UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Salt.pro: herramienta de gestión de tickets de traducción

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Saliasi , Lisa

Tutor/a: Valderas Aranda, Pedro José

Cotutor/a externo: MENDOZA GUASCH, ANA MARIA

CURSO ACADÉMICO: 2021/2022

Resumen

El presente Trabajo de Fin de Grado cubre el desarrollo en Java desde cero de una nueva herramienta de gestión de tiques de traducción que sustituye a la antigua utilizada por los funcionarios de la GVA. La plataforma antigua arrastra una serie de problemas principalmente de usabilidad y rendimiento, que invitan a que no se reutilicen las tecnologías con las que está implementada. Los nuevos requisitos para este desarrollo serán comentados más detalladamente en este documento, así como también nuevas tecnologías utilizadas como Bootstrap, Spring Boot, Hibernate o PostgreSQL.

Palabras clave: herramienta, plataforma, GVA, tecnologías, Spring Boot.

Resum

El present Treball de Fi de Grau cobreix el desenvolupament a Java des de zero d'una nova eina de gestió de tiquets de traducció que substitueix a l'antiga utilitzada pels funcionaris de la GVA. La plataforma antiga arrossega una sèrie de problemes principalment d'usabilitat i rendiment, que conviden que no es reutilitzen les tecnologies amb les quals està implementada. Els nous requisits per a aquest desenvolupament seran comentats més detalladament en aquest document, així com també noves tecnologies utilitzades com Bootstrap, Spring Boot, Hibernate o PostgreSQL.

Palabras clave: eina, plataforma, GVA, tecnologies, Spring Boot.

Abstract

This Final Degree Project covers the development in Java from scratch of a new translation ticket management tool that replaces the old one used by GVA officials. The old platform carries a series of problems, mainly of usability and performance, which invite to not reuse the technologies with which it is implemented. The new requirements for this development will be discussed in more detail in this document, as well as new technologies used such as Bootstrap, Spring Boot, Hibernate or PostgreSQL.

Keywords : tool, platform, GVA, technologies, Spring Boot.

Tabla de contenidos

Tabla de ilustraciones

1. Introducción

Hoy en día la tecnología está muy vinculada a la vida de cada persona. Debido a esto ha pasado de ser importante a ser necesaria. En nuestras casas podemos encontrarla en cada rincón, a veces sin darnos cuenta. Prácticamente cada gestión la podemos realizar desde la comodidad de nuestra casa, desde concertar una cita con el médico, realizar clases de forma *on-line* hasta pedir que nos entreguen la cena en casa.

Con esto dicho, podemos deducir que la tecnología está presente en casi todos los campos laborales y esta tecnología evoluciona a pasos de gigante. A raíz de esta evolución es muy importante la actualización, accesibilidad y profesionalidad para la satisfacción de los usuarios. Por esta razón nos hemos visto obligados a sustituir unas herramientas por otras.

En este documento se detallará la aportación realizada para el desarrollo de la nueva plataforma desde cero que sustituirá a la antigua plataforma Salt.pro como medida tanto para incrementar el rendimiento de esta como para mejorar el flujo de trabajo.

1.1 Motivación

Como se ha dicho anteriormente la tecnología va de la mano con una constante evolución, por lo que es muy importante estar actualizados con las últimas tecnologías para tener un mayor rendimiento. La motivación que impulsa el cambio que se lleva a cabo en la plataforma de Salt.pro es la experiencia de todos los usuarios que podrán trabajar en un entorno desarrollado con tecnologías innovadoras.

Por otro lado, a nivel personal ha sido una gran motivación poder poner en práctica todos los conocimientos adquiridos en la rama de *Tecnologías de la Información* como en mi periodo de prácticas en la empresa donde he aprendido cómo se trabaja en equipo, a utilizar herramientas profesionales y desarrollar con tecnologías y lenguajes que me ayudarán a lo largo de mi trayecto profesional.

1.2 Objetivos

Los objetivos principales de este proyecto son las mejoras en el rendimiento respecto de la actual plataforma Salt.pro (almacenamiento de archivos fuera de base de datos, mejora en la carga de plantillas, etc.) así como el rediseño de la interfaz y simplificación del flujo de trabajo (simplificación de tareas delimitadas a los roles adecuados, rediseño de tareas habituales, rediseño de listados y elementos de pantalla, etc.).

