, “fin”.
- **Labels**: Son el general elementos gráficos como una firma o un sello. Se marcan como polígonos dentro del documento.

En la Figura 2.1 se puede ver un screenshot del LabelMe donde se observan algunas de los tipos de etiquetas mencionados previamente.

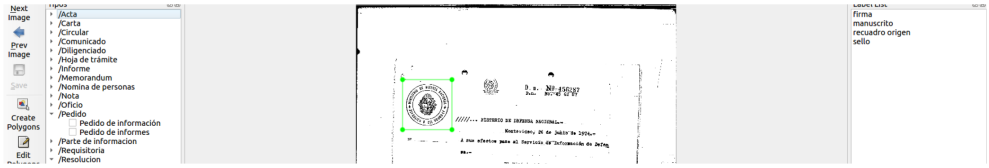

Figura 2.1: LabelMe

El contexto donde se utiliza concretamente esta herramienta es el siguiente. Dentro de la FIC, se dicta un curso que consiste en el etiquetado de estos documentos a cambio de créditos para su carrera. Allí los docentes seleccionan de a medio rollo o un rollo completo de documentos y se los copian en una carpeta determinada dentro de la sesión de usuario de los alumnos que correspondan. A partir de allí los alumnos utilizan LabelMe para crear las etiquetas sobre los documentos. Estas se guardan en un directorio al cual tienen acceso docentes de la Facultad de Ingeniería, UdelaR, que participan del proyecto de Memoria Histórica. Ellos toman esta información, la guardan y la utilizan como base para otros procesos, como por ejemplo, para la clasificación de imágenes por atributos o para entrenar software que realice las clasificaciones automáticamente.

Existen varios problemas con dicha herramienta y este flujo de trabajo, los cuales se detallan en la siguiente sección.

2.2 Definición del Problema

En la actualidad, LabelMe es utilizado y cumple su función pero existen varios problemas y limitantes. En primer lugar, no es tan intuitiva para usuarios que no se encuentran relacionados a áreas tecnológicas y es limitada en cuanto a las anotaciones que se pueden realizar. Las etiquetas que un usuario puede poner son únicamente las predeterminadas por lo que no se

pueden agregar personalizadas si fuera necesario. Por otro lado, LabelMe es una aplicación de escritorio que solo se encuentra instalada en algunas computadoras de la FIC, haciendo que su uso este limitado solo a cierta cantidad de personas al mismo tiempo y en determinado horario.

Además de estas limitaciones, el grupo de investigación tiene la necesidad de contar con una herramienta integradora que centralice todo el proceso de creación, asignación y ejecución de tareas, así como también una herramienta de etiquetado dinámica y flexible a las particularidades de cada documento.

A raíz de lo impráctico que resulta la asignación de los documentos a los estudiantes, el etiquetado de los mismos y la obtención de las etiquetas generadas, el problema a resolver se puede resumir en los siguientes puntos:

- Crear una plataforma web que centralice el creado de tareas de etiquetado de documentos de la última dictadura militar uruguaya.
- Brindar una herramienta de etiquetado flexible y completa con la cual se pueda registrar toda la información necesaria.
- Registrar quién realizo cada etiqueta sobre qué documento, cuándo la hizo, si fue modificada, eliminada, etc.
- Contar con la seguridad necesaria ya que se están manejando documentos con información sensible.

Por lo tanto, en pocas palabras se podría decir que el objetivo principal de la solución es construir un espacio centralizado en donde los docentes puedan asignarle los documentos a etiquetar a sus alumnos fácilmente y que ellos puedan etiquetarlos allí mismo. Luego, los docentes podrán revisar el trabajo realizado por los estudiantes y brindar devoluciones si lo desean.

2.3 Análisis de requerimientos

En esta sección se analiza la idea de desarrollar una plataforma web integradora en la cual se puedan crear tareas, revisarlas, validarlas y, como parte de las mismas, etiquetar documentos en formato de imagen. En este caso en particular, cabe destacar que dichos documentos son sensibles y conllevan gran valor histórico para nuestro país y la región.

En la subsección 2.3.1 se muestran los requerimientos funcionales de la plataforma mientras que en la 2.3.2 se encuentran los requerimientos no funcionales.

A los requerimientos contenidos dentro de cada grupo se los referenciará como:
[Tipo de requerimiento][Grupo del requerimiento][Número de requerimiento].
Por ejemplo, al primer requerimiento funcional listado dentro del grupo “Manejo de Tareas”

se lo referenciará con el código RFMT1, mientras que el código del segundo requerimiento no funcional dentro del listado del grupo “Seguridad” es RNFS2.

A lo largo del presente documento se utilizarán estas referencias para hacer alusión a los requerimientos definidos.

2.3.1 Requerimientos funcionales (RF)

Dentro de los requerimientos funcionales distinguimos dos grandes grupos:

- Manejo de tareas (RFMT)
- Manejo de usuarios (RFMU)

Manejo de Tareas (RFMT)

Dentro de este primer grupo de requerimientos se encuentran los siguientes:

1. Se deben poder crear tareas de tres tipos diferentes: Etiquetar, Revisar y Validar
2. Una tarea puede tener uno de los siguientes estados: Nueva, Comenzada o Finalizada.
3. Al crear una tarea de tipo Etiquetar se debe poder ver una sección en donde se listen los documentos y donde se puedan realizar búsquedas por nombre para seleccionar los que se desean incluir.
4. Al crear una tarea de tipo Revisar se debe poder ver una sección en donde se listen las tareas con estado Finalizada.
5. Al crear una tarea de tipo Validar se debe poder ver una sección en donde se listen las tareas con estado Finalizada.
6. Se le debe poder agregar una descripción a la tarea.
7. Se le debe poder determinar una fecha de entrega a la tarea.
8. Una tarea debe poder ser asignada a un usuario habilitado.
9. Una tarea de tipo Etiquetar puede ser asignada tanto a un usuario individual como a un grupo (previamente creado).
10. Una vez que un usuario finalizó una tarea no puedo volver a acceder a la misma.
11. Se debe implementar un mecanismo de anotación similar a LabelMe.
12. Cuando se ejecuta una tarea de tipo Etiquetar, se accede a un etiquetador el cual permite crear y guardar anotaciones sobre el documento que se está viendo.

13. Cuando se ejecuta una tarea de tipo Revisar o Validar, se accede al etiquetador y se pueden ver la etiquetas creadas previamente sobre los documentos. Allí se pueden crear nuevas etiquetas, modificar existentes y/o dejar comentarios sobre las mismas.

Manejo de Usuarios (RFMU)

Dentro de este grupo de requerimientos se encuentran los siguientes:

1. Manejar dos tipos de agentes en la plataforma, usuarios (personas) y sistemas automatizados.
2. Un usuario *autorizado*[1] debe poder iniciar sesión utilizando su usuario de MH y constraseña, y mantenerse logueado durante todo el tiempo que se encuentre en la plataforma.
3. Un usuario debe poder cerrar sesión en la plataforma.
4. Un usuario puede tener uno o más roles: administrador, asignador o etiquetador.
5. Cada rol debe tener asignado un conjunto de permisos los cuales determinan a qué actividades o sitios dentro de la plataforma pueden acceder.
6. Un usuario administrador debe poder ver y determinar cuáles son los usuarios habilitados para acceder y utilizar la plataforma.
7. Un usuario administrador debe poder crear roles, determinando sus permisos correspondientes. A su vez debe poder ver, editar y eliminar cualquiera de ellos.
8. Un usuario administrador debe poder asignar roles a cualquiera de los usuarios. Por defecto todos los usuarios habilitados tendrán el rol etiquetador.
9. Un usuario administrador debe poder asignar y/o eliminar permisos individuales a cualquiera de los usuarios.
10. Un usuario administrador puede ver, editar o eliminar cualquier grupo o tarea del sistema.
11. Un usuario asignador debe poder crear grupos, los cuales son subconjuntos de usuarios habilitados de la plataforma. A su vez debe poder ver, editar y eliminar los creados por él mismo.
12. Un usuario asignador debe poder ver un listado de los usuarios habilitados y así poder crear los grupos y/o asignar tareas.
13. Un usuario asignador debe poder crear y asignar tareas.
14. Un usuario asignador debe poder ver una lista de las tareas que creó en donde podrá

[1] Debe tener permiso de acceso a Memoria Histórica

visualizar más detalles de la misma como:

- tipo de la tarea
- a quién fue asignada
- fecha de entrega
- documentos o tareas asociados a dicha tarea

15. Un usuario asignador debe poder editar una tarea creada por él, siempre y cuando esta no esté comenzada o finalizada.
16. Un usuario asignador debe poder eliminar una tarea creada por él.
17. Un usuario etiquetador debe poder ver y acceder a sus tareas asignadas, visualizar los documentos asociados a las mismas y ejecutar la tarea.

2.3.2 Requerimientos no funcionales (RNF)

- **Seguridad (RNFS)**
  1. La página web debe utilizar el protocolo HTTPS.
  2. Seguridad dentro de la plataforma para el acceso a las distintas secciones de la misma y al manejo de los documentos.
  3. La plataforma debe poder funcionar desplegada en un servidor detrás de un proxy reverso. [3]
- **Interfaz (RNFI)**
  1. Un usuario debe poder ingresar a la plataforma desde la web.
  2. Disponible para navegadores Chrome y Firefox.
  3. La plataforma debe ser responsive con tamaño mínimo tablet.
  4. Interfaz amigable para todo tipo de usuarios.
- **Autenticación (RNFA)**
  1. Autenticación de usuarios mediante protocolo LDAP.
  2. Integración con el sistema de autenticación del servidor de Memoria Histórica.
- **Persistencia (RNFP)**
  1. Se deben persistir los datos en una base de datos.

2. Integración con la base de datos relacional de documentos ya existente.
3. Se debe registrar toda modificación que haga un usuario sobre los documentos. Se debe mantener un control de versiones.

- **Extensibilidad (RNFE)**
  1. Extensibilidad de la plataforma en cuanto a tipos de tareas y herramientas de etiquetado.

2.4 Análisis de herramientas de gestión de tareas

A partir de la definición del problema surgió la idea de utilizar un software de gestión de procesos, como BPMS, para manejar el flujo de creación y ejecución de las tareas dentro de la plataforma. A continuación se presenta el análisis realizado para determinar si la inclusión de alguna herramienta de este tipo resulta conveniente.

2.4.1 Software de gestión por procesos (BPMS)

La Gestión de Procesos de Negocio o Business Process Management (BPM) es la disciplina orientada a mejorar el desempeño de las organizaciones mediante la optimización y la automatización de los procesos de negocio que amparan su actividad [4]. Dichos procesos, los cuales representan una serie discreta de actividades o pasos, se deben diseñar, modelar, organizar, documentar y optimizar de forma continua.

Para llevar a cabo dicha gestión es necesario contar con un conjunto de herramientas, llamado Business Process Management Software (BPMS), cuyo objetivo es brindar el soporte necesario para cumplir con el ciclo de vida de BPM. Normalmente siguen una notación común, denominada Business Process Modeling Notation (BPMN)[2].

En primera instancia parecía que la utilización de un BPMS ayudaría a lograr simplificar la lógica de la creación y realización de tareas. Sin embargo, luego de tener los requerimientos definidos y analizados, se pudo establecer que el flujo de nuestro caso de uso era muy sencillo para una herramienta de tal magnitud ya que solo cuenta con un único proceso. El uso de BPM está orientado a procesos complejos, variables y con múltiples actividades, actores y dependencias entre ellos. En la medida que existieran más procesos y con una mayor complejidad, se podría pensar en migrar a un motor de procesos.

[2] https://www.bpmn.org/

2.4.2 Librerías para la gestión de la realización de las tareas (D-Flow y Rule-Engine)

La librería D-Flow[3] es un motor de flujo de datos que proporciona las herramientas necesarias para definir procesos dentro de la aplicación mediante una serie finita de pasos y se encarga de mantener un estado válido para ellos. Esta herramienta es muy útil cuando se deben manejar varios procesos con muchas condiciones entre las subtareas ya que garantiza la validez del estado de los procesos en todo momento. Sin embargo, es una librería muy poco utilizada y los creadores dejaron de brindar soporte desde mediados del año 2020.

Por otro lado, Rule-Engine[4] es un motor para procesar reglas de negocio. Esta librería, le permite al usuario definir reglas a través de un nombre, una condición y una consecuencia. Si la condición se cumple, se aplica la consecuencia.

Con el objetivo de ver como se podrían adaptar las herramientas D-Flow y Rule-Engine al proyecto, se realizó una prueba de concepto para cada una de las librerías. Esta prueba parte de una misma aplicación base representando la funcionalidad principal de la plataforma: la realización de tareas de etiquetado sobre documentos. Se implementaron unas pocas pantallas en donde el usuario puede comenzar una tarea de etiquetado de 2 documentos, etiquetarlos, guardar y finalmente finalizar la tarea. Por último, a partir de esta aplicación base se implementó una versión utilizando D-Flow y otra diferente utilizando Rule-Engine.

Al finalizar la prueba de concepto se concluyó que no se iba a utilizar ninguna de las dos librerías debido a las siguientes razones:

- Ambas librerías son muy poco utilizadas actualmente y tienen poco o nulo soporte.
- El fuerte de ambas librerías es la organización del estado de la aplicación en el caso de haber muchas reglas y condiciones, lo cual no es el caso. Las reglas de negocio involucradas en el proceso de realización de una tarea son escasas, y los estados en los que se puede encontrar una tarea también son pocos (Nueva, Comenzada y Finalizada) y probablemente no cambien en el futuro.
- Agregan mucha complejidad al código para una lógica relativamente simple.

2.4.3 Conclusión

Luego de analizar las herramientas presentadas anteriormente, se decidió no utilizar ninguna de ellas en la implementación de la solución ya que el flujo de creación y ejecución de tareas no es tan complejo como para justificar su uso. En consecuencia, se resolvió desarrollar el flujo involucrado en la gestión de tareas mediante una implementación propia, sin utilizar herramientas externas.

[3] https://www.npmjs.com/package/@d-flow/engine
[4] https://www.npmjs.com/package/@wizeapps/rule-engine

A grandes rasgos, dicha implementación consta de tener definidos estados para cada tarea (Nueva, Comenzada y Finalizada) y para cada documento dentro de la misma, tener un registro del estado en el que se encuentran actualmente y manejar reglas de negocio que certifiquen que el flujo de dicho proceso sea correcto en cada paso.

2.5 Modelado de la solución

En esta sección se definen los conceptos clave, los actores involucrados en la solución y las relaciones entre ellos. Además se presenta el modelo de dominio obtenido a partir del análisis.

Como consecuencia del relevamiento de los requerimientos y el análisis del problema, se definen los siguientes conceptos.

Concepto	Descripción	Atributos
Agente	Clase abstracta. Un agente puede ser un usuario o un sistema automatizado el cual puede crear tareas, realizarlas, etc. (req. RFMU1)	▪ id: String ▪ roles: Set(Rol) ▪ individualPermissions: Set(Permiso)
Usuario	Subclase de Agente. Un usuario es un humano, como por ejemplo un estudiante o un docente, que actúa sobre la plataforma.	▪ universityUsername: String ▪ universityEmail: String ▪ name: String ▪ surname: String ▪ groups: Set(Grupo) ▪ enabled: Boolean
Sistema	Subclase de Agente. Un sistema es un agente programable que puede realizar muchas de las funcionalidades de un usuario pero de forma automatizada.	▪ description: String

Grupo	Agrupación de usuarios.	▪ id: String ▪ name: String ▪ members: Set(Usuario) ▪ createdById: String ▪ isDeleted: String ▪ deletedAt: Date ▪ deletedById: String
Rol	Agrupación de permisos con un nombre definido.	▪ id: String ▪ name: String ▪ permissions: Set(Permiso) ▪ isDeleted: String ▪ deletedAt: Date ▪ deletedById: String
Permiso	Clase abstracta. Un permiso es una regla que define si el agente o rol que lo posee puede realizar cierta acción en la plataforma o no (definido por el atributo allowAccess).	▪ id: String ▪ name: String ▪ description: String ▪ allowAccess: Boolean ▪ slug: String
Permiso de app	Subclase de Permiso. Un permiso de app controla el acceso a los diferentes recursos de la plataforma. Por ejemplo: si un usuario tiene un permiso de app cuyo recurso es “grupo” y la acción es “crear”, significa que el mismo se encuentra habilitado para crear grupos.	▪ can: Set(resource: String Enum, action: String Enum)

Permiso de dominio	Subclase de Permiso. Un permiso de dominio cuyo tipo de tarea relacionado es X, significa que al que le sea asignado el permiso, podrá o no realizar tareas de ese tipo (dependiendo del valor del atributo allowAccess).	▪ taskType: TipoDeTarea
Tipo de tarea	Clasificación de una tarea determinada por su nombre. Las opciones existentes son: Etiquetar, Revisar y Validar (req. RFMT1).	▪ id: String ▪ name: String
Tarea	Clase abstracta. Una tarea es una actividad la cual tiene que ser realizada por el agente asignado a la misma.	▪ id: String ▪ type: TipoDeTarea ▪ status: Enum(Nueva, Comenzada, Finalizada) ▪ assignee: Agente ▪ records: Set(Registro) ▪ description: String ▪ deadline: Date ▪ createdBy: Agente ▪ isDeleted: String ▪ deletedAt: Date ▪ deletedById: String
Etiquetar	Subclase de Tarea. Una tarea Etiquetar consta de señalar características de los documentos asociados a la misma.	▪ documents: Set(Documento)
Revisar	Subclase de Tarea. Una tarea Revisar consta de revisar la tarea realizada por otro agente. Al revisar se pueden dejar comentarios o crear nuevas etiquetas.	▪ taskToReview: Tarea

Validar	Subclase de Tarea. Una tarea Validar consta de validar la tarea realizada por otro agente. Al validar se pueden dejar comentarios o crear nuevas etiquetas.	▪ taskToValidate: Tarea
Registro	Entidad que guarda toda la información de las acciones que un agente realiza sobre un documento como parte de una tarea. Es donde se guardan las etiquetas realizadas sobre un documento.	▪ id: String ▪ type: String Enum ▪ taskId: String ▪ documentId: String ▪ data: String
Documento	Entidad que contiene la información asociada a un documento (imagen individual).	▪ id: String ▪ name: String ▪ records: Set(Registro) ▪ url: String ▪ rollId: String
Rollo de documentos	Agrupación de documentos con un nombre definido.	▪ id: String ▪ name: String

Modelo de Dominio

A continuación se presenta el modelo de dominio creado a partir del análisis de los requerimientos funcionales. Ver Figura 2.2. Dicho modelo permite observar cómo se relacionan las clases definidas en la sección anterior.