Como otros objetivos secundarios, se pidieron, entre otros:

- Gestión de roles para delimitar las peticiones a las que tienen acceso los usuarios de la plataforma (usuarios, técnicos lingüísticos y técnicos externos, administradores).
- Evitar que el administrador tenga que validar las altas de usuario (comprobación automática de que el usuario es funcionario de la Generalitat).
- Gestión de sesiones de usuario para poder controlar quién y cuándo accede a la plataforma.
- Comunicación ágil entre los usuarios de la plataforma sin tener que recurrir obligatoriamente a herramientas ajenas a ella (teléfono, correo electrónico, etc.).
- Publicación de documentación de ayuda al usuario (Preguntas Más Frecuentes y Manual de Usuario).
- Obtención de estadísticas de uso de la plataforma.

1.3 Estructura de la memoria

Este documento está estructurado de la siguiente forma, en primer lugar, una **introducción** en qué consiste el desarrollo del proyecto, las motivaciones que han inspirado la implementación de la nueva plataforma y objetivos planteados en las distintas fases de desarrollo.

En segundo lugar, **Gestión de tiques de traducción en la GVA**, se explica la situación actual de la plataforma Salt.pro incluyendo el diseño tecnológico modular actual, la implicación de las distintas tecnologías y los requisitos planteados.

En tercer lugar, la **Metodología** utilizada para estructurar, planificar y controlar el proceso de desarrollo de la plataforma.

En cuarto lugar, **Análisis de Requisitos,** se explica la toma de requisitos, se desarrollan los casos de uso, los escenarios y diagrama de clases para dar soporte a los requisitos.

En quinto lugar, **Diseño**, se trata del modelo de datos y prototipos de la interfaz de la aplicación.

En sexto lugar, **Desarrollo de la solución**, se explica la arquitectura de la aplicación y como interactúa con plataformas externas. Se presentan las herramientas, el lenguaje de programación y los patrones utilizados.

En séptimo lugar, **Producto desarrollado**, se explica cómo se utiliza la aplicación para cumplir con los escenarios propuestos anteriormente.

En octavo lugar, **Pruebas unitarias**, se explican las pruebas unitarias realizadas para comprobar el funcionamiento correcto del código.

Por último, en el apartado **Conclusiones,** se muestras ciertas conclusiones a las que se han llegado una vez finalizado el desarrollo y posibles mejoras en versiones posteriores.

2. Gestión de tiques de traducción en la GVA

El actual funcionamiento de Salt.pro es muy similar al de un gestor de tiques, en el que un usuario solicita que se realice una acción y uno o varios técnicos se encargan de dar una respuesta al usuario.

2.1 Situación actual

En Salt.pro, actualmente tenemos 3 tipos de usuarios: los usuarios básicos, los técnicos de traducción, y los administradores.

Los usuarios básicos que, salvo excepciones muy contadas (por ejemplo, miembros de entes instrumentales), son funcionarios de la Generalitat Valenciana, pueden solicitar sobre los documentos que suben a la plataforma las siguientes acciones:

- Traducción del documento del castellano al valenciano, con revisión opcional del documento traducido.
- Traducción del documento del valenciano al castellano, con revisión opcional del documento traducido.
- Únicamente corrección del documento subido.

Una vez enviada la solicitud a Salt.pro, ésta queda pendiente de ser atendida por un técnico de traducción. Los técnicos, al acceder a Salt.pro, ven un listado de todas las tareas del sistema, algunos datos básicos de las mismas, y en qué estado se encuentran. Con este listado de tareas los técnicos de traducción pueden:

- Ver detalles de la tarea de traducción, incluidos algunos datos sobre el solicitante, como un teléfono o un e-mail de contacto, y estadísticas básicas como cuántas páginas y palabras tiene el documento.
- Asignarse la tarea de traducción.
- Descargar el documento sobre el cual se solicita una acción.
- Descargar la traducción automática que se ofrece a través del servicio web de *Apertium* para utilizarla como base del trabajo.
- Subir a la plataforma el documento ya traducido o revisado.
- Añadir anotaciones breves sobre la tarea.
- Marcar la tarea como finalizada para que ésta esté disponible para el usuario solicitante.
- Consultar tareas ya realizadas y descargadas. Las tareas ya realizadas que han sido descargadas por el solicitante desaparecen del listado de tareas, aunque su información permanece en base de datos. Un técnico puede utilizar el buscador de Salt.pro para consultar los datos de estas tareas de traducción que ya no están disponibles para el resto de los usuarios.