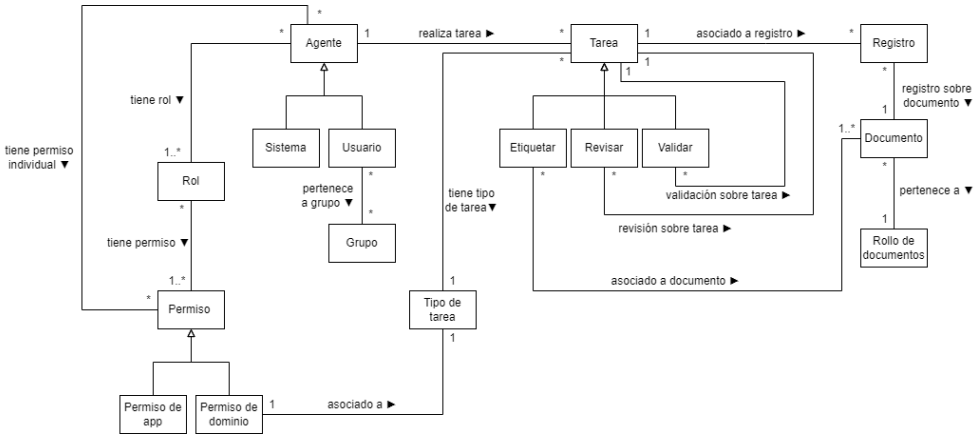

Figura 2.2: Modelo de Dominio

Restricciones no estructurales

- Si un agente “agX” tiene el rol “rolY” y “rolY” tiene el permiso “perm1” entonces el “agX” tiene el permiso “perm1”.
- Si un registro “reg1” está asociado a la tarea “tar1” y al documento “doc1”, entonces la tarea “tar1” debe incluir a “doc1” como uno de sus documentos adjuntos.
- Si una tarea es de tipo “Revisar”, la revisión debe ser sobre una tarea que tenga estado “Finalizada”.
- Si una tarea es de tipo “Validar”, la validación debe ser sobre una tarea que tenga estado “Finalizada”.
- El atributo “deadline” de una Tarea “tar1” debe ser mayor que el atributo “createdAt” de la misma.

3 Diseño

En este capítulo se muestra, a través de varias herramientas de modelado y diagramas, el diseño de la solución al problema definido en la sección anterior.

En la sección 3.1 se define el diseño de la base de datos a utilizar. Por otro lado, en la sección 3.2 se muestran los diseños iniciales de la interfaz gráfica de la plataforma, es decir, el diseño de las pantallas.

3.1 Diseño de base de datos

A partir del análisis del problema y los conceptos definidos en la sección anterior, se creó el siguiente Modelo Entidad-Relación (MER). Ver Figura 3.1. Este modelo es una herramienta que facilita la representación de entidades en una base de datos relacional.

Todas las entidades presentes en el MER poseen además los atributos createdAt (fecha de creación) y updatedAt (fecha de la última modificación), los cuales fueron ignorados en el diagrama por una cuestión de simplicidad. Sin embargo, existe una excepción. La entidad Registro no posee el atributo updatedAt ya que, como regla general, nunca se modifican los registros creados sino que siempre se crean nuevos con el fin de conservar la trazabilidad de las acciones realizadas sobre los documentos.

Por otro lado, vale aclarar que el atributo deletedById que aparece asociado a varias entidades, hace referencia al id de un agente.

Luego de diseñar el MER, se aplicó el procedimiento definido en el libro “Fundamentos de Sistemas de Base de Datos”[5] para transformarlo al Modelo Relacional y así lograr definir concretamente la estructura de la base de datos a utilizar para la persistencia.

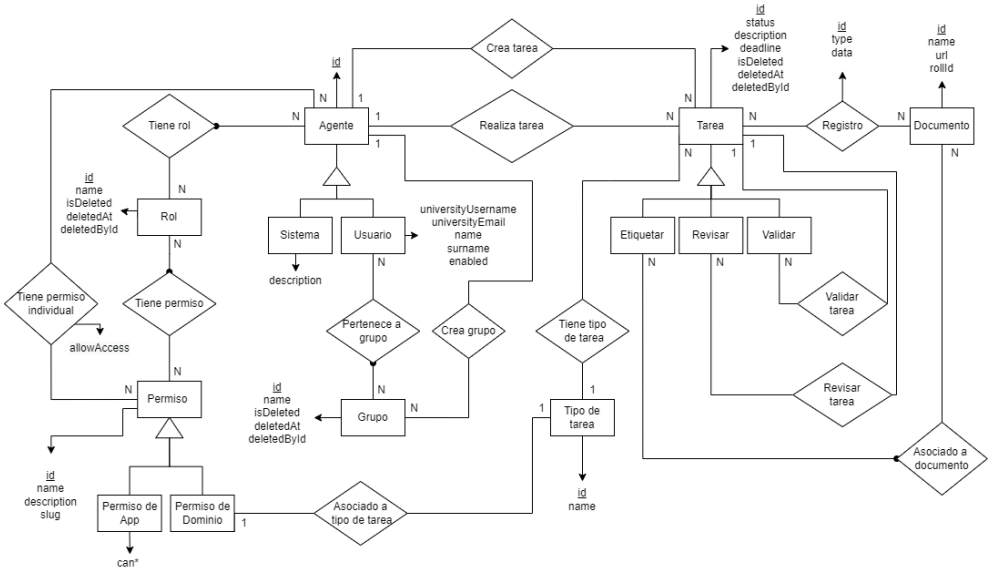

Figura 3.1: Modelo Entidad-Relación

AGENTS (id)
USERS (id, universityUsername, universityEmail, name, surname, enabled)
SYSTEM_AGENTS (id, description)

$\Pi id(USERS) \subseteq \Pi id(AGENTS)$
$\Pi id(SYSTEM\_AGENTS) \subseteq \Pi id(AGENTS)$

ROLES (id, name, isDeleted, deletedAt, deletedById)

$\Pi deletedById(ROLES) \subseteq \Pi id(AGENTS)$

PERMISSIONS (id, name, description, slug)
APP_PERMISSIONS (id)
DOMAIN_PERMISSIONS (id, taskTypeId)

$\Pi id(APP\_PERMISSIONS) \subseteq \Pi id(PERMISSIONS)$
$\Pi id(DOMAIN\_PERMISSIONS) \subseteq \Pi id(PERMISSIONS)$
$\Pi taskTypeId(DOMAIN\_PERMISSIONS) \subseteq \Pi id(TASK\_TYPES)$

APP_PERMISSIONS_CAN (permissionId, action, resource)

$\Pi permissionId(PERMISSIONS\_CAN) \subseteq \Pi id(PERMISSIONS)$

GROUPS (id, name, createdById, isDeleted, deletedAt, deletedById)

$\Pi createdById(GROUPS) \subseteq \Pi id(AGENTS)$
$\Pi deletedById(GROUPS) \subseteq \Pi id(AGENTS)$

TASKS (id, createdById, status, description, deadline, taskTypeId, assigneeId, isDeleted, deletedAt, deletedById)

$\Pi createdById(TASKS) \subseteq \Pi id(AGENTS)$
$\Pi taskTypeId(TASKS) \subseteq \Pi id(TASK\_TYPES)$
$\Pi assigneeId(TASKS) \subseteq \Pi id(AGENTS)$
$\Pi deletedById(TASKS) \subseteq \Pi id(AGENTS)$

TAG_TASKS (id)
REVIEW_TASKS (id, taskToReviewId)
VALIDATE_TASKS (id, taskToValidateId)

$\Pi id(TAG\_TASKS) \subseteq \Pi id(TASKS)$
$\Pi id(REVIEW\_TASKS) \subseteq \Pi id(TASKS)$
$\Pi id(VALIDATE\_TASKS) \subseteq \Pi id(TASKS)$
$\Pi taskToReviewId(REVIEW\_TASKS) \subseteq \Pi id(TASKS)$
$\Pi taskToValidateId(VALIDATE\_TASKS) \subseteq \Pi id(TASKS)$

TASK_TYPES (id, name)

DOCUMENTS (id, name, url, rollId)

AGENTS_PERMISSIONS (agentId, permissionId, allowAccess)

$\Pi agentId(AGENTS\_PERMISSIONS) \subseteq \Pi id(AGENTS)$
$\Pi permissionId(AGENTS\_PERMISSIONS) \subseteq \Pi id(PERMISSIONS)$

ROLES_PERMISSIONS (roleId, permissionId)

$\Pi roleId(ROLES\_PERMISSIONS) \subseteq \Pi id(ROLES)$
$\Pi permissionId(ROLES\_PERMISSIONS) \subseteq \Pi id(PERMISSIONS)$

AGENTS_ROLES (agentId, roleId)

$\Pi roleId(AGENTS\_ROLES) \subseteq \Pi id(ROLES)$

$\Pi agentId(AGENTS\_ROLES) \subseteq \Pi id(AGENTS)$

USERS_GROUPS (userId, groupId)

$\Pi userId(USERS\_GROUPS) \subseteq \Pi id(USERS)$
$\Pi groupId(USERS\_GROUPS) \subseteq \Pi id(GROUPS)$

RECORDS (id, taskId, documentId, type, data)

$\Pi taskId(RECORDS) \subseteq \Pi id(TASKS)$
$\Pi documentId(RECORDS) \subseteq \Pi id(DOCUMENTS)$

TAG_TASKS_DOCUMENTS (taskId, documentId)

$\Pi taskId(TAG\_TASKS\_DOCUMENTS) \subseteq \Pi id(TASKS)$
$\Pi documentId(TAG\_TASKS\_DOCUMENTS) \subseteq \Pi id(DOCUMENTS)$

3.2 Interfaz gráfica

A partir del análisis de la herramienta LabelMe y de los requerimientos definidos en el capítulo anterior, se realizó, como propuesta inicial de la solución, un prototipo de la interfaz gráfica de la plataforma buscando que la misma sea amigable para todo tipo de usuario (req. RNFI4). Para realizarlo se utilizó Balsamiq Wireframes[1]. Esta es una herramienta de estructuración de interfaz de usuario de baja fidelidad que simula la experiencia de dibujar en un bloc de notas o pizarrón pero usando una computadora. Se eligió utilizar dicha herramienta ya que se buscaba enfocar el prototipo en la estructura y el contenido, quitando el foco sobre colores y detalles que deberían surgir más adelante en el proceso.

El prototipo consta de varias pantallas en donde se pueden apreciar los distintos componentes que conforman las funcionalidades. Además se implementaron las transiciones entre pantallas, simulando la interacción con un usuario.

Este diseño de pantallas de baja fidelidad se realizó en etapas iniciales del proyecto por lo cual hay que tener en cuenta que el mismo sirvió de inspiración para el diseño final pero hay muchos detalles que fueron cambiando como en todo proceso iterativo.

A continuación se detalla el flujo principal de la plataforma junto con el diseño de interfaz inicial de las pantallas involucradas.

1. Un docente inicia sesión en la plataforma utilizando su email y contraseña.

[1] https://balsamiq.com/

2. Se le presenta una pantalla de bandeja de entrada en donde puede ver las tareas que le fueron asignadas y tiene la opción de crear una tarea nueva.

Figura 3.2: Diseño de interfaz: bandeja de entrada del docente

3. El docente hace click en “Crear Tarea” y es redirigido a una nueva pantalla en donde puede crear la tarea seleccionando todos los detalles de la misma. En este caso, crea una tarea de tipo etiquetar y selecciona los documentos a ser etiquetados por el estudiante Juan Pérez.

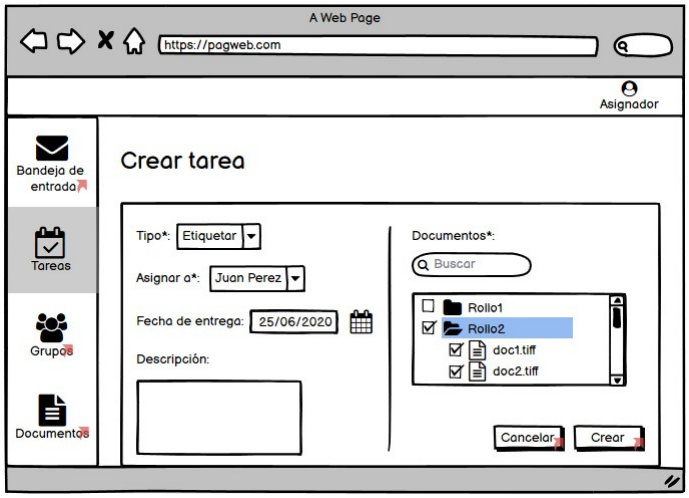

Figura 3.3: Diseño de interfaz: crear tarea

4. Por otro lado, Juan Pérez inicia sesión y es redirigido a su bandeja de entrada en donde se encuentra la tarea que le asignaron.

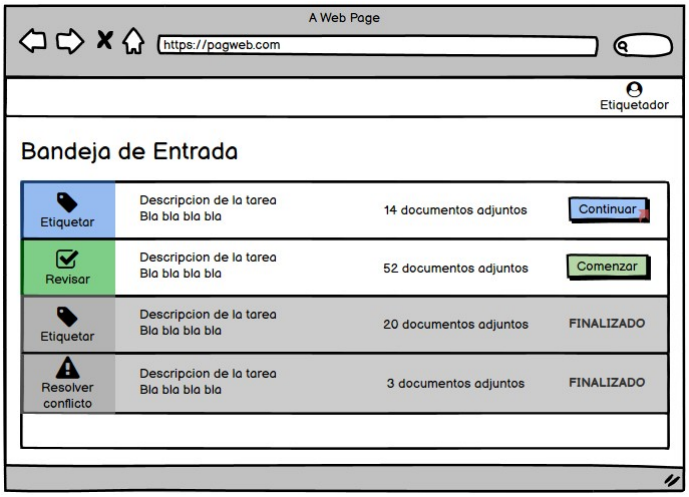

Figura 3.4: Diseño de interfaz: bandeja de entrada del estudiante

5. Hace click en “Continuar” y es redirigido al etiquetador de documentos.

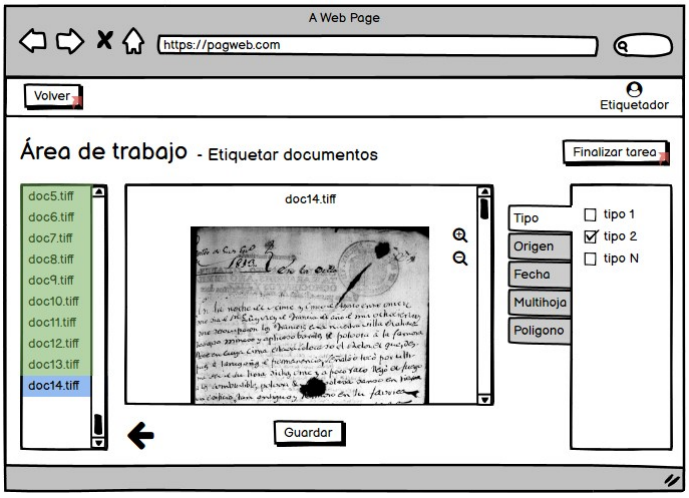

Figura 3.5: Diseño de interfaz: etiquetador

6. Etiqueta todos los documentos asociados a la tarea y hace click en “Finalizar Tarea”

Como ya se mencionó, este es el flujo principal de la plataforma pero cabe destacar que el diseño de la interfaz incluye todas las pantallas relacionadas a las funcionalidades que se desprenden de los requerimientos recabados.

En la sección siguiente se plantea la implementación de la solución final considerando todo lo que se ha analizado en las secciones de análisis y diseño.

4 Solución desarrollada

En el presente capítulo se detalla la solución al problema definido en la sección 2.2.

En la sección 4.1 se presenta la solución desarrollada a través de la descripción de sus componentes principales. A continuación, en la sección 4.2, se detallan en profundidad las funcionalidades con las que cuenta la plataforma. En la sección 4.3 se muestra cómo se implementó dicha solución y las tecnología utilizadas, y por último, en la sección 4.4 se describen todos los aspectos relacionados al despliegue de los diferentes componentes que componen la solución.

4.1 Descripción general

A grandes rasgos, la solución consta de una plataforma a la cual se la denominó "*Rutas de la Memoria Histórica*" disponible de forma online en donde los docentes le pueden asignar tareas a sus alumnos para el etiquetado de documentos y los alumnos pueden realizarlas desde la plataforma. Luego, es posible crear tareas para revisar y/o validar el trabajo realizado por los estudiantes. En estos casos posiblemente las personas asignadas a dichas tareas sean docentes o tengan conocimientos más específicos en el área.

Considerando todos los requerimientos, funcionales y no funcionales definidos previamente, se plantea una solución compuesta por seis elementos: cuatro internos (implementados como parte de la solución) y dos externos (con los cuales interactúa la plataforma, pero ya existían previamente).

Componentes internos:

1. Una base de datos en donde se persisten datos como las tareas creadas, los usuarios del sistema, las etiquetas realizadas sobre los documentos, entre otros.

2. Un backend el cual contiene lógica de negocio y expone una API para ser utilizada por los componentes frontend.
3. Un frontend de la plataforma, en donde los docentes y estudiantes pueden, a través de una interfaz gráfica, interactuar con las funcionalidades que la misma ofrece, tales como crear y realizar tareas.
4. Un frontend del etiquetador, cuya principal funcionalidad es concretamente el etiquetado de los documentos.

Componentes externos:

1. Una base de datos en donde se encuentra la información básica de los documentos: su ubicación (url), nombre, el rollo al que pertenecen, entre otros.
2. Un servidor LDAP que contiene la información básica de autenticación sobre los usuarios pertenecientes al proyecto de Memoria Histórica.

A continuación se presenta un diagrama el cual muestra cómo es la interacción entre los componentes anteriormente mencionados. Los componentes internos se encuentran dentro de la línea punteada y los externos por fuera de ella.

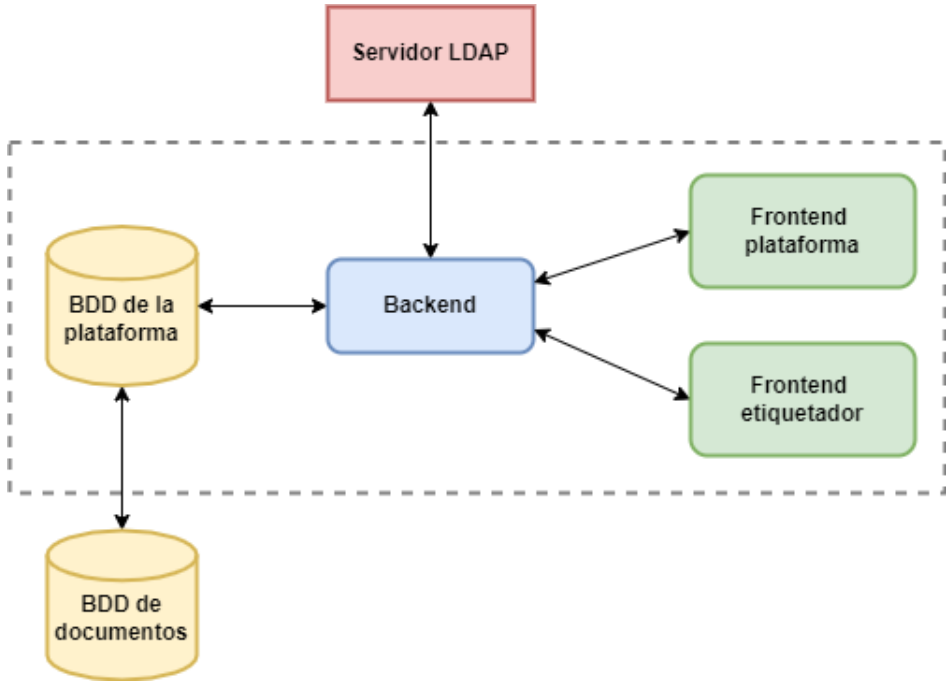

Figura 4.1: Diagrama de los componentes de la solución

En las próximas subsecciones se describe cada área de la solución detalladamente: persistencia (4.1.1), backend (4.1.2) y frontend (4.1.3).

4.1.1 Persistencia

Para la persistencia de los datos se creó una base de datos relacional (req. RNFP1), siguiendo el diseño planteado en la sección 3.1. En ella se persisten todos los datos requeridos para el cumplimiento de los requerimientos, y por ende, para el correcto funcionamiento de la plataforma.

La mayoría de las entidades del MER se implementaron como tablas individuales con los atributos definidos en la etapa de diseño. Sin embargo, para todas las clases abstractas, es decir, agentes, permisos y tareas, se utilizaron vistas en vez de tablas, con el fin de simplificar las consultas generales, es decir, las que engloban a todos los agentes, permisos o tareas. En particular, si se quiere hacer una consulta para obtener todas las tareas, se hace una consulta a la vista de tareas (llamada TASKS), la cual devuelve todos los datos asociados a las tablas de etiquetar (TAG_TASKS), revisar (REVIEW_TASKS) y validar (VALIDATE_TASKS).

Con el fin de asegurar la integridad de los datos, se generaron FOREIGN KEYS[1] en todas las tablas que tuvieran una columna que hiciera referencia a un elemento de otra tabla (por ejemplo, tablas con atributos como *taskTypeId*, *rollId*, entre otras). De esta forma, no se puede agregar ninguna entrada a una tabla que posea una referencia a un elemento que no exista. Sin embargo, como algunas tablas poseen referencias a las vistas mencionadas anteriormente (agentes, permisos o tareas), y no se pueden crear FOREIGN KEYS relacionadas a vistas, se crearon TRIGGERS[2] de manera de igualar el comportamiento de una FOREIGN KEY.

Todas las tablas poseen las columnas *createdAt* y *updatedAt* las cuales significan fecha de creación y fecha de actualización respectivamente. Se configuraron valores por defecto y TRIGGERS con el fin de actualizar estos valores automáticamente, sin tener la necesidad de hacerlo manualmente desde la plataforma cada vez que se crea o modifica un elemento.

Cabe destacar que cuando un usuario elimina un rol, grupo o tarea desde la plataforma, se realiza un *soft delete*. Esto quiere decir que el elemento a eliminar no es realmente destruido en la base de datos, sino que se marca el atributo *isDeleted* como verdadero, haciendo que este no sea visible para el usuario. Además, al ser el registro de toda actividad de los usuarios dentro de la plataforma un punto fundamental en este proyecto, se decidió agregar la fecha en la que se elimina cada elemento, junto con el identificador del usuario que realizó dicha acción.

Se eligió una base de datos relacional para persistir los datos de la plataforma por varias razones. En primera instancia, se conoce el esquema general y la estructura de todos los datos a ser persistidos. Por otro lado, este tipo de base de datos se caracterizan por su robustez e in-

[1] https://www.postgresql.org/docs/8.3/tutorial-fk.html
[2] https://www.postgresql.org/docs/13/plpgsql-trigger.html

tegridad, ya que garantizan que no se produzca la duplicidad de registros, y que, por ejemplo, al eliminarse un registro, se eliminen también todos sus registros dependientes. Por último, permite de manera directa y sencilla, la interacción con otras bases de datos relacionales.

Este último punto es fundamental ya que la integración con la base de datos de documentos ya existente, la cual es una base relacional, forma parte de los requerimientos no funcionales (req. RNFP2). Dicha base contiene la información básica de los documentos, la cual necesita ser utilizada por la plataforma para que los docentes puedan adjuntar documentos a las tareas y para que los alumnos puedan verlos al realizarla.

Para llevar a cabo dicha integración se decidió crear una FOREIGN TABLE [3] en la base de datos de la plataforma, de manera que esta se conecte y obtenga los datos de la vista (en la base de datos externa) en donde se encuentra la información de los documentos. Con el fin de obtener los datos deseados, se realiza un mapeo entre las columnas de la vista perteneciente a la base de datos externa y las columnas de la tabla perteneciente a la base de datos interna.

Una ventaja de utilizar este método es que se pueden elegir las columnas a ser consultadas por la tabla, descartando las que no son necesarias para la plataforma. Otra ventaja es que se evita la duplicación de la información. En consecuencia de que la FOREIGN TABLE actúa como un espejo de la tabla donde se encuentra guardada realmente la información de los documentos, si en ella ocurre algún cambio, este se verá reflejado automáticamente en nuestra base de datos. Por último, una virtud muy importante de utilizar este método es que si en el futuro cambia la ubicación en donde se encuentra la información de los documentos, solo hay que realizar cambios de configuración en la base de datos, sin necesidad de modificar el código.

A continuación se presenta un diagrama que muestra de manera simplificada todo lo antes mencionado sobre cómo funciona la interacción entre la base de datos de la plataforma y la base de datos en donde se encuentra la información de los documentos.

[3] https://www.postgresql.org/docs/10/sql-createforeigntable.html

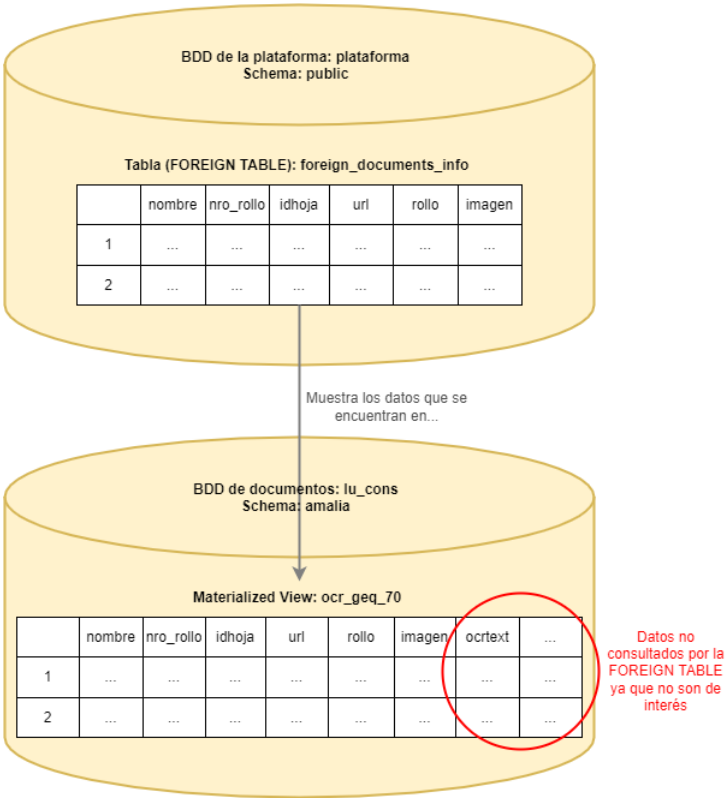

Figura 4.2: Diagrama de interacción entre bases de datos

4.1.2 Backend

El backend es el componente de la solución que contiene la mayor parte de la lógica de negocio de la plataforma. A través de una API REST [4], este componente expone varias funcionalidades, las cuales pueden ser ejecutadas por los clientes frontend. En el caso de que alguna funcionalidad implique leer, agregar, modificar o eliminar algo en la base de datos, el backend se comunica con la base de datos de la plataforma para realizar la acción que sea necesaria.

4.1.2.1 Arquitectura en capas

Con el fin de organizar la lógica dentro del backend, se decidió utilizar una arquitectura en capas. De esta forma, el código resulta más claro, escalable, desacoplado y más fácil de tes-

[4] https://www.redhat.com/es/topics/api/what-is-a-rest-api

tear. Las capas son las siguientes: API (endpoints), capa de casos de uso y capa de datos; y se ubican como se muestra en el diagrama a continuación.

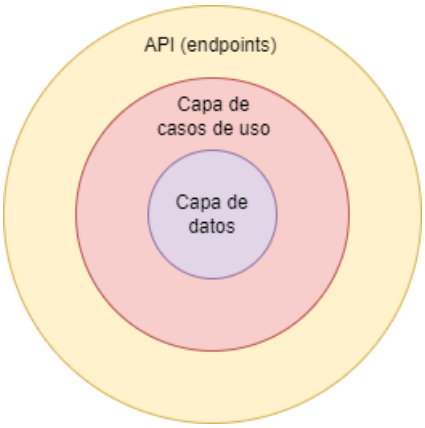

Figura 4.3: Arquitectura de capas del backend

En primer lugar, el círculo más grande representa la capa de la API. Dicha capa se comporta como una interfaz para el mundo externo, en donde se definen y exponen todos los servicios (a través de endpoints) que ofrece el backend. Además, en esta capa se define a qué caso de uso se debe llamar para llevar a cabo la funcionalidad correspondiente a cada uno de ellos y qué retornar luego de finalizado.

A modo de ejemplo, consideremos la funcionalidad de obtener todos los grupos existentes en la plataforma. Esta capa se encarga de:

1. Definir la existencia de un endpoint */groups* con el método de petición HTTP GET.
2. Establecer que al ser llamado se deberá invocar al caso de uso *getGroups* el cual devuelve todos los grupos existentes en la plataforma.
3. Definir la respuesta a ser devuelta considerando lo que fue retornado por el caso de uso. Si hubo algún error, retornar un error acorde, y de lo contrario, devolver la lista con todos los grupos.

Como se puede observar en el diagrama, esta capa se comunica únicamente con la capa de casos de uso. Esta regla aporta claridad y legibilidad al código y logra desacoplar la definición de la funcionalidad con la lógica de la misma. A simple vista, cualquier persona que lea el código puede saber cuáles son los endpoints disponibles, y, a grandes rasgos, saber qué es lo que hacen y retornan.

En segundo lugar, se encuentra la capa de los casos de uso. La misma se compone de la definición e implementación de todas las funcionalidades que ofrece el sistema. Cada caso de uso contiene chequeos de validación de los datos de entrada y toda la lógica de negocio

relacionada al mismo. Si un caso de uso necesita acceder a los datos persistidos del sistema, el mismo puede interactuar con la capa de datos para obtener los datos deseados.

Una gran ventaja de tener esta capa desacoplada del resto es que solo con mirar los nombres de los casos de uso definidos se puede saber de qué se trata el backend, es decir, qué funcionalidades ofrece y sobre qué dominios se maneja.

Por último se encuentra la capa de datos. La misma expone varias interfaces (una por cada recurso: grupos, tareas, etc) las cuales abstraen la implementación de la comunicación con la base de datos. Los métodos definidos en estas interfaces son utilizados por la capa de casos de uso. Algunos ejemplos son: *findOneById(id), findAll(), deleteOneById(id)*, entre otros.

La existencia de dichas interfaces permite desacoplar completamente la definición de los métodos con su implementación. Este hecho presenta una gran ventaja. Como se mencionó anteriormente, se eligió una base de datos relacional para la persistencia de los datos de la plataforma, pero si en un futuro esto cambia, todo seguiría funcionando tan solo cambiando la implementación de los métodos definidos en las interfaces mencionadas. La implementación de dichos métodos no posee ninguna regla de negocio sino que se limita a la implementación de consultas a la base de datos.

A continuación se presenta un diagrama que representa la interacción entre el frontend, las capas del backend y la base de datos de la plataforma.

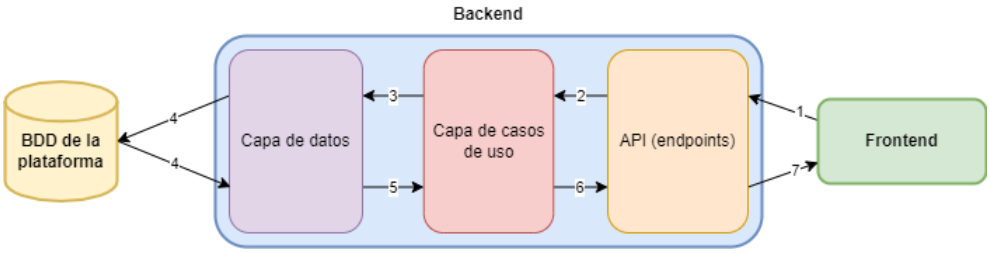

Figura 4.4: Diagrama de interacción con el backend

A continuación se ejemplifica con un caso en concreto cómo se da esta interacción:

1. El frontend hace un llamado al endpoint GET /groups
2. El backend recibe dicha petición, chequea si es un endpoint válido, y llama al caso de uso correspondiente: getGroups.
3. Dentro del caso de uso de getGroups se llama al método findAll() de la interfaz de grupos de la capa de datos.
4. La capa de datos se comunica con la base de datos de la plataforma y obtiene todos los grupos.

5. Luego, se los retorna al caso de uso.
6. El caso de uso le retorna los grupos a la capa de API.
7. Al no recibir errores, se le responde al frontend con un código HTTP de respuesta 200 (OK) junto con todos los grupos obtenidos.

4.1.2.2 Autenticación

La mayoría de los endpoints que componen la API del backend requieren que el usuario este autenticado. El único endpoint que no posee esta regla es el de login, ya que es justamente el que utiliza el usuario para autenticarse. En consecuencia de la existencia de este chequeo, no se permite que cualquier persona pueda acceder a las funcionalidades y datos de la plataforma, lo cual es fundamental teniendo en cuenta los requisitos de seguridad.

Como se observa en el diagrama de componentes de la solución (ver figura 4.1) el backend se comunica con un servidor LDAP el cual contiene la información básica de los usuarios pertenecientes al proyecto de Memoria Histórica. El backend utiliza este servidor para poder verificar que un usuario que se quiere autenticar pertenece efectivamente al proyecto y que tiene los permisos correspondientes para acceder a la plataforma (req. RNFA1 y RNFA2).

En la sección 4.3.4 que se encuentra más adelante, se detalla la implementación de todos los aspectos relacionados a la autenticación dentro de la plataforma.

4.1.2.3 Permisos

Los endpoints no solo se encuentran protegidos ante usuarios que no se encuentran autenticados, sino que además, una vez que se verificó que la autenticación es válida, se realiza un chequeo de los permisos del usuario sobre la app. Como se mencionó en el capítulo del análisis del problema, la clase “Permiso de app” es una subclase de permiso la cual controla el acceso a las funcionalidades y los diferentes recursos de la plataforma. Un usuario puede tener varios permisos de app asociados de forma individual, o a través de los roles a los cuales pertenece. Estos permisos son los que se chequean al llamar a un endpoint. Si el usuario que ejecuta una acción no posee el permiso que le permite realizar determinada acción sobre el recurso asociado al endpoint, se devolverá un error.

Los recursos existentes son:

- Task
- CreatedTask
- CompletedTask
- ReceivedTask

- Document
- DocumentRoll
- User
- Group
- Role
- Permission
- Record

Las acciones existentes se definen como: "*operación*:*alcance*" donde operación se corresponde con una acción CRUD (Create, Read, Update, Delete) y alcance hace referencia a si el recurso fue creado por el usuario (*own*) o si es cualquier recurso del sistema (*any*). Por lo tanto las acciones son:

- create:own
- create:any
- read:own
- read:any
- update:own
- update:any
- delete:own
- delete:any

Cada permiso de app posee un conjunto de pares recurso-acción dentro de su atributo *can*. Por ejemplo, el permiso de app "Ver usuarios" contiene los siguientes elementos: [recurso: User, acción: read:any] y [recurso: Permission, acción: read:any]. Esto significa que para que un usuario pueda efectivamente ver el listado de los usuarios en la interfaz gráfica correctamente, necesita acceder a los endpoints que le permiten leer todos los usuarios y leer todos los permisos.

Todos los endpoints se encuentran asociados a una acción y un recurso determinados. Por ejemplo, el endpoint GET /users se encuentra relacionado con la acción read:any y el recurso User. Cuando un endpoint es llamado, luego de verificar que el usuario esté autenticado, se analizan los permisos de app del usuario para verificar que dentro de los mismos (específicamente en el atributo *can*), se encuentre la acción y el recurso asociados al endpoint. Si se encuentra, significa que el usuario puede efectivamente acceder al endpoint requerido. Si no, se retorna un error.

Este sistema de permisos permite asegurar que no todos los usuarios de la plataforma puedan acceder a todos los endpoints definidos en la API del backend, sino que solo puedan acceder a los necesarios para poder realizar las funcionalidades que tienen permitidas. A modo de ejemplo, de esta forma un usuario etiquetador, a pesar de estar autenticado correctamente, no puede obtener todas las tareas existentes en el sistema (lo cual es una funcionalidad permitida solo para administradores).

4.1.3 Frontend

Como se muestra en el diagrama de componentes de la solución (ver figura 4.1), se implementaron dos componentes frontend: plataforma y etiquetador. El frontend de la plataforma está compuesto principalmente por la interfaz gráfica web (req. RNFI1) en donde los usuarios pueden iniciar sesión, crear tareas, ver sus tareas pendientes, crear grupos, etc. Por otro lado, el frontend del etiquetador está compuesto únicamente por una pantalla en donde se presenta una imagen, y el usuario puede realizar anotaciones (etiquetas) sobre la misma (req. RFMT11). Cabe destacar que la aplicación es responsive (tamaño mínimo de tablet, req. RNFI3) y se encuentra disponible para navegadores como Chrome y Firefox (req. RNFI2).

Ambos frontend se comunican con el backend mediante endpoints utilizando el protocolo HTTPS (HyperText Transfer Protocol Secure, protocolo seguro de transferencia de hipertexto) (req. RNFS1).

4.1.3.1 Interacción entre el frontend de la plataforma y el etiquetador

La separación en dos componentes frontend es el resultado del cumplimiento del requerimiento no funcional sobre la extensibilidad de la plataforma en cuanto a las herramientas de etiquetado. Si en el futuro se desea cambiar la herramienta con la que se realiza el etiquetado de los documentos, se puede llevar a cabo por fuera de la lógica de la plataforma en sí.

El frontend de la plataforma y el del etiquetador no se comunican directamente entre sí, pero interactúan por medio de redirecciones y pasaje de datos por la url (mediante la utilización de parámetros). El usuario no debe autenticarse nuevamente al ser redirigido ya que se mantiene la sesión del usuario (ver sección 4.3.4 para más detalles).