Los administradores del sistema pueden realizar las mismas tareas que los técnicos de traducción, pero además pueden:

- Dar de alta o de baja a los usuarios de Salt.pro. Actualmente el alta de los usuarios en el sistema se realiza solicitándolo a un administrador a través de un correo electrónico. Un administrador se encarga de verificar que el usuario es funcionario o un usuario autorizado a solicitar traducciones y, en caso afirmativo, darlo de alta en el sistema.
- Eliminar tareas de traducción. Si, por ejemplo, por cualquier razón un documento sobre el que solicita traducción se sube dañado y no puede ser procesado, la tarea de traducción es rechazada y eliminada del sistema por un administrador.
- Obtener estadísticas sobre las traducciones realizadas.

Una vez la acción solicitada por el usuario es llevada a cabo sobre el documento y el técnico ha marcado la tarea como finalizada, Salt.pro envía un *e-mail* al usuario solicitante. Desde este momento, el usuario puede acceder a la plataforma y descargarse el documento traducido/revisado.

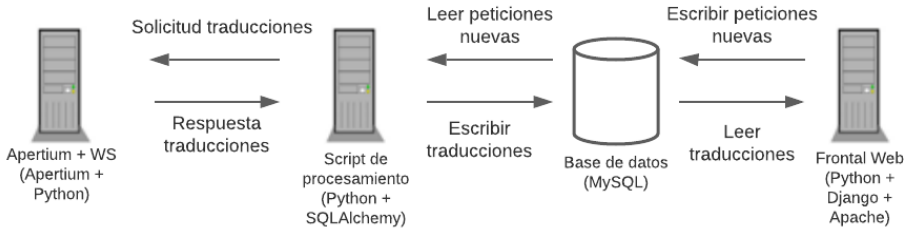

Ilustración 2.1: Diseño tecnológico modular de la antigua aplicación de Salt

El diseño tecnológico modular actual de Salt.pro implica la intervención de varias tecnologías, scripts y servicios muy diversos.

Los usuarios interactúan con el sistema a través de una interfaz web desarrollada sobre *Django*, un *framework* de desarrollo de aplicaciones sobre *Python*. Este *framework* permite un desarrollo de aplicaciones web de forma sencilla, con integración de autenticación de usuarios e interacción con varios sistemas de bases de datos

El principal problema con el que nos encontramos es que puede resultar insuficiente a la hora de desarrollar soluciones complejas como la que nos ocupa.

Cada acción de los usuarios a través de la interfaz tiene su reflejo en tareas escritas sobre una base de datos *MySQL*. Sin embargo, *MySQL* no es el sistema de base de datos elegido por la Generalitat para integrar en la mayoría de sus sistemas, por lo que es altamente recomendable utilizar el sistema de base de datos de *Oracle* o *PostgreSQL*.

Existe un script controlando los cambios de estado de las tareas que se ejecuta en segundo plano cada cierto tiempo. En cada ejecución el script comprueba si hay tareas en alguno de los estados que tiene establecidos, realiza las acciones pertinentes, y escribe el estado resultante en base de datos. Dicho script está escrito en *Python* y se comunica con la base de datos mediante la librería *SQLAlchemy*.

El uso del script de control de Salt.pro introduce un cierto retardo entre las acciones que ejecutan los usuarios sobre la interfaz y el momento en que estos ven reflejado en el sistema el resultado a sus acciones.

Todo documento que se sube al sistema y toda traducción automática que se obtiene de *Apertium* se almacena directamente en base de datos. Esto, a la larga, provoca un cuello de botella considerable en la base de datos, además de ser altamente ineficiente.

2.2 Requisitos planteados

A continuación, se detallan algunos requisitos funcionales y no funcionales planteados a desarrollar:

Requisitos funcionales

- RF1: Simplificar la interfaz y los flujos de trabajo

La interfaz de funcionamiento actual de *Salt.pro*, aunque efectiva, es confusa y poco intuitiva. Realizar tareas sencillas como por ejemplo la ordenación de elementos de un listado o la eliminación de un documento requieren de más pasos de los necesarios, los cuales se podrían simplificar usando otras tecnologías.