A continuación se muestra cómo funciona la interacción entre los dos componentes frontend a través de un diagrama secuencial.

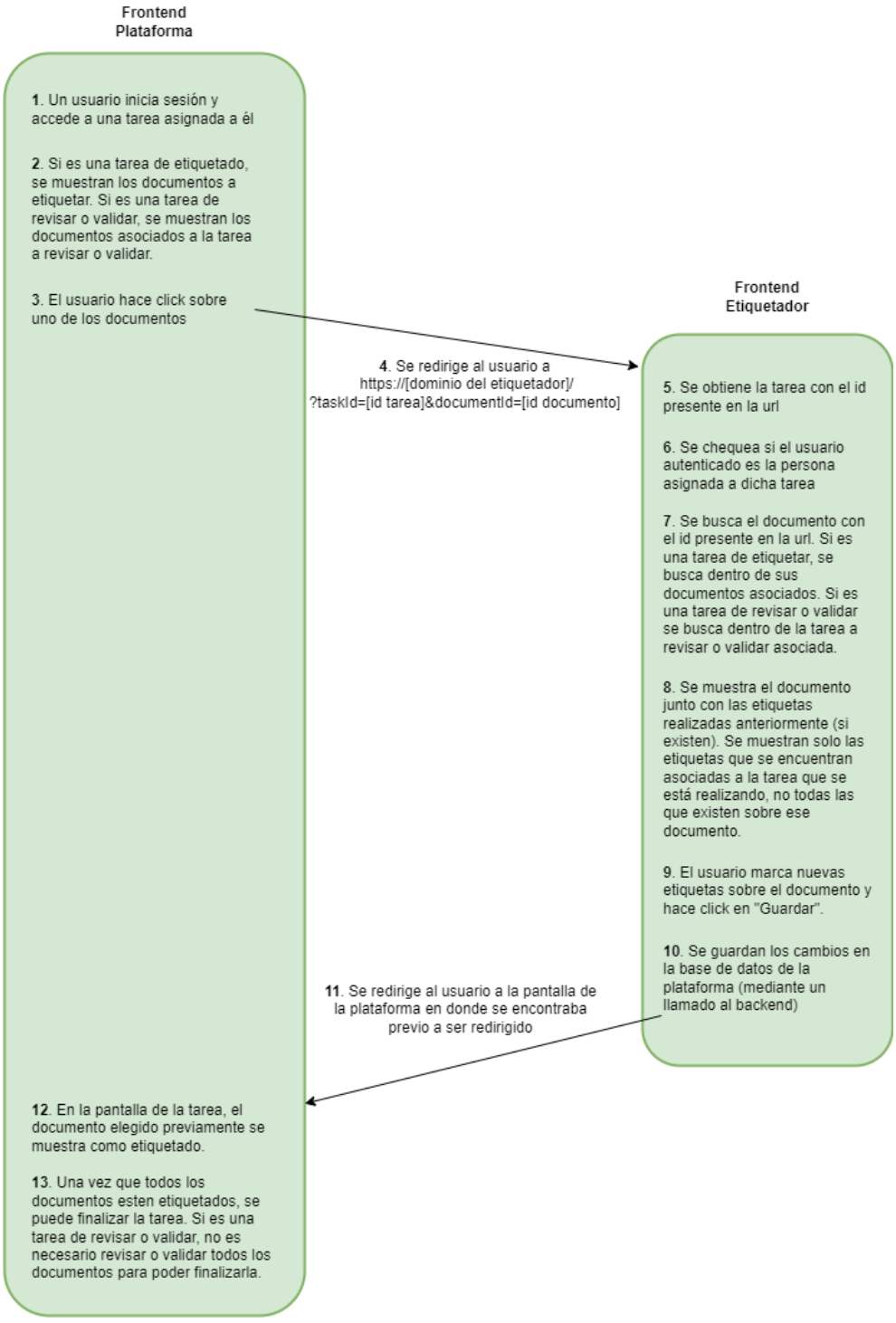

Figura 4.5: Diagrama de interacción entre los frontend

4.1.3.2 Permisos

El frontend de la plataforma, al igual que el backend, utiliza los permisos de app para limitar lo que un usuario puede ver y hacer dentro de ella (req. RNFS2). Algunos ejemplos de permisos de app son: crear tarea, editar tarea, eliminar tarea propia, crear rol, etc.

Cada pantalla tiene asociado uno o más de estos permisos de app, los cuales determinan si el usuario autenticado puede acceder o no. Incluso en algunos casos dichos permisos están asociados a botones, los cuales se muestran o no dependiendo de los permisos del usuario. Por ejemplo, si un usuario no posee el permiso de eliminar tareas propias, no aparecerá ningún botón de "Eliminar" en la pantalla en donde se listan todas las tareas creadas por el usuario.

Este comportamiento mejora notablemente la experiencia de usuario con el uso de la plataforma, ya que no le aparecerán secciones ni botones a los que no puede acceder por falta de permisos. Todas las funcionalidades que puede realizar dentro de la plataforma se encuentran visibles y accesibles, y las que no, no se muestran.

Por otro lado, también existen los permisos de dominio los cuales determinan si un usuario puede realizar cierto tipo de tarea (Etiquetar, Revisar, Validar). Si un usuario posee el permiso de dominio relacionado al tipo de tarea "Etiquetar" esto significa que puede ser asignado para ejecutar cualquier tarea de etiquetado. De caso contrario, dicho usuario no aparecerá como posible candidato a ser asignado a una tarea de este tipo. De esta manera se evita el hecho de asignarle una tarea a alguien que no tenga los permisos para poder llevarla a cabo.

4.2 Funcionalidades

En esta sección se describen todas las funcionalidades que posee la plataforma en detalle, teniendo como referencia los requisitos definidos en la sección 2.3.

En la subsección 4.2.1 se describe cada funcionalidad implementada como parte de la solución, mientras que en la subsección 4.2.2 se detallan las reglas de negocio establecidas en la plataforma.

4.2.1 Descripción

Con el fin de cumplir con los requisitos funcionales que involucran roles, en concreto, para determinar las acciones que un usuario puede realizar dentro de la plataforma según el rol que posea, se crearon tres roles en la plataforma: Administrador, Asignador y Etiquetador (req. RFMU4).

Cada uno de ellos posee un grupo de permisos de app y de dominio los cuales definen las funcionalidades a las que los usuarios con dicho rol pueden acceder.

A continuación se presentan todas las funcionalidades desarrolladas junto con sus permisos de app y/o dominio asociados, y al final se detallan los permisos que están asociados por defecto a los roles previamente mencionados.

4.2.1.1 Funcionalidades y permisos asociados

Nombre	Descripción	Permisos asociados
Iniciar sesión (RFMU2)	Lo primero que le aparece al usuario al ingresar a la plataforma es un formulario de inicio de sesión en donde deberá ingresar nombre de usuario y contraseña. Estas credenciales sos provistas por el administrador del proyecto de Memoria Histórica. Dichos datos se validan contra el servidor LDAP y si son válidos el usuario podrá acceder a la plataforma.	-
Cerrar sesión (RFMU3)	Una vez autenticado, el usuario puede cerrar sesión en cualquier momento haciendo click en un botón ubicado en el menú superior de la página.	-
Ver roles (RFMU7)	Se listan todos los roles existentes en el sistema, sin importar su creador. Se muestra el nombre de cada rol junto con sus permisos asociados y una descripción de cada uno de ellos. Como ya se mencionó, existen tres roles predefinidos: Etiquetador, Asignador y Administrador.	Ver roles (permiso de app)
Crear rol (RFMU7)	Al presionar el botón “Crear Rol” se abre un formulario con los campos *Nombre* y *Permisos*. El campo *Nombre* es requerido. Los permisos se seleccionan de un listado en el cual se ve el nombre de permiso y su descripción.	Crear rol (permiso de app)
Editar rol (RFMU7)	Al presionar el botón “Editar” se abre el mismo formulario que al crear un rol pero con los campos completados con la información existente. En este caso un usuario puede cambiar el nombre del rol o agregar/quitar cualquier permiso que desee del listado.	Editar rol (permiso de app)
Eliminar rol (RFMU7)	Se elimina el rol del sistema haciendo que si un usuario tenía ese rol, este ya no lo tenga más. Además desaparece del listado general de roles.	Eliminar cualquier rol (permiso de app)

Ver usuarios (RFMU6)	Se listan todos los agentes del sistema separados en dos pestañas: “Usuarios” y “Sistemas”. De cada usuario se puede ver su nombre, roles y permisos, además de los botones que representan las acciones que se pueden hacer sobre él (“Ver” y/o “Editar”). Al seleccionar el botón “Ver” se accede a una vista resumida de toda la información del usuario anteriormente descripta y además se puede ver el estado del mismo dentro de la plataforma, el cual puede ser “Habilitado” o “Deshabilitado”. Por otra parte se cuenta con un buscador de usuarios por apellido, el cual facilita el hecho de encontrar a una persona en concreto. Por el lado de los sistemas se lista una descripción de cada uno, y al igual que en el caso de los usuarios, sus roles, permisos y los botones que representan las acciones que se pueden hacer sobre él.	Ver usuarios (permiso de app)
Editar usuario (RFMU6, RFMU8 y RFMU9)	Al editar un usuario se accede a un formulario donde se pueden ver y/o cambiar cualquiera de estos campos: *Estado*, *Roles* y *Permisos*. Al agregar/quitar un rol a un usuario se agregan/quitan todos los permisos que dicho rol tenga asociados.	Editar usuario (permiso de app)
Ver grupos (RFMU)	Se listan todos los grupos del sistema mostrando su nombre, integrantes y los botones que representan las acciones que el usuario pueden realizar sobre cada uno. Estos pueden ser “Ver”, “Editar” y/o “Eliminar”. Al seleccionar el botón “Ver” se muestra una vista más detallada de la información del grupo.	Ver grupos (permiso de app)
Crear grupo (RFMU11)	Al presionar el botón “Crear Grupo” se abre un formulario donde se debe ingresar un nombre para el grupo y seleccionar del listado de usuarios habilitados del sistema cuáles serán los miembros del mismo. Ambos campos son obligatorios.	Crear grupo (permiso de app)
Editar grupo (creado por el usuario) (RFMU11)	Al presionar el botón “Editar” se abre el mismo formulario que al crear un grupo pero con los campos completados con la información existente. En este caso el usuario creador puede cambiar el nombre del grupo o agregar/quitar miembros al mismo.	Editar grupo (permiso de app)

Eliminar grupo (creado por el usuario) (RFMU11)	El usuario creador del grupo puede ver y seleccionar el botón de eliminar grupo (representado por un tacho de basura) y eliminar el grupo del sistema.	Eliminar grupo propio (permiso de app)
Editar cualquier grupo (RFMU10)	Mismo comportamiento que en el caso de editar grupo propio, pero en este caso puede editarlo cualquier usuario que tenga el permiso requerido, sin importar si este es su creador o no.	Editar cualquier grupo (permiso de app)
Eliminar cualquier grupo (RFMU10)	Mismo comportamiento que en el caso de eliminar grupo propio, pero en este caso puede eliminarlo cualquier usuario que tenga el permiso requerido, sin importar si este es su creador o no.	Eliminar cualquier grupo (permiso de app)
Ver tareas creadas (RFMU14)	Se listan todas las tareas del sistema creadas por el usuario autenticado. De cada tarea listada se puede ver su tipo, estado, persona asignada (si tiene), descripción (si tiene), cantidad de documentos adjuntos o tareas a revisar/validar y los botones que representan las acciones que el usuario puede realizar sobre cada tarea. Estos pueden ser “Ver”, “Editar” y/o “Eliminar”. Al seleccionar el botón “Ver” se muestra una vista más detallada de la información de la tarea donde además de los datos mencionados anteriormente se muestra también el nombre del creador de la tarea, su fecha de entrega (si tiene) y un listado con los documentos que tiene adjuntos (si la tarea es de tipo etiquetar), o la descripción de la tarea a revisar/validar. Es posible a su vez visualizar cada documento adjunto o acceder a la información de la tarea a revisar/validar.	Ver tareas creadas (permiso de app)

Crear tarea (RFMU13)	Al presionar el botón “Crear Tarea” se abre un formulario donde se deben completar varios campos, algunos de ellos requeridos y otros opcionales. Dentro de los campos requeridos se encuentra el tipo de tarea. ▪ Si se selecciona “Etiquetar” se muestra un nuevo campo (no obligatorio) el cual es para elegir si la tarea es grupal o individual. Además se muestra una descripción de lo que implica seleccionar cualquiera de las opciones. A continuación está la posibilidad de elegir la persona asignada de una lista de usuarios habilitados para realizar una tarea de ese tipo (req. RFMT8), o seleccionar el grupo asignado de un listado de grupos existentes en el sistema (req. RFMT9). También existe la opción de dejar la tarea sin asignar. El campo clave, y por este motivo, requerido en este tipo de tarea es el de *Documentos* (req. RFMT3). En este se muestra un listado paginado de los rollos de documentos existentes. Allí se puede buscar un rollo por su nombre y, si el usuario lo desea, seleccionar todos los documentos del rollo como parte de la tarea sin tener que marcar uno por uno. Además, al hacer click sobre cualquiera de ellos, se despliega a la derecha otro listado paginado con los documentos contenidos dentro de ese rollo. En esa sección también existe un buscador donde se puede buscar por nombre cualquier documento dentro del rollo y seleccionar los que se desee. A su vez se puede visualizar cualquiera de los documentos. ▪ Si se selecciona el tipo de tarea “Revisar” o “Validar”, solo existe la opción de asignación individual por lo tanto se listarán los usuarios habilitados a realizar el tipo de tarea seleccionado y se podrá seleccionar uno de ellos o dejar la tarea sin asignar. El campo clave requerido en estos casos es la tarea a revisar o la tarea a validar. En este lugar se listan todas las tareas de tipo etiquetar que tengan estado “Finalizada” (req. RFMT4 y RFMT5). De cada tarea se muestra su descripción (si es que tiene) y el usuario que la realizó. En este caso se puede seleccionar solo una del listado. A su vez si se hace click sobre el botón “Ver” se puede ver toda la información de la tarea a revisar /validar.	Crear tarea (permiso de app)

	Por otra parte existen otros campos a completar que no son obligatorios pero que suman información sobre la tarea. Uno de ellos es *Descripción* (req. RFMT6) y otro *Fecha de finalización* (req. RFMT7). En este último caso se puede seleccionar de un calendario cualquier día a partir del día de creación o dejarlo sin fecha límite.	
Editar tarea (creada por el usuario) (RFMU15)	Una tarea se puede editar tanto si se encuentra comenzada como si no, pero los campos que se pueden editar son distintos. De esto se desprende que una tarea con estado "Finalizada" no puede ser editada. ▪ Si el estado de la tarea es "Nueva", o sea no ha sido comenzada, se puede modificar cualquier campo excepto por el tipo de tarea. ▪ Por otra parte, si el estado de la tarea es "Comenzada", los únicos campos que se pueden modificar son *Descripción* y *Fecha de finalización*.	Editar tarea (permiso de app)
Eliminar tarea (creada por el usuario) (RFMU16)	Dentro del listado de tareas creadas, al hacer click sobre el botón de eliminar tarea (representado por un tacho de basura), se elimina dicha tarea del sistema. Esto implica que la tarea se borra del listado de tareas generales del sistema, del listado de tareas creadas por el usuario y de la bandeja de entrada del usuario asignado (si tenía alguno). Cabe destacar que una tarea puede ser eliminada sin importar en el estado en que se encuentre.	Eliminar tarea propia (permiso de app)
Ver todas las tareas del sistema (RFMU10)	Se listan todas las tareas del sistema. De cada una se puede ver su tipo, estado, creador, persona asignada (si tiene), descripción (si tiene), cantidad de documentos adjuntos o tareas a revisar/validar y los botones que representan las acciones que el usuario puede realizar sobre cada tarea. Estos pueden ser "Ver", "Editar" y/o "Eliminar". Al seleccionar el botón "Ver" ocurre lo mismo que se describió en la funcionalidad "Ver tareas creadas". Además de esto se cuenta con un buscador de tareas en el cual se puede buscar por nombre o apellido del creador, nombre o apellido del asignado o descripción de la tarea. A su vez existen filtros por tipo y estado de la tarea.	Ver cualquier tarea (permiso de app)

Editar cualquier tarea del sistema (RFMU10)	Mismo comportamiento que en el caso de editar tarea propia, pero en este caso puede editarla cualquier usuario que tenga el permiso requerido, sin importar si este es su creador o no.	Editar cualquier tarea (permiso de app)
Eliminar cualquier tarea del sistema (RFMU10)	Mismo comportamiento que en el caso de eliminar tarea propia, pero en este caso puede eliminarla cualquier usuario que tenga el permiso requerido, sin importar si este es su creador o no.	Eliminar cualquier tarea (permiso de app)
Ver tareas asignadas (RFMU17)	Se listan todas las tareas que tiene asignadas el usuario autenticado. De cada tarea se puede ver su tipo, creador, descripción (si tiene), fecha de entrega (si tiene), cantidad de documentos adjuntos o tarea a revisar/validar y un botón diferente dependiendo del estado de la misma. Si el estado es “Nueva” aparece un botón clickeable con la palabra “Comenzar”, y si el estado es “Comenzada” se puede ver un botón clickeable con la palabra “Continuar”. Por otra parte, si el estado de la tarea es “Finalizada”, la tarea se vuelve inaccesible para el usuario, la fila se pinta de color gris y aparece un botón deshabilitado con el nombre de su estado. Además de esto se puede elegir si ver todas las tareas o filtrar las tareas finalizadas.	Ver bandeja de entrada (permiso de app)

Realizar tarea de etiquetado asignada (RFMU17 y RFMT12)	Dentro de la sección “Bandeja de entrada”, al hacer click en “Comenzar” o “Continuar” en una tarea de tipo “Etiquetar”, se accede a una pantalla donde se muestran listados todos los documentos incluidos en la tarea. De cada uno de ellos se puede ver su nombre, estado y un botón de “Etiquetar”. Además, en la parte superior de la pantalla se muestra la descripción de la tarea, el usuario que la creó, la fecha de finalización y la cantidad de documentos adjuntos. A su vez aparece un botón para finalizar la tarea el cual se encontrará deshabilitado hasta que se haya creado al menos una etiqueta en cada documento incluido en la tarea. Al hacer click sobre el botón “Etiquetar” se accede a la vista del etiquetador. En esta se puede ver el documento, hacer zoom sobre él, rotarlo, scrollear y obviamente crear etiquetas sobre él. Las etiquetas pueden ser rectangulares (por defecto) o poligonales, dónde el usuario determina manualmente su forma. El texto de la etiqueta puede ser el que el usuario quiera. Una vez que el usuario determine que finalizó de etiquetar el documento, presiona el botón “Guardar” y regresa a la vista anterior. Si el usuario creó al menos una etiqueta, se podrá ver que el estado del documento cambió a *Etiquetado*.	Etiquetar (permiso de dominio) y Ejecutar tarea (permiso de app)

Realizar tarea de revisión asignada (RFMU17 y RFMT13)	Dentro de la sección “Bandeja de entrada”, al hacer click en “Comenzar” o “Continuar” en una tarea de tipo “Revisar”, se accede a una pantalla donde se muestran listados todos los documentos incluidos en la tarea a revisar y el botón “Ver documento”. Además, en la parte superior de la pantalla se muestra la descripción de la tarea, el usuario que la creó y la fecha de finalización. A su vez aparece un botón para finalizar la tarea el cual, en este caso, siempre se encuentra habilitado ya que no es obligatorio para el usuario que revisa la tarea crear ninguna etiqueta nueva. Al hacer click sobre el botón “Ver documento” se accede a la vista del etiquetador. En esta se puede ver el documento junto a las etiquetas creadas por el usuario que fue asignado a realizar la que, en este caso, sería la tarea a revisar. Cabe destacar que no se muestran todas las etiquetas “historicamente” creadas sobre ese documento, sino únicamente las creadas por el asignado a la tarea a revisar. Al igual que en el caso anterior, se puede hacer zoom sobre el documento, rotarlo, scrollear y crear etiquetas sobre él. Además se pueden dejar comentarios sobre las etiquetas previamente creadas o eliminarlas. Una vez que el usuario determine que finalizó de revisar el documento, (creando o no nuevas etiquetas) presiona el botón “Guardar” y regresa a la vista anterior donde puede seguir revisando documentos y luego finalizar la tarea.	Revisar (permiso de dominio) y Ejecutar tarea (permiso de app)
Realizar tarea de validación asignada (RFMU17 y RFMT13)	Análogo a la funcionalidad “Realizar tarea de revisión asignada”	Validar (permiso de dominio) y Ejecutar tarea (permiso de app)

4.2.1.2 Roles definidos y permisos asociados

Rol	Permisos asociados
Administrador	▪ Crear grupo ▪ Crear rol ▪ Crear tarea ▪ Editar cualquier grupo ▪ Editar cualquier tarea ▪ Editar grupo (propio) ▪ Editar rol ▪ Editar tarea (propia) ▪ Editar usuario ▪ Ejecutar tarea ▪ Eliminar cualquier grupo ▪ Eliminar cualquier rol ▪ Eliminar cualquier tarea ▪ Eliminar grupo propio ▪ Eliminar tarea propia ▪ Ver bandeja de entrada ▪ Ver cualquier tarea ▪ Ver grupos ▪ Ver roles ▪ Ver tareas creadas ▪ Ver usuarios ▪ Etiquetar documento ▪ Revisar documento ▪ Validar documento
Asignador	▪ Crear grupo ▪ Crear tarea ▪ Editar grupo (propio) ▪ Editar tarea (propia) ▪ Eliminar grupo propio ▪ Eliminar tarea propia ▪ Ver grupos ▪ Ver tareas creadas

Etiquetador	▪ Ejecutar tarea ▪ Ver bandeja de entrada ▪ Etiquetar documento ▪ Revisar documento ▪ Validar documento

4.2.2 Reglas de negocio

A continuación se detallan las principales reglas de negocio implementadas como parte de la solución. Estas son algunas restricciones o pautas a considerar a la hora de utilizar la plataforma.

- A la hora de realizar una tarea de tipo “Etiquetar”, se considera que un documento tiene estado “Etiquetado” cuando se ha creado al menos una etiqueta sobre él en el contexto de esta tarea.
- Para poder finalizar una tarea de tipo “Etiquetar”, todos sus documentos adjuntos deben tener estado “Etiquetado”.
- A la hora de realizar una tarea de tipo “Revisar” o “Validar”, no es necesario realizar modificaciones ni crear nuevas etiquetas en todos los documentos adjuntos para poder finalizarla.
- Solo las tareas de tipo “Etiquetar” pueden ser asignadas a un grupo. Los demás tipos de tarea sólo pueden ser asignadas a un agente en particular.
- Al crear una tarea y asignarla a un grupo, lo que ocurre es que los documentos adjuntos se dividen equitativamente entre todos los miembros del grupo, creando tareas individuales para cada uno de ellos. Estas tareas aparecerán en la bandeja de entrada de cada miembro del grupo al igual que como si hubieran sido creadas como individuales inicialmente.

  Esta herramienta es útil para los agentes “Asignadores” ya que pueden crear una sola tarea con todos los documentos que consideren y automáticamente estos se dividen entre los miembros del grupo.
- Dentro del caso anterior, si ocurre que dentro del grupo seleccionado existen personas no habilitadas a ejecutar el tipo de tarea elegido, se mostrará un mensaje informando la situación y se dividirán los documentos asignados entre las personas habilitadas del grupo.
- El campo *Fecha de entrega* de una tarea no es una limitante a la hora de finalizarla. El

usuario creador de la tarea es quien decide que hacer si este plazo no se cumple.

- Una vez que una tarea es finalizada, el usuario que la realizó no puede volver a ver los documentos ni editar sus etiquetas (req. RFMT10).
- Un grupo tiene que tener al menos un miembro.
- Un rol tiene que tener al menos un permiso asociado.
- Un usuario deshabilitado no puede ingresar a la plataforma ni ser asignado a tareas o grupos.
- Por defecto cuando un usuario accede a la plataforma por primera vez, tiene estado "Habilitado" y rol "Etiquetador".

4.3 Implementación

En esta sección se describe cómo se llevó a cabo la implementación de la solución analizando los siguientes puntos: tecnologías utilizadas (4.3.1), etiquetador (4.3.2), registros (4.3.3), autenticación (4.3.4) y testing (4.3.5).

4.3.1 Tecnologías

A continuación se describe cuáles fueron las tecnologías analizadas y finalmente las utilizadas para la implementación de cada uno de los componentes de la solución.

4.3.1.1 Persistencia

Como se mencionó en la descripción general de la solución, para persistir los datos de la plataforma se utilizó una base de datos relacional. Específicamente se decidió utilizar una base de datos PostgreSQL[5].

PostgreSQL

PostgreSQL es una base de datos relacional con más de 30 años de desarrollo activo. La misma corre en todos los grandes sistemas operativos y se ha ganado una sólida reputación por su confiabilidad y extensibilidad.

Se decidió utilizar dicha base de datos ya que, además de contar con las ventajas mencionadas anteriormente, posee todas las funcionalidades necesarias para lograr la conexión con la base

[5] https://www.postgresql.org/

de datos de documentos externa, lo cual forma parte de los requerimientos no funcionales (req. RNFP2).

4.3.1.2 Backend

Para la implementación del backend, se evaluaron varias tecnologías y lenguajes detallados a continuación.

Javascript

En primer lugar, debido a la flexibilidad y a la comunidad que hay detrás de este lenguaje, se consideró utilizar el lenguaje Javascript[6].

Javascript es un lenguaje de programación de alto nivel creado en 1995 en un principio como un lenguaje de scripting (secuencias de comandos) para páginas web, aunque hoy en día es usado en muchos entornos fuera del navegador, tal como Node.js [6].

Es un lenguaje interpretado, es decir, que la mayoría de sus implementaciones ejecutan las instrucciones directamente, sin una previa compilación del programa a instrucciones en lenguaje de máquina.

Javascript es un lenguaje dinámico, lo cual significa, por ejemplo, que es posible agregar nuevas propiedades o métodos a un objeto mientras el programa está en ejecución [7]. A su vez admite estilos de programación orientados a objetos, imperativos y funcionales.

Por otro lado, Javascript es un lenguaje no bloqueante lo que quiere decir que las tareas que se realizan no quedan bloqueadas esperando ser finalizadas bloqueando a las siguientes [8]. Por ejemplo, si hay una tarea que requiere buscar un dato en la base de datos, cuando se realiza el llamado a la base, Javascript libera al procesador para que siga ejecutando otras tareas mientras espera a recibir la respuesta. Una vez que la recibe, se retoma la tarea inicial haciendo uso del dato recibido. En otras palabras, el proceso no se queda suspendido esperando una respuesta, sino que se libera para que se puedan realizar otras tareas mientras tanto.

La popularidad del lenguaje ha ido incrementando en los últimos años, siendo hoy uno de los 5 lenguajes más utilizados para el desarrollo web.

Sin embargo, este lenguaje presenta una gran desventaja si se desea escribir un código mantenible y extensible a largo plazo: es un lenguaje de tipado débil. En otras palabras, en Javascript no es necesario especificar el tipo de dato al declarar una variable a diferencia de otros lenguajes como Java.

Dicha característica aporta flexibilidad y rapidez en el desarrollo, lo cual es deseable en otras áreas de la solución como en el frontend. No obstante, en el caso de la implementación del backend de la plataforma, la mayor prioridad es la claridad y mantenibilidad del código, por

[6] https://www.javascript.com/

lo que se terminó optando por un lenguaje tipado como lo es Typescript.

Typescript

A diferencia de Javascript, Typescript[7] es un lenguaje de programación de tipado fuerte, el cual compila a Javascript de forma legible y estandarizada. Typescript agrega sintaxis adicional de tipos con el objetivo de lograr un código autodescriptivo, más limpio, robusto y mantenible. De esta forma, las clases detalladas en la sección 2.4.1 pudieron ser definidas junto con sus atributos y utilizadas en el código aportando claridad y organización.

A su vez, el editor de código que se este utilizando puede detectar posibles errores (relacionados a los tipos de datos) en la etapa de escritura del código, lo cual agiliza el desarrollo.

En consecuencia de que Typescript compila a Javascript, este puede correr donde sea que Javascript pueda correr: en el browser, en Node.js, etc.

En conclusión, mediante la utilización de Typescript se obtienen las ventajas de Javascript sumado a las ventajas que posee un lenguaje tipado.

Node.js

Una vez definido el lenguaje a utilizar en el backend y con el fin de poder correr Javascript en el servidor web, se decidió utilizar Node.js[8].

Node.js es un entorno de ejecución para JavaScript orientado a eventos asíncronos. El mismo proporciona el entorno para ejecutar código Javascript fuera del browser, lo cual permite a los desarrolladores crear aplicaciones Javascript del lado del servidor [9]. Puede correr en varios sistemas operativos como Windows, Linux, MacOS, etc.

Node.js corre en un solo hilo y es no bloqueante (ya que ejecuta Javascript) lo cual es muy eficiente para la memoria. Por ello, es muy oportuno desarrollar sistemas escalables en él [10].

Un módulo en Node.js es un conjunto de archivos Javascript que definen un conjunto de funcionalidades [11]. El mismo puede ser reutilizado a lo largo de toda la aplicación Node.js. Cada módulo cuenta con su propio contexto y existen tres tipos de módulos: los módulos centrales, los locales y los de terceros.

Los módulos centrales contienen las funcionalidades mínimas de Node.js [12]. Dichos módulos son compilados y cargados automáticamente cuando el proceso Node.js comienza. Sin embargo, es necesario importarlos explícitamente en caso de necesitarlos en la aplicación. Algunos de estos son: *url* (incluye los métodos para la resolución y parseo de URLs), *path* (incluye los métodos para lidiar con las rutas de los archivos) y *fs* (incluye las clases, métodos y eventos para leer o escribir archivos).

Los módulos locales son módulos definidos y utilizados en la misma aplicación Node.js. Sin

[7] https://www.typescriptlang.org/
[8] https://nodejs.org/es/

embargo, estos módulos se pueden empaquetar y publicar vía NPM, para que cualquier desarrollador los pueda utilizar. NPM (Node Package Manager) es una herramienta open source que actúa como gestor de paquetes en donde las personas pueden publicar o descargarse módulos Node.js.

Los módulos de terceros son módulos implementados y publicados por otros desarrolladores. Dichos módulos se encuentran disponibles para descargar y utilizar en gestores de paquetes Node.js como por ejemplo NPM. Este sistema de modularización le permite a los desarrolladores compartir código de una forma sencilla y escalable, evitando "reinventar la rueda".

Express.js

Express.js[9] es una infraestructura de aplicaciones web de Node.js la cual proporciona mecanismos para la escritura de manejadores de peticiones HTTP con diferentes rutas y verbos (GET, POST, PUT, DELETE, etc).

Se decidió utilizar dicho framework de Node.js ya que permite definir con facilidad las rutas que componen la API del backend, junto con lo que se debe ejecutar al ser llamadas. A su vez existen paquetes de middleware compatibles para abordar muchos de los problemas más comunes dentro del desarrollo web [13]. Hay librerías para trabajar con cookies, sesiones, inicios de sesión de usuario, cabeceras de seguridad y muchas más.

Passport.js

Para todo lo relacionado a la autenticación y el manejo de la sesión de los usuarios se utilizó el middleware de autenticación Passport.js[10].

El mismo provee más de 500 estrategias para la autenticación de usuarios, entre ellas, una llamada passport-ldapauth[11] que es la que se utilizó para la autenticación con el servidor de LDAP de la Facultad de Ingeniería.

Jest

Para la implementación de los tests unitarios sobre los casos de uso del backend se utilizó la herramienta Jest [12]. La misma es un framework de testing para aplicaciones Javascript.

Inicialmente fue creada por Facebook para testear aplicaciones React y actualmente es una de las herramientas más populares para el testing unitario. Ganó sus popularidad debido a que puede ser aplicada para testear Javascript tanto en aplicaciones frontend, como backend.

Algunas de las ventajas de utilizar Jest son:

- Aislamiento. Asegura que los diferentes tests no influyan en el resultado de los demás. Los mismo son ejecutados en paralelo.

[9] https://expressjs.com/
[10] https://www.passportjs.org/
[11] https://www.passportjs.org/packages/passport-ldapauth/
[12] https://jestjs.io

- Cubrimiento. Ofrece una forma sencilla de conocer el cubrimiento de código abarcado por los tests implementados.
- Moqueo de funciones. Provee una manera simple de moquear funciones, clases, etc.
- Documentación. La documentación es muy completa y contiene múltiples ejemplos para todas sus funcionalidades.

Debido a las ventajas mencionadas y a que toda la solución podría llegar a ser testeada utilizando la misma herramienta es que se seleccionó Jest como framework de testing unitario.

4.3.1.3 Frontend

En el caso de los componentes frontend, en ambos casos (plataforma y etiquetador) se utilizó el lenguaje Javascript de la mano del framework React.js. A su vez, únicamente para el frontend de la plataforma se utilizó la librería Redux.

React.js

React[13] es una librería de Javascript creada por Facebook que se utiliza para construir interfaces de usuario.

Es una librería basada en componentes los cuales son piezas de código lógicas y autocontenidas que describen una parte de la interfaz del usuario [14]. Dichos componentes son definidos por los desarrolladores. Por ejemplo, uno podría definir que un formulario junto con un botón de "Guardar" es un componente, mientras que otro podría definir que solo el botón ya es un componente por sí solo. Estos componentes se pueden juntar para crear una interfaz de usuario completa.

React abstrae la mayor parte del trabajo de renderizado, permitiéndole al desarrollador enfocarse únicamente en el diseño de la interfaz. Precisamente, una ventaja que posee React con respecto a otras librerías de Javascript es el manejo del DOM. Gracias al manejo de una DOM virtual, React es capaz de, ante un cambio en la interfaz, actualizar solo los elementos del DOM que cambiaron, evitando volver a renderizar los elementos sin cambios.

React utiliza una extensión de la sintaxis de JavaScript llamada JSX. Dicha extensión le permite a los desarrolladores escribir un código más limpio, e integrar, de una forma más intuitiva, el código HTML con Javascript según las necesidades del proyecto [15].

En lugar de mantener la lógica y el maquetado en archivos separados, React reconoce la fuerte dependencia que existe entre ellos y los mantiene juntos en cada uno de los componentes a través de la utilización de JSX.

Redux

[13] https://es.reactjs.org/

Por otro lado, Redux[14] es una librería también de JavaScript que permite manejar el estado de una aplicación. Es una herramienta independiente pero es comúnmente utilizada junto con React.

Esta librería evoluciona las ideas de Flux[15], la cual es una arquitectura cuya principal característica es que el flujo de datos es unidireccional.

Una aplicación autocontenida por lo general contiene las siguientes partes [16]:

- El estado (donde se guardan los datos que actúan como fuente de verdad para la aplicación)
- La vista (interfaz gráfica la cual depende del estado)
- Las acciones (eventos que ocurren a causa de las acciones del usuario los cuales generalmente causan un cambio en el estado)

En una arquitectura en donde el flujo de datos es unidireccional, dichas partes interactúan de la siguiente forma:

1. El estado describe la condición de la aplicación en un momento específico.
2. La interfaz se renderiza de acuerdo a ese estado.
3. Cuando algo sucede, por ejemplo cuando un usuario hace click en un botón, el estado se actualiza acordemente.
4. La interfaz se vuelve a renderizar basándose en el nuevo estado.

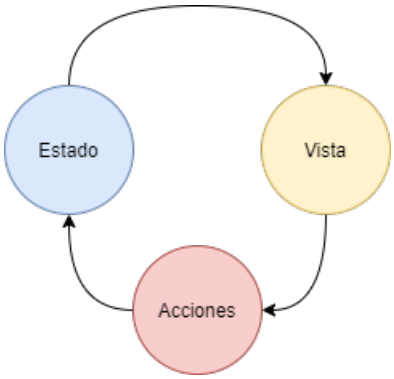

Figura 4.6: Flujo unidireccional de Redux

A su vez Redux cuenta con los siguientes principios:

- Existe un único estado global de la aplicación, es decir, hay una única fuente de verdad.

[14] https://es.redux.js.org/
[15] http://facebook.github.io/flux/

- El estado es inmutable, es decir, cada vez que se hace un cambio en el estado global de la aplicación, se debe crear un nuevo objeto el cual pasará a ser el nuevo estado.
- El estado es solo lectura. La única forma de cambiar el estado es a través de acciones. La interfaz no puede cambiarlo directamente.
- Para especificar como cambia el estado dependiendo de las acciones, se escriben funciones "reducer". Los reducers son funciones puras que reciben el estado anterior y una acción, y retornan el nuevo estado.

Se decidió utilizar Redux para la implementación del frontend de la plataforma pero no para la implementación del frontend del etiquetador ya que en este último caso no era necesario contar con un estado global de la aplicación ni con nada que justificara su uso.

En conclusión, mediante la utilización de los frameworks mencionados, debido a todas las ventajas y puntos fuertes que poseen, se logró un código organizado, flexible, extensible y performante.

4.3.2 Etiquetador

En esta subsección se menciona la librería utilizada para implementar el requerimiento funcional de crear etiquetas sobre documentos (req. RFMT12). Además se detalla la manera en la que fue implementado el etiquetador de forma de cumplir con el requerimiento no funcional de ser extensible (req. RNFE1), pudiendo así cambiarse por otra herramienta de etiquetado cuando se requiera.