El flujo de trabajo de *Salt.pro*, desde la subida del documento hasta la obtención del resultado deseado, implica varias fases de trabajo y el concurso de varios participantes con diversos roles. La interfaz está diseñada para aprovechar al máximo los elementos comunes de las tareas de estos participantes, basándose en la ocultación y muestra de componentes de la interfaz según el rol, simplificando así la tarea de desarrollo de la interfaz.

Se propone rediseñar la interfaz para mejorar la facilidad de uso de la herramienta utilizando tecnologías más modernas, como por ejemplo *Bootstrap*, que permiten ampliar el abanico de posibilidades de diseño y funcionamiento de la interfaz.

- RF2: Creación de solicitudes

La aplicación debe permitir crear solicitudes de traducción y/o corrección de documentos mediante un formulario.

- RF3: Consulta de solicitudes abiertas (rol Usuario)

La aplicación debe permitir consultar las solicitudes a los usuarios solicitantes un listado de las solicitudes abiertas por ellos. El listado debe incluir las tareas no abiertas cuya antigüedad no supere los 7 días.

- RF4: Consulta de solicitudes recibidas (rol Administrador)

La aplicación debe permitir consultar a los administradores de la aplicación un listado de las solicitudes creadas en el sistema para poder determinar si están correctamente cumplimentadas antes de que un técnico se pueda asignar la tarea.

- RF5: Consulta de solicitudes no asignadas (rol Administrador / rol Técnico)

La aplicación debe permitir consultar a los administradores de la aplicación un listado de las solicitudes validadas, pero no asignadas aún a ningún técnico. A su vez, los técnicos lingüísticos deben poder acceder a este mismo listado.

- RF6: Detalle de tarea

La aplicación ofrecerá a usuarios, técnicos y administradores un detalle de los datos de una tarea, en el que se mostrarán datos referentes a la creación y procesamiento de la solicitud, datos referentes al fichero original subido y datos referentes al usuario solicitante.

- RF7: Buscador de tareas procesadas

Se permitirá realizar búsquedas de tareas ya procesadas (finalizadas, descargadas, canceladas o rechazadas) a los usuarios de la aplicación mediante un formulario. Los usuarios solicitantes podrán consultar mediante este formulario solicitudes propias, los técnicos lingüísticos podrán consultar solicitudes ya procesadas que le fueron asignadas en el pasado, y los administradores podrán consultar cualquier solicitud.

- RF8: Preguntas más frecuentes

La aplicación tendrá una sección dedicada a Preguntas Más Frecuentes acerca de la aplicación y del proceso de traducción. La gestión estará limitada a los administradores de la aplicación.

- RF9: Estadísticas (rol Administrador)

Los administradores podrán consultar un formulario con una serie de estadísticas referentes a las solicitudes procesadas en la aplicación. Además, también podrá consultar un desglose de las tareas procesadas y por procesar en función de su estado. El formulario de estadísticas permitirá filtrar por usuario solicitante, técnico, Conselleria del solicitante y un rango de fechas a determinar.

- RF10: Modo Automático (rol Administrador)

La aplicación debe establecer un mecanismo para que las ausencias de los administradores de la aplicación ya sean por vacaciones, bajas médicas o cualquier otro motivo, no supongan una paralización del flujo de trabajo de los técnicos.

Se controlará la ausencia de todos los administradores con un Modo Vacaciones, que deberá ser activado por el último administrador que se vaya a ausentar. Dicho Modo Vacaciones se activará desde el panel de configuración, al cual se puede acceder mediante la opción de menú “Supervisión > Configuración”.

Mientras el Modo Vacaciones esté activado, las tareas que se den de alta en la aplicación por parte de los usuarios se aplicarán los siguientes criterios:

- Las tareas se validarán automáticamente (pasarán directamente a estado “No Asignada”).
- Las tareas de traducción por defecto no requerirán valorar la necesidad de una revisión por parte de un administrador.

Se mostrará a todos los administradores un aviso en pantalla indicando que el Modo Vacaciones está activado. Los criterios automáticos de este modo se aplicarán hasta que un administrador lo desactive.

- RF11: Externalización de tareas en Modo automático

Se permitirá la selección entre la validación y la externalización automáticas cuando el modo automático