Annotorious

Annotorious [16] es una librería JavaScript para anotación de imágenes en páginas web. Se pueden crear diferentes formas de etiquetas, editarlas, eliminarlas o agregar comentarios sobre ellas. Una etiqueta se compone de una selección de una parte de la imagen, la cual puede ser rectangular o poligonal, junto con una descripción. También se registran otros datos como quién la creo, comentarios, fecha de creación, entre otros que en este caso no son relevantes. El formato en el que se registra esta información es JSON.

Es una librería liviana, pesando solo 300 kB. Además se caracteriza por ser flexible, extensible e interoperable. Está basada en el estandar W3C para anotaciones web. Es completamente customizable ya que se puede cambiar su aspecto con CSS, agregar formateadores propios para aplicar estilos de anotación basados en reglas o crear nuevos plugins.

Seleccionamos dicha herramienta por todas las ventajas que posee además de que, al estar implementada en JavaScript, fue fácil integrarla a la plataforma. Annotorious, en su versión estándar, se adapta perfectamente a las necesidades del problema y permitió que se le puedan agregar funcionalidades extra como poder rotar o hacer zoom sobre las imágenes.

[16] https://recogito.github.io/annotorious/

Extensibilidad

Como ya se mencionó anteriormente en este documento, se decidió implementar el etiquetador como un frontend separado del de la plataforma con el fin de que, si en algún momento se implementa o se encuentra una herramienta de etiquetado que se adapte mejor a las necesidades del problema, sea más sencillo integrarla a la plataforma (req. RNFE1).

Si lo que se desea es únicamente cambiar la herramienta pero seguir manteniendo el frontend del etiquetador, lo que se debe hacer es en primer instancia es adaptar el código a la nueva herramienta. Esta debe cumplir con una característica principal para poder ser integrada y mantener el comportamiento actual del etiquetado de los documentos: las etiquetas creadas deben ser guardadas en formato *string* y el etiquetador debe poder obtenerlas de la base de datos en dicho formato y mostrarlas adecuadamente en la pantalla.

Si, por otra parte, lo que se desea es integrar un nuevo frontend a la plataforma, se debe cumplir la característica mencionada anteriormente y además tener algunas consideraciones.

- Cambiar la variable de entorno *REACT_APP_ANNOTATOR_URL* en los .env de los proyectos del backend y el frontend de la plataforma.
- Mantener la auteticación del usuario al pasar de la plataforma al etiquetador.
- Comunicarse con los siguientes endpoints del backend: *GET /received-tasks/:taskId* para obtener la tarea correspondiente y *POST /records* para guardar las etiquetas sobre cada documento.
- Al guardar las etiquetas, redireccionar al frontend principal de la plataforma, idealmente a la vista en la que el usuario se encontraba antes de acceder al etiquetador.

4.3.3 Registros

Cuando un usuario crea y guarda una etiqueta sobre un documento adjunto a una tarea, lo que ocurre es que se crea un elemento en la tabla “RECORDS” de la base de datos. La información que se guarda es el *id* del documento, el *id* de la tarea, su fecha de creación, las etiquetas creadas sobre él y el tipo de registro.

Como se mencionó anteriormente, la herramienta Annotorious registra la siguiente información de las etiquetas: coordenadas, forma de la etiqueta (rectangular o poligonal), creador, fecha de creación, descripción y comentarios en formato JSON. Esta información es transformada a string a la hora de ser guardada en la base de datos de la plataforma.

Por otro lado, el tipo de registro puede ser uno de los siguientes:

- *partial-tag*
- *tag*

- *partial-review*
- *review*
- *partial-validate*
- *validate*

En el momento en que el usuario guarda una o más etiquetas desde el etiquetador, se crean registros de tipo "partial-*tipoDeTarea*". Esto indica que la tarea no ha sido finalizada completamente, por lo tanto las etiquetas creadas no tienen por qué ser las definitivas. Cuando el usuario presiona el botón "Finalizar tarea" se crean registros de tipo *tag*, *review* o *validate* dependiendo el tipo de tarea para cada documento adjunto. Esto indica que las etiquetas guardadas en dichos registros son las definitivas dentro de la tarea.

Cabe destacar que los registros no pueden ser editados ni eliminados desde la plataforma como parte del requerimiento no funcional de registrar todo lo que un usuario hace sobre un documento (req. RNFP3). A partir del campo que contiene el identificador de la tarea se desprende cuál usuario creó las etiquetas, ya que la persona asignada es única y es esta la que puede acceder a realizar la tarea.

Un hecho relevante se da en las tareas de tipo "Revisar" y "Validar". Como fue mencionado anteriormente, un usuario que revisa/valida no necesariamente tiene que crear nuevas etiquetas para poder finalizar la tarea. Se podría pensar que en estos casos no se crea un registro nuevo ya que no hay etiquetas nuevas, pero no es lo que ocurre. En esos casos también se crean registros de tipo "partial-*tipoDeTarea*" con las etiquetas previamente creadas en la tarea a revisar/validar. Análogamente se crean registros de tipo *review* o *validate* cuando se finalizan las mismas.

4.3.4 Autenticación

Para contextualizar, la plataforma implementada en este proyecto es parte de un grupo de aplicaciones encargadas de realizar diferentes acciones sobre los documentos de la dictadura militar. Este conjunto de aplicaciones se encuentra desplegado en el mismo servidor bajo el nombre "Memoria Histórica"(MH). Este servidor se encuentra protegido bajo un sistema de autenticación el cual chequea que cualquier usuario que quiera acceder a él esté autorizado y tenga los permisos requeridos.

Al formar parte de Memora Histórica, la solución implementada cuenta con este método de autenticación. Esto se traduce a que, si un usuario ingresa a la dirección de la plataforma, lo primero que verá es el formulario para iniciar sesión perteneciente al servidor mencionado. El formulario cuenta con los campos *nombre de usuario* y *contraseña*. Si este logra autenticarse, el usuario es redirigido a la plataforma y se agrega un header de nombre *remote_user* a la request de redirección y a todas las demás que se hagan a continuación.

Al redirigir el usuario a la plataforma, se llama a un endpoint del backend el cual verifica la existencia de dicho header en la request. Si se encuentra, se chequea que el usuario exista en el servidor LDAP asociado. Luego, se busca al usuario dentro de la base de datos de la plataforma. Si ya se encuentra registrado, se autentica dentro de la plataforma y puede acceder a todas las funcionalidades que sus permisos le permitan. De lo contrario, se crea un nuevo Usuario en la base de datos con la información obtenida del servidor LDAP, con estado “Habilitado” y el rol “Etiquetador” por defecto. Finalmente se autentica e ingresa a la plataforma como en el caso anterior.

Con el fin de poder ejecutar la plataforma en forma local (por fuera del servidor MH) y contar con un sistema de autenticación que también valide contra el servidor LDAP, se implementó un formulario de inicio de sesión propio de la plataforma. El formulario cuenta con los campos *nombre de usuario* y *contraseña*. Dichos datos se validan contra el servidor LDAP utilizando Passport.js y finalmente, si los datos son correctos, se autentica al usuario dentro de la plataforma. Vale la pena destacar que dicho formulario no se presentará al usuario si la plataforma se encuentra corriendo en el servidor de MH. En ese caso se utiliza el flujo de autenticación mencionado en primera instancia.

Cuando un usuario es autenticado, el middleware de sesión de Express.js crea y guarda una cookie en el navegador llamada *connect.sid*. A partir de ese momento, la misma es incluida en todas las requests y responses realizadas entre servidor Node.js y el navegador. Dicho procedimiento mantiene la sesión del usuario y brinda seguridad.

En consecuencia de que la cookie se guarda en el navegador, es posible compartir la sesión entre las dos aplicaciones frontend que componen la plataforma. De esta forma, el usuario no debe autenticarse nuevamente al ser redirigido al etiquetador durante la ejecución una tarea.

En la figura 4.7 se presenta un diagrama que muestra el flujo explicado anteriormente.

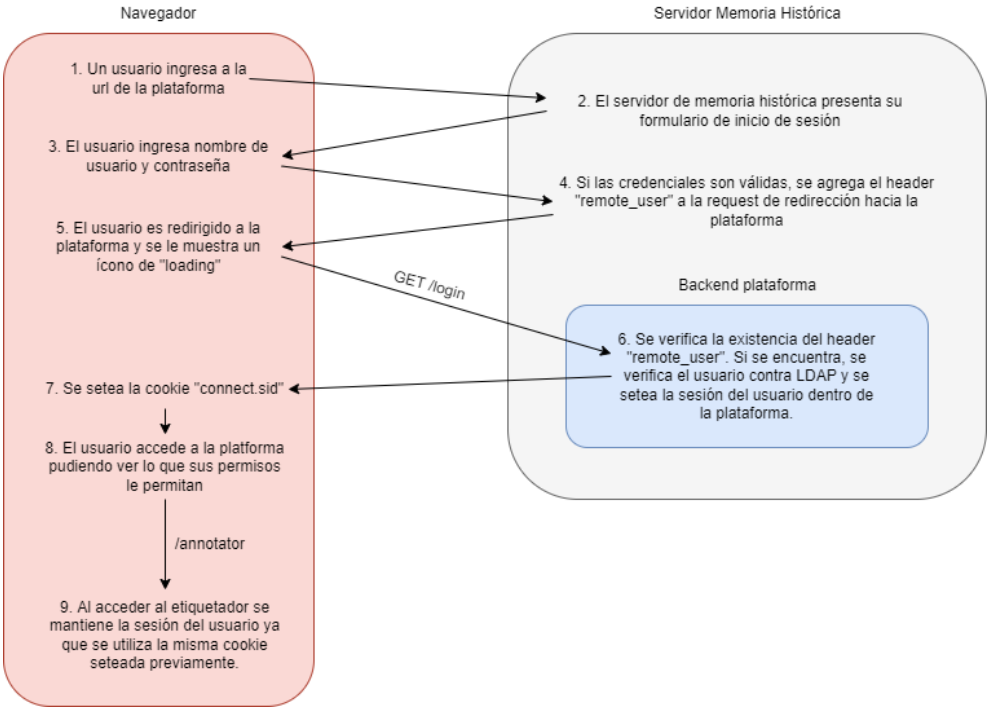

Figura 4.7: Diagrama de flujo de autenticación

4.3.5 Testing

Es de público conocimiento que el testing es una pieza fundamental a la hora de desarrollar una aplicación confiable y de alta calidad. En particular, el testing unitario aporta calidad al código ya que verifica que las partes más pequeñas de una aplicación funcionen como es esperado de forma aislada. Usualmente los tests unitarios son rápidos de ejecutar y precisos en el feedback que proveen.

En el caso del presente proyecto se decidió implementar este tipo de tests para todos los casos de uso desarrollados en el backend. Esta decisión fue tomada considerando que la mayor parte de la lógica de negocios se encuentra allí.

Se planteó como objetivo lograr el 80 % de cubrimiento de líneas en los 34 casos de uso actualmente definidos.

Durante el desarrollo de los tests se encontraron errores menores que fueron corregidos.

Finalmente, se alcanzó y superó el objetivo planteado con respecto al cubrimiento de líneas (ver figura 4.8).

File	% Stmts	% Branch	% Funcs	% Lines
All files	96.16	86.23	98.11	95.97
changeTaskStatus.ts	95.12	91.3	100	95
createGroup.ts	94.73	87.5	100	94.44
createGroupTask.ts	100	100	100	100
createRecord.ts	100	100	100	100
createRole.ts	93.33	50	100	92.85
createTask.ts	96.38	88.88	100	96.34
createUser.ts	94.11	50	100	93.75
deleteGroup.ts	100	85.71	100	100
deleteRole.ts	100	100	100	100
deleteTask.ts	100	100	100	100
getAllTasks.ts	100	100	100	100
getCompletedTasks.ts	100	100	100	100
getCreatedTasks.ts	100	80	100	100
getDocumentsFromDocumentsRoll.ts	100	83.33	100	100
getDocumentsRolls.ts	100	80	100	100
getGroups.ts	100	100	100	100
getPermissions.ts	100	50	100	100
getReceivedTasks.ts	100	100	100	100
getRoles.ts	100	100	100	100
getSingleAgentById.ts	100	100	100	100
getSingleDocument.ts	100	100	100	100
getSingleGroup.ts	100	100	100	100
getSingleRole.ts	100	100	100	100
getSingleSystemAgent.ts	100	100	100	100
getSingleTask.ts	100	100	100	100
getSingleUserByEmail.ts	100	100	100	100
getSingleUserById.ts	100	100	100	100
getSingleUserByUsername.ts	100	100	100	100
getSystemAgents.ts	100	100	100	100
getUsers.ts	100	100	100	100
helpers.ts	84.61	79.16	84.61	84.61
index.ts	100	100	100	100
updateAgent.ts	93.02	71.42	91.66	92.85
updateGroup.ts	100	92.85	100	100
updateRole.ts	100	100	100	100
updateTask.ts	95.12	87.87	100	94.93

Figura 4.8: Reporte de cubrimiento de los tests unitarios

4.4 Despliegue

En esta sección se describe el proceso de despliegue de la plataforma dentro del servidor de Memoria Histórica, junto con otros detalles sobre la ejecución de los componentes de la solución y la comunicación entre ellos.

4.4.1 Entornos de desarrollo

Con el objetivo de poder ejecutar la aplicación localmente y luego poder ejecutarla en un ambiente de producción, contamos con variables de entorno en todos los componentes implementados, es decir, en el backend, frontend de la plataforma y frontend del etiquetador. En todos ellos existen dos archivos *.env* y *.env.production* los cuales se encargan de definir estas variables de entorno (las cuales se corresponden esencialmente con rutas y puertos).

Para poder ejecutar la plataforma localmente, es necesario conectarse mediante SSH al ser-

vidor de Memoria Histórica y configurar los túneles necesarios entre los puertos para poder establecer una conexión con el servidor de LDAP y la base de datos.

Actualmente ambos entornos se comunican con la misma base de datos, la cual se encuentra dentro del servidor de Memoria Histórica. Si se desea utilizar una base de datos independiente de la de producción al desarrollar localmente, el desarrollador deberá cambiar algunas variables definidas en el archivo *.env* (nombre de la base, host, puerto, usuario y contraseña).

Por otro lado, cabe destacar que al correr la plataforma de forma local no es posible visualizar los documentos. Esto se debe a que para poder acceder a ellos es necesario que la plataforma corra dentro del servidor de Memoria Histórica. Sin embargo, esto no es un impedimento a la hora de desarrollar.

4.4.2 Despliegue dentro del servidor de Memoria Histórica

Como parte de los requerimientos no funcionales, la plataforma debía ser desplegada dentro del servidor de Memoria Histórica de la Facultad de Ingeniería el cual funciona detrás de un proxy reverso (req. RNFS3). El mismo posee el sistema operativo Linux.

Dado que la plataforma no fue pensada para recibir mucho tráfico concurrente y para aprovechar los recursos del servidor de Memoria Histórica, se decidió que los tres componentes principales de la aplicación (backend, frontend de la plataforma y frontend del etiquetador) se sirvan bajo el mismo servidor web Node + Express. Para ello, se siguieron los pasos mencionados a continuación.

En primer lugar, para poder subir y ejecutar la plataforma dentro del servidor, se nos otorgaron usuarios con los permisos necesarios para realizar dicha tarea. Luego, dentro de la carpeta correspondiente al usuario, se creó una carpeta dedicada al proyecto. Dentro de ella se subió la carpeta con el código del proyecto backend junto con los códigos minificados de los dos proyectos frontend. Finalmente, se aplicaron las configuraciones pertinentes para el correcto funcionamiento en lo que refiere al proxy reverso y al inicio/fin de la ejecución del servidor web Node + Express. Para esto último se utilizó *systemd*[17] el cual es un conjunto de bloques de construcción básicos para Linux. Ofrece un conjunto de demonios de inicialización los cuales interactúan con el kernel de Linux. Esto hace que si el servidor sufre alguna baja, los procesos asociados a esta herramienta se volverán a ejecutar automáticamente cuando el servidor se reanude tras la carga del kernel de Linux.

Como se mencionó previamente, el servidor de Memoria Histórica funciona bajo un proxy reverso, el cual se encarga de, a la hora de recibir una consulta a la url *https://www.fing.edu.uy/mh/dev/**, redireccionarla al puerto que corresponda sin que el cliente sepa a cuál. En este caso, todas las consultas dirigidas hacia *https://www.fing.edu.uy/mh/dev/rutas/** serán enviadas al puerto en donde se encuentra corriendo el servidor web de Node + Express.

[17] https://systemd.io/

El servidor web Node + Express es el que se encarga de resolver, por un lado, qué realizar al momento de recibir una consulta a la API y por otro, de retornar los archivos HTML, css y javascript de ambas aplicaciones frontend.

Concretamente:

- Si se recibe una consulta a la ruta */api/**, significa que se quiere llamar a un endpoint de la API, por lo que devuelve la respuesta correspondiente en formato JSON.
- Si se recibe una consulta a la ruta */app/**, significa que se desea obtener un archivo relacionado a una página de la plataforma, por lo que devuelve el archivo HTML, css o javascript correspondiente.
- Si se recibe una consulta a la ruta */annotator/**, significa que se desea obtener un archivo relacionado a una página del etiquetador, por lo que devuelve el archivo HTML, css o javascript correspondiente.

A continuación, en la Figura 4.9 se muestra dónde se encuentran desplegados los componentes que forman parte la solución desarrollada y cómo interactúan entre sí.

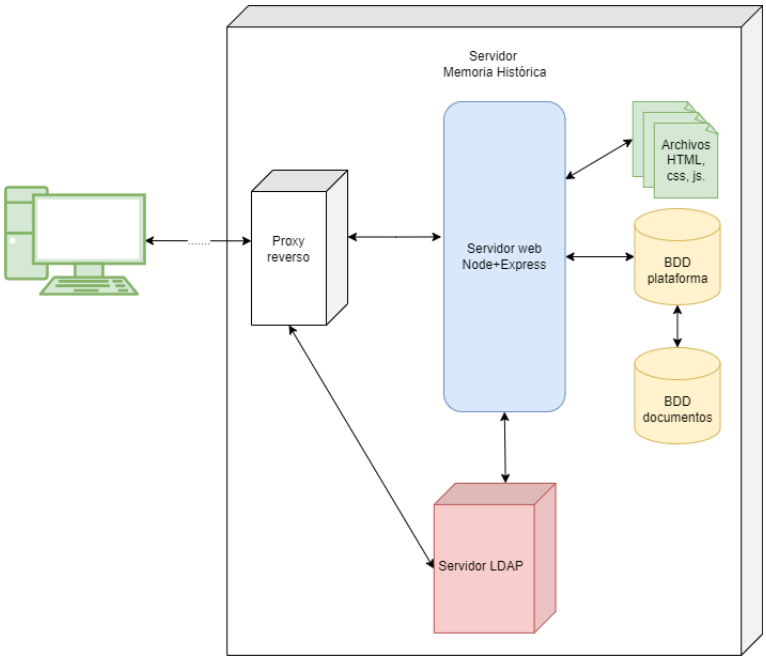

Figura 4.9: Diagrama de despliegue

5 Caso de estudio

En el presente capítulo se muestran dos de los flujos principales de la plataforma: crear una tarea (sección 5.1) y realizar una tarea (sección 5.2).

Se eligieron estos flujos para mostrar todo el proceso por el que pasa una tarea, desde su creación hasta que es finalizada.

En primer lugar se mostrará la creación de una tarea de tipo Etiquetar, esto implica el siguiente flujo de actividades:

1. Autenticarse con un usuario de rol Asignador (req. RFMU2)
2. Visualizar las tareas creadas (req. RFMU14)
3. Crear una nueva tarea (req. RFMU13)
    a) Seleccionar que la tarea sea de tipo Etiquetar (req. RFMT1)
    b) Elegir que la tarea sea asignada a un grupo (req. RFMT9)
    c) Definir una fecha de entrega para la tarea (req. RFMT7)
    d) Agregar una descripción a la tarea (req. RFMT6)
    e) Adjuntar los documentos a etiquetar (req. RFMT3)

En segundo lugar se mostrará el flujo de actividades que se llevan a cabo a la hora de realizar una tarea de tipo Etiquetar:

1. Autenticarse con un usuario de rol Etiquetador (req. RFMU2)
2. Visualizar la bandeja de entrada (req. RFMU17)
3. Comenzar la tarea (req. RFMU17)

4. Agregar etiquetas sobre los documentos asignados (req. RFMT12)
5. Reanudar y finalizar la tarea (req. RFMT2)

5.1 Crear Tarea

Este caso de estudio comienza cuando Virginia, docente de la FIC con rol “Asignador” se autentica en la plataforma ya que desea crear tareas de tipo “Etiquetar” para uno de sus grupos de estudiantes (Figura 5.1).

Figura 5.1: Iniciar sesión (docente)

Para llevar esto a cabo ingresa a la sección “Tareas creadas” que se encuentra en el menú lateral (Figura 5.2).

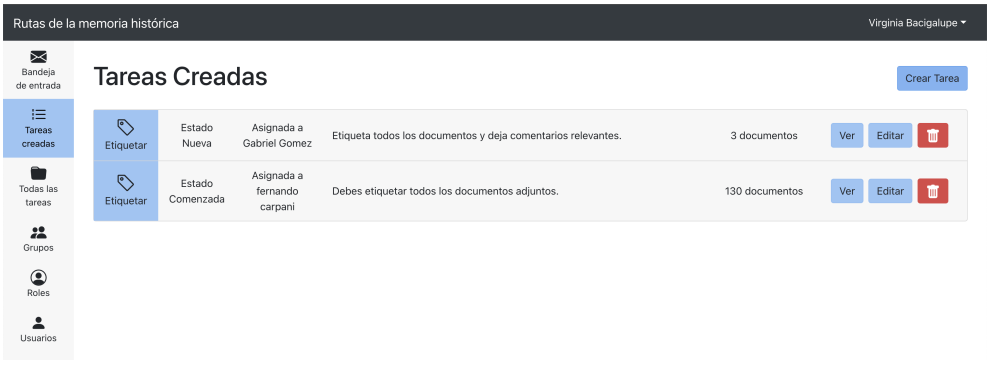

Figura 5.2: Sección “Tareas creadas” de la docente

Presiona el botón “Crear tarea” y se encuentra con el formulario que se muestra en la Figura 5.3.

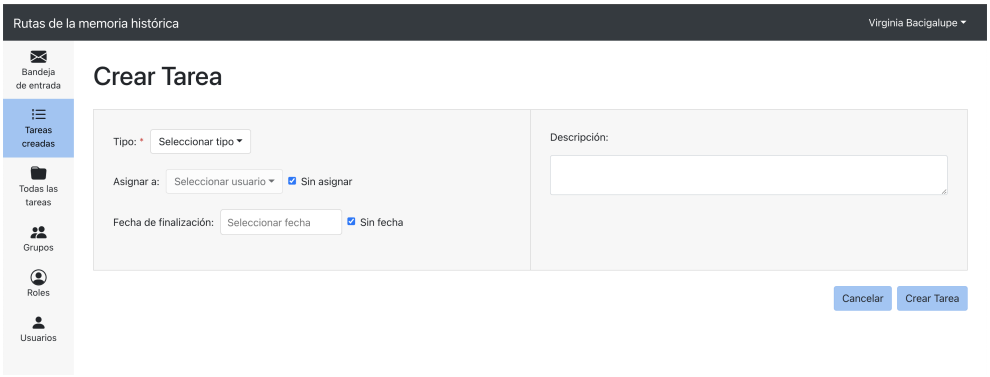

Figura 5.3: Vista de “Crear Tarea”

En primer lugar selecciona el tipo de tarea “Etiquetar” en el campo *Tipo* (Figura 5.4).

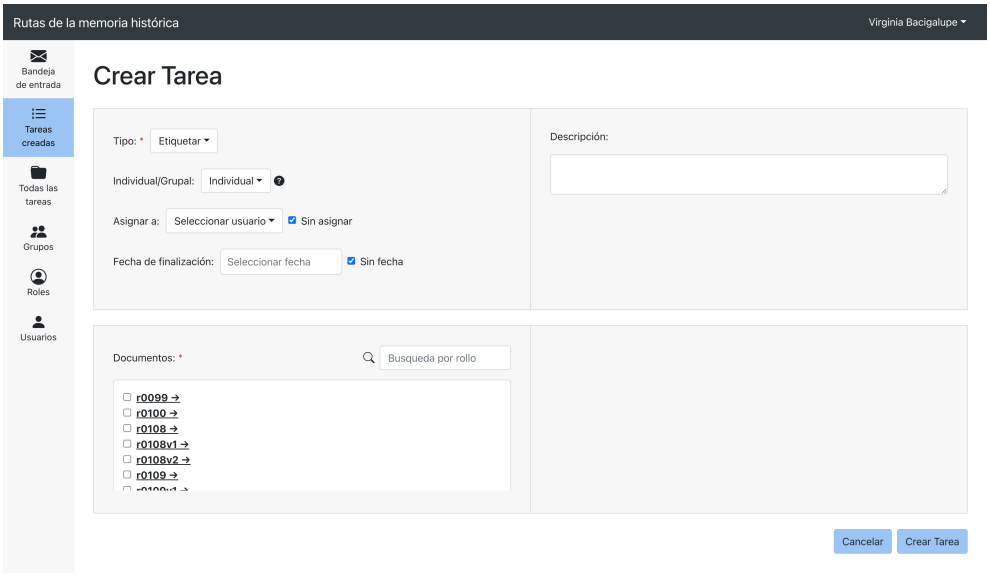

Figura 5.4: Vista de “Crear Tarea”. Se seleccionó tipo de tarea “Etiquetar”

Luego selecciona que quiere que dicha tarea sea grupal y elige de un listado cuál es el grupo al que se la quiere asignar. En este caso, selecciona el grupo denominado “Clase A”.

A continuación determina que la fecha de entrega será una semana a partir de la fecha actual y agrega una breve descripción para que sus alumnos comprendan lo que se espera que hagan (Figura 5.5).

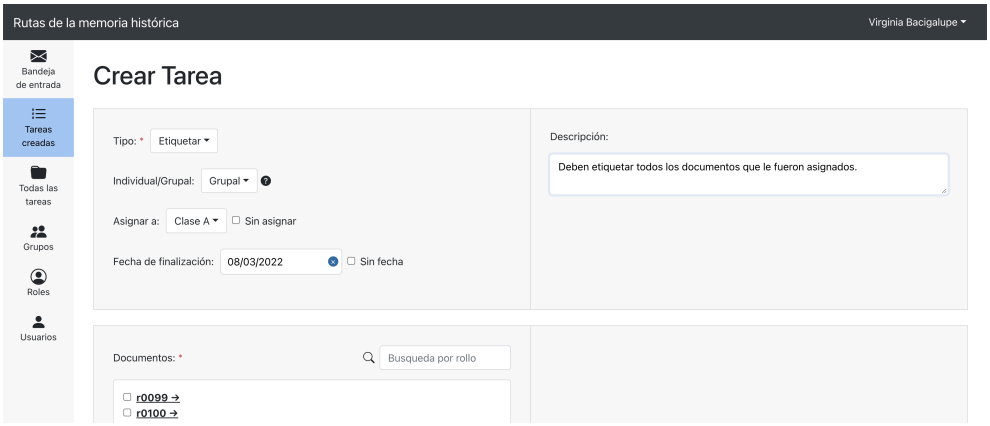

Figura 5.5: Vista de “Crear Tarea”. Se agregó una descripción y fecha de entrega

Como es una tarea de tipo etiquetar, Virginia debe proceder a seleccionar los documentos adjuntos. Para ello utiliza el buscador para encontrar el rollo *r0108* que es el que le interesa analizar. Lo selecciona completo haciendo click en el checkbox que se encuentra a la izquierda de su nombre y a la vez puede ver cualquiera de los documentos que lo integran para comprobar que sean los correctos (Figura 5.6).

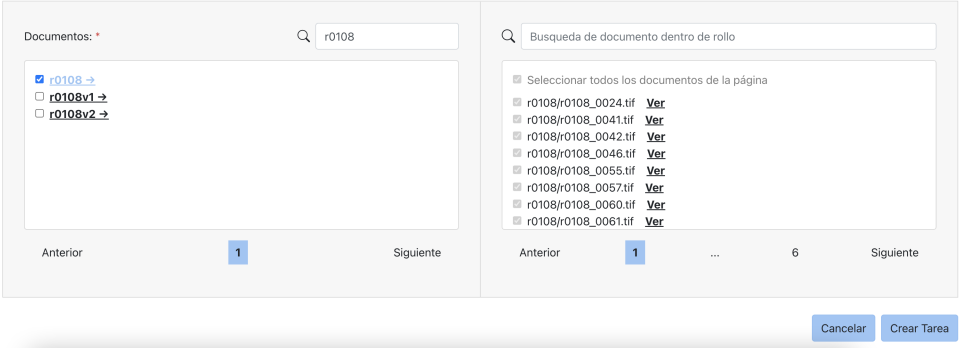

Figura 5.6: Vista de “Crear Tarea” con el rollo *r0108* seleccionado completamente

Al estar conforme con todos los datos que completó en el formulario, Virginia presiona el botón “Crear Tarea” y regresa a la vista anterior donde puede ver que se creó una tarea de tipo “Etiquetar” por cada miembro del grupo seleccionado (tres en total). Cada tarea contiene la misma descripción, fecha de entrega y estado “Nueva”. Además puede visualizar la cantidad de documentos de cada una, la cual es la cantidad de documentos seleccionados divididos equitativamente entre los miembros del grupo (Figura 5.7).

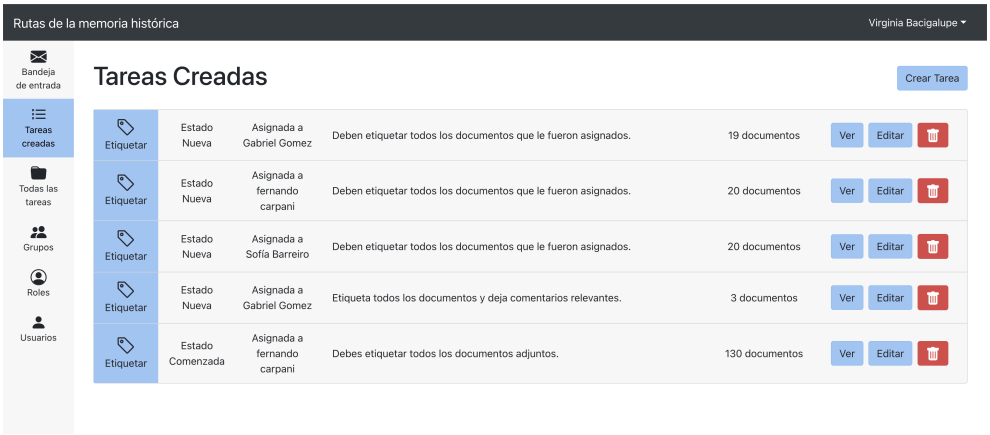

Figura 5.7: Sección “Tareas Creadas” con las nuevas tareas

Allí mismo puede presionar el botón “Ver” y comprobar si todo está bien. Si encuentra algún detalle que desea cambiar puede presionar el botón “Editar” y arreglarlo. Por otra parte si se da cuenta que no creó la tarea correctamente o hubo algún tipo de problema, puede eliminar cualquiera de ellas (Figura 5.8).

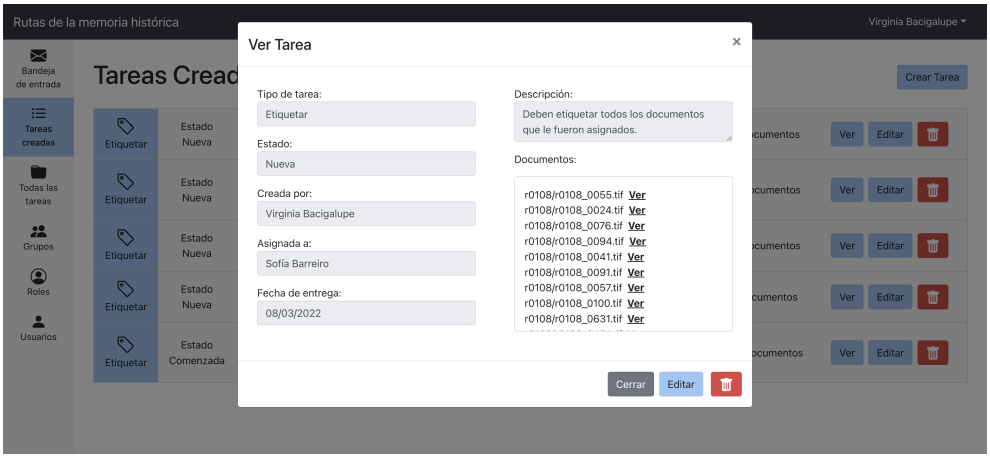

Figura 5.8: Sección “Tareas Creadas”. Seleccionado el botón “Ver” correspondiente a una tarea

5.2 Realizar Tarea de tipo Etiquetar

El otro lado de la historia ocurre cuando Sofía, estudiante de la FIC con rol “Etiquetador” se autentica en la plataforma (Figura 5.9).

Figura 5.9: Iniciar sesión (alumna)

Allí se encuentra con que en su bandeja de entrada hay una nueva tarea de tipo “Etiquetar” a la cual ha sido asignada. Desde ahí puede ver que la persona que la creó fue su docente Virginia, la descripción de la tarea, su fecha de entrega y la cantidad de documentos que tiene asignados para etiquetar (Figura 5.10).

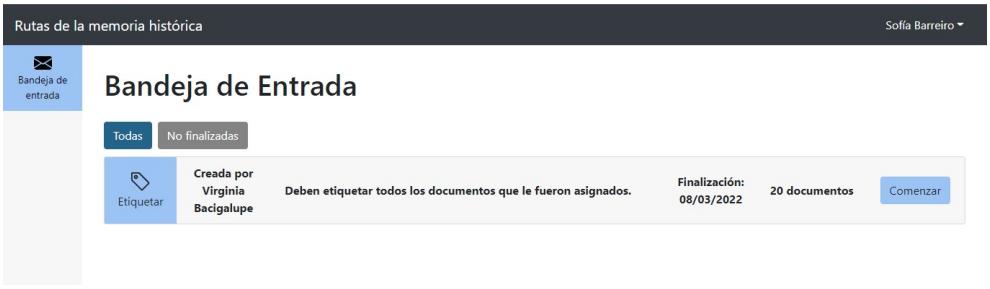

Figura 5.10: Bandeja de entrada de la alumna

En ese momento decide que quiere comenzar a realizarla por lo que presiona el botón “Comenzar”. Luego se le presenta una pantalla en donde se detallan los datos de la tarea junto con un listado de sus documentos adjuntos, el estado de cada uno de ellos (etiquetado o no etiquetado) y un botón de “Etiquetar”.

Sofía percibe que hay un botón de “Finalizar Tarea” pero este se encuentra deshabilitado. Al pasar el cursor sobre él, ve un mensaje que dice que la tarea puede ser finalizada una vez que se etiqueten todos los documentos (Figura 5.11).

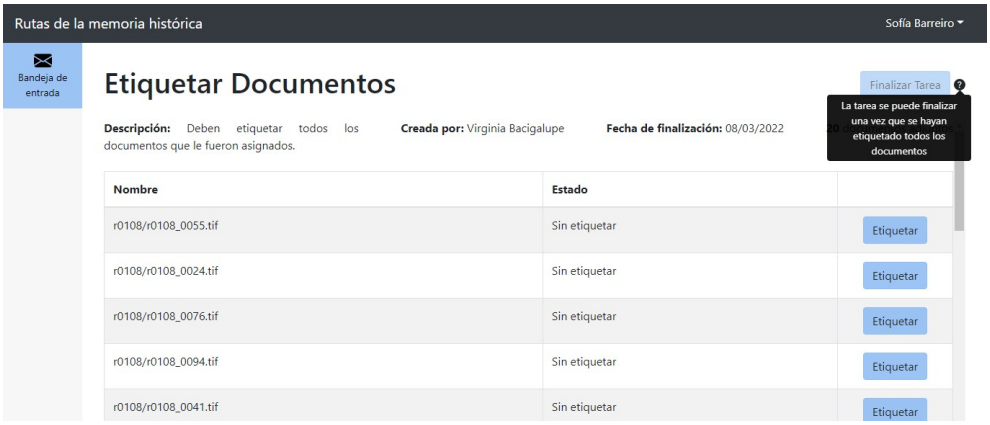

Figura 5.11: Comienzo de tarea de etiquetado

A continuación, presiona el botón “Etiquetar” y es redirigida al etiquetador (Figura 5.12). Allí se encuentra con una imágen del documento en tamaño grande, la cual puede rotar, hacer zoom y hacer scroll sobre ella. Además se encuentra con que puede crear dos tipos de etiquetas: una rectangular por defecto o una poligonal para casos como sellos o firmas que no respetan una forma regular.

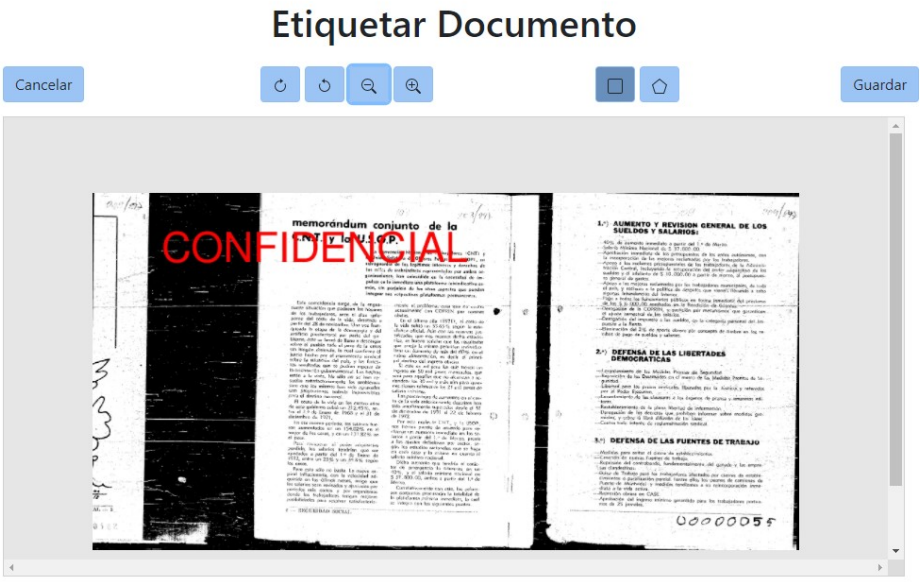

Figura 5.12: Vista del etiquetador

Sofía procede a leer en detalle todo el documento y a medida que encuentra datos relevantes

crea etiquetas con los comentarios que considera adecuados (Figura 5.13). Si se equivoca puede editarlos o eliminarlos. Al crear una etiqueta puede ver que aparece el momento en que fue creada y el nombre de quién la hizo (Figura 5.14).

Figura 5.13: Creación de una etiqueta

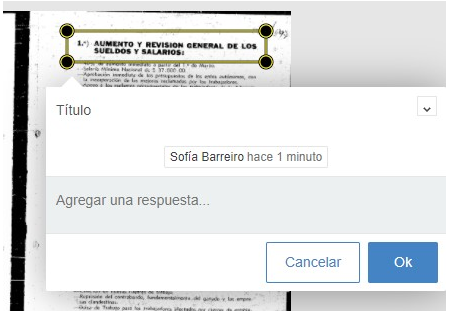

Figura 5.14: Visualización de una etiqueta creada

Finalmente Sofía termina de realizar el documento habiendo creado cinco etiquetas. Quedando conforme con su trabajo, presiona el botón “Guardar” y vuelve a la pantalla anterior. Allí ve que el primer documento cambió su estado a “Etiquetado” y su fila se encuentra de color verde (Figura 5.15).

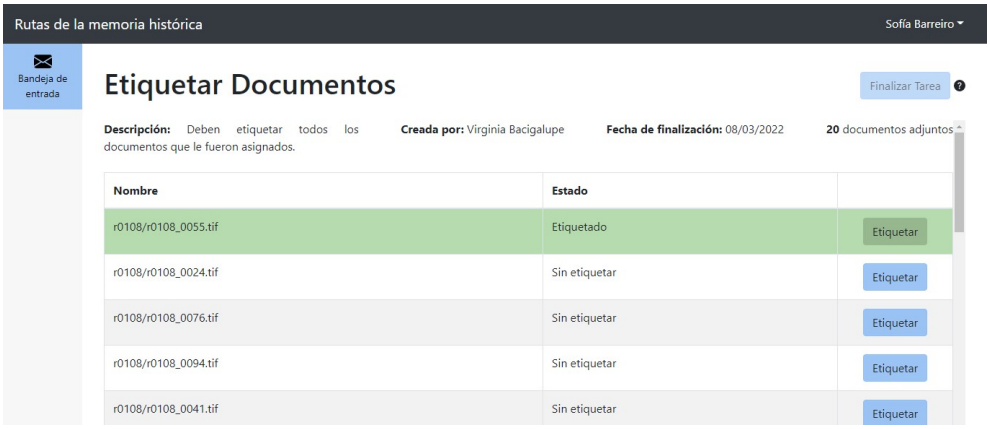

Figura 5.15: Vista de la tarea con un documento etiquetado

Sofía etiqueta seis documentos más, decide dejar la tarea por el día y sale de la plataforma. Al día siguiente, ingresa nuevamente y ve en su bandeja de entrada que el botón correspondiente a la tarea que comenzó ayer, dice “Continuar” (Figura 5.16).

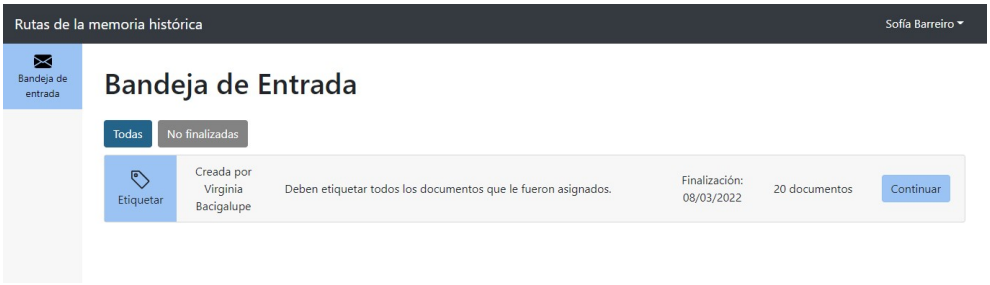

Figura 5.16: Bandeja de entrada de la alumna luego de haber comenzado la tarea

Presiona el botón de “Continuar” y vuelve a acceder al lista de documentos (Figura 5.17).

Figura 5.17: Vista de la tarea con varios documentos etiquetados

Allí busca el primer documento con estado “Sin etiquetar” y continúa con su trabajo. Una vez que etiquetó todos los documentos que le faltaban, ve que el botón “Finalizar tarea” se encuentra habilitado (Figura 5.18).

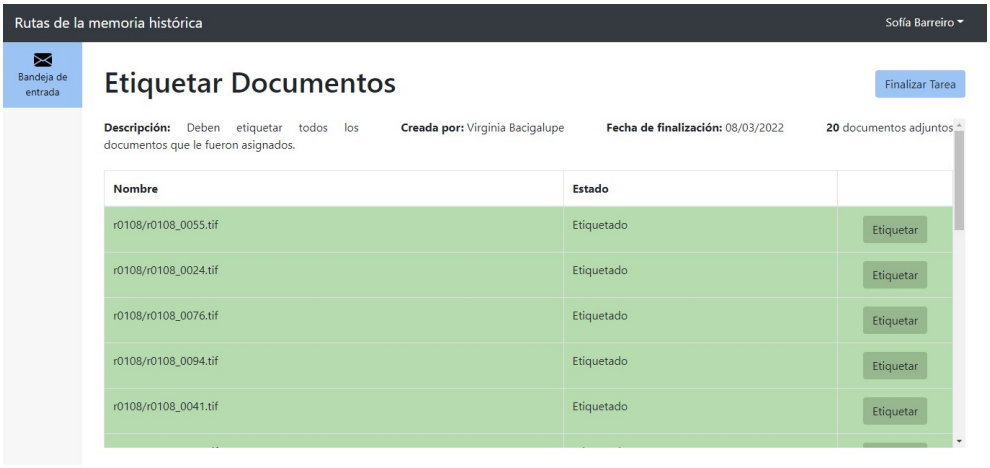

Figura 5.18: Vista de la tarea con el botón de “Finalizar Tarea” habilitado

Lo presiona, y es redirigida a su bandeja de entrada (Figura 5.19). En ese momento ve que ya no puede acceder más a la tarea y la misma se encuentra pintada de color gris con un mensaje que dice “FINALIZADA” en lugar de contar con un botón para acceder a la misma.

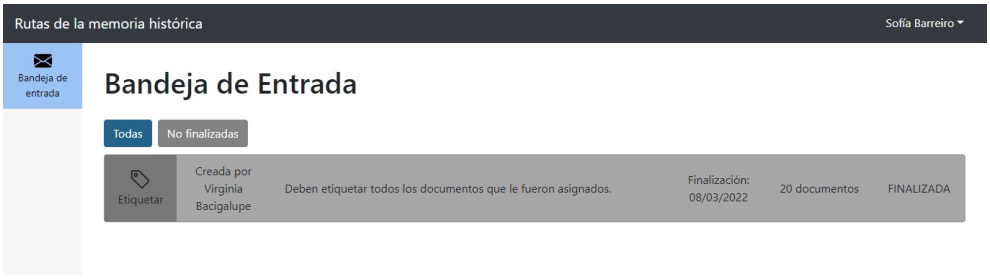

Figura 5.19: Vista de la bandeja de entrada con la tarea finalizada

Ese mismo día por la tarde, la docente Virginia se encuentra en la plataforma y al acceder a sus tareas creadas nota que su alumna Sofía ha finalizado la tarea que tenía asignada (Figura 5.20).

Figura 5.20: Sección “Tareas Creadas” de la docente con la tarea de Sofía finalizada

6 Conclusiones

En este capítulo se plantea la evaluación de la solución (sección 6.1) y mejoras que podrían considerarse en un trabajo futuro (sección 6.2).

6.1 Evaluación de la solución

El objetivo principal de este proyecto fue la creación de una plataforma web de gestión de tareas que facilite a usuarios de áreas no técnicas (profesionales, docentes y alumnos del área de humanidades) el análisis de documentos conformados por imágenes resultantes de escaneos.

Luego de finalizar el trabajo se puede concluir que se cumplieron satisfactoriamente los objetivos planteados al comienzo del presente proyecto de grado.

Se logró construir una plataforma web de gestión de tareas intuitiva, disponible para navegadores como Chrome y Firefox, la cual permite acceder a diferentes funcionalidades dependiendo del rol del usuario. Además, el acceso a cada recurso se encuentra protegido por un sistema de permisos y se registra toda la información asociada a las etiquetas creadas por un usuario sobre los documentos. Vale destacar que además se persisten todos los datos relacionados a usuarios, tareas, roles y grupos, los cuales son las entidades principales.

Por otro lado, el acceso a la plataforma se encuentra restringido únicamente a usuarios pertenecientes al proyecto de Memoria Histórica, por lo tanto, para poder ingresar, previamente se debe tener un usuario con los permisos necesarios. Además existen tres tipos de tareas que se pueden crear y llevar a cabo completamente dentro de la plataforma, siendo posible conocer el grado de avance en la ejecución de cada una de ellas. También es posible crear grupos desde la plataforma y asignarles tareas de etiquetado. De esta forma se facilita el trabajo del docente ya que se evita tener que crear y asignar tareas una por una.

Por otra parte se desarrolló una herramienta de etiquetado extensible con una interfaz amigable para el usuario, la cual surge como sustituta de *LabelMe*. *LabelMe* es la herramienta utilizada actualmente para llevar a cabo el etiquetado de los documentos pero presenta varias dificultades para los usuarios no técnicos. Las mismas fueron resueltas con el desarrollo de esta nueva herramienta de etiquetado. El etiquetador implementado permite la creación de etiquetas rectangulares y poligonales sobre cualquier parte del documento que el usuario considere pertinente. A su vez, se pueden crear comentarios sobre etiquetas ya existentes o eliminar las que no se consideren adecuadas.

En cuanto al objetivo planteado para el testing unitario, podemos afirmar que el mismo fue cumplido y superado.

Algo a destacar es que la herramienta ya se encuentra desplegada dentro del servidor de Memoria Histórica y por lo tanto disponible para cualquier usuario habilitado por la administración del proyecto. Al hablar con una estudiante del curso de la FIC que hoy en día utiliza la herramienta de etiquetado de escritorio, notamos el gran valor que implica que la plataforma este disponible en la web. Nos comentó que durante los dos años de pandemia el curso debió ser suspendido por no poder asistir a la universidad lo cual generó que la investigación se viera suspendida también. Esto no hubiera ocurrido de contar con una solución disponible desde la web.

Desde un principio, cuando se nos planteó la idea del proyecto, nos pareció una propuesta muy interesante desde un punto de vista ingenieril y tecnológico pero aún más desde un punto de vista humano, ya que somos conscientes que estamos contribuyendo a que se obtenga más información sobre un período que marcó la historia de nuestro país. El hecho de terminar nuestra carrera con un proyecto que, además de haber sido un desafío y un gran aprendizaje en lo profesional, sea de valor para la sociedad, nos llena de orgullo.

Por otra parte, consideramos que siempre se puede mejorar lo realizado y que puede ser útil agregar nuevas funcionalidades. Algunas ideas de posibles mejoras se detallan en la próxima sección.

6.2 Trabajo a futuro

Debido al alcance del proyecto y al tiempo estimado de duración del mismo, se decidieron dejar de lado algunas funcionalidades no prioritarias además de algunas optimizaciones de código.

A continuación se presentan posibles oportunidades de mejora para la solución desarrollada:

- Realizar una validación de la herramienta por parte de los usuarios de la FIC. Consideramos de gran valor evaluar la dificultad de adopción de la herramienta y realizar los ajustes pertinentes si fueran necesarios.

- En el etiquetador, crear tipos de etiquetas con opciones válidas precargadas. Se considera que esto podría facilitar el etiquetado para los alumnos como también el análisis de los datos recabados ya que los mismos serían uniformes.

- Agregar más filtros y un buscador en las secciones "Tareas creadas" y "Bandeja de entrada". Actualmente solo existen en la sección "Todas las tareas". Se considera que esto sería de utilidad en las demás secciones en el caso de existir gran cantidad de tareas en el sistema.

- Implementar una lógica de paginación al obtener usuarios, tareas, grupos y roles (si se considera necesario). Actualmente, se encuentra implementada únicamente sobre rollos y documentos. Por el contrario, para el resto de los recursos se obtiene la información completa, lo que puede generar demoras si la cantidad de datos es grande.

- Mostrar información más detallada del estado de ejecución de las tareas. Actualmente un usuario "Asignador" puede ver el estado de las tareas que creó (Nueva, Comenzada o Finalizada). Se considera que sería útil mostrar además la cantidad de documentos etiquetados hasta el momento para dar una noción más clara del avance de la ejecución de cada tarea.

- Establecer un comportamiento concreto al terminar de validar una tarea. En la actualidad, este tipo de tarea tiene el mismo comportamiento que las de tipo "Revisar" y no genera ningún efecto diferente sobre los documentos que se encuentran adjuntos a ella. Podría ser deseable que, por ejemplo, luego de finalizar una tarea de tipo "Validar" los documentos asociados a la misma ya no necesiten ser etiquetados nuevamente y por lo tanto no aparezcan en el listado de documentos al crear una tarea nueva de etiquetado.

- Realizar mejoras a nivel de código. A pesar de haber utilizado tecnologías modernas, se podrían realizar varios ajustes para hacer mejor uso de ellas. Por ejemplo, actualmente existe lógica de negocio relacionada al chequeo de permisos que se encuentra duplicada en el frontend y en el backend que podría ser consolidada en un único módulo Javascript común a ambos. De esta forma se aprovecharía la ventaja de estar utilizando el mismo lenguaje en ambos componentes. Por otro lado, siempre es posible refactorizar código, reduciendo el tamaño de los componentes y reutilizando lógica si es posible.

- Implementar tests de integración. Es de general conocimiento que el testing aporta calidad al código pero que requiere de un tiempo considerable. Debido a los tiempos del proyecto y a la existencia de requerimientos prioritarios, no se implementaron tests de este tipo en el presente proyecto. Sin embargo, todas las funcionalidades de la plataforma fueron probadas manualmente en profundidad teniendo en cuenta todos los posibles casos. Dicho esto, de todos modos se cree que agregar este tipo de tests podría ser una mejora útil a futuro.

- Integrar sistemas automatizados a la plataforma que ejecuten la creación de múltiples tareas de etiquetado automáticamente. La arquitectura de la plataforma soporta este tipo de agente y se considera que serían útiles para este tipo de tareas.

Bibliografía

[1] Lorena Etcheverry y col. «A computational framework for the analysis of the Uruguayan dictatorship archives». En: *Proceedings of the Conference on Digital Curation Technologies (Qurator 2021), Berlin, Germany, February 8th - to - 12th, 2021*. Ed. por Adrian Paschke y col. Vol. 2836. CEUR Workshop Proceedings. CEUR-WS.org, 2021. URL: http://ceur-ws.org/Vol-2836/qurator2021%5C_paper%5C_20.pdf.

[2] Bryan C. Russell y col. «LabelMe: A Database and Web-Based Tool for Image Annotation». En: *Int. J. Comput. Vis.* 77.1-3 (2008), págs. 157-173. DOI: 10.1007/s11263-007-0090-8. URL: https://doi.org/10.1007/s11263-007-0090-8.

[3] Apache. *Apache HTTP Server: mod_proxy*. Disponible en: https://httpd.apache.org/docs/2.0/mod/mod_proxy.html#forwardreverse. (Última consulta: 23/02/2022).

[4] Mathias Weske. *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer, 2019. ISBN: 978-3-662-59431-5. DOI: 10.1007/978-3-662-59432-2. URL: https://doi.org/10.1007/978-3-662-59432-2.

[5] Ramez Elmasri y Shamkant B. Navathe. *Fundamentals of Database Systems, 3rd Edition, chapter 7*. Addison-Wesley-Longman, 2000. ISBN: 978-0-8053-1755-8.

[6] Mozilla. *JavaScript | MDN*. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript. (Última consulta: 13/10/2021).

[7] Mozilla. *Lenguaje de programación dinámico - Glosario | MDN*. Disponible en: https://developer.mozilla.org/es/docs/Glossary/Dynamic_programming_language. (Última consulta: 13/10/2021).

[8] Manz. *¿Qué es la asincronía? - Javascript en español - Lenguaje JS*. Disponible en: https://lenguajejs.com/javascript/asincronia/que-es/. (Última consulta: 13/10/2021).

[9] Mozilla. *Express/Node introduction - Learn web development | MDN*. Disponible en: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction. (Última consulta: 15/10/2021).

[10] w3schools. *Node.js Introduction*. Disponible en: https://www.w3schools.com/nodejs/nodejs_intro.asp. (Última consulta: 15/10/2021).

[11] w3schools. *Node.js Modules*. Disponible en: https://www.w3schools.com/nodejs/nodejs_modules.asp. (Última consulta: 15/10/2021).

[12] TutorialsTeachers. *Node.js Modules*. Disponible en: https://www.tutorialsteacher.com/nodejs/nodejs-modules. (Última consulta: 15/10/2021).

[13] Mozilla. *Introducción a Express/Node - Aprende sobre desarrollo web | MDN*. Disponible en:https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction. (Última consulta: 15/10/2021).

[14] Mozilla. *Primeros pasos en React - Aprende sobre desarrollo web | MDN*. Disponible en: https://developer.mozilla.org/es/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started. (Última consulta: 16/10/2021).

[15] Simon Hoyos. *¿Qué es JSX? | by Simon Hoyos | Medium*. Disponible en: https://medium.com/@simonhoyos/qu%C3%A9-es-jsx-95006a2f94f9. (Última consulta: 16/10/2021).

[16] Dan Abramov y los autores de la documentación de Redux. *Redux Fundamentals, Part 2: Concepts and Data Flow | Redux*. Disponible en: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow. (Última consulta: 16/10/2021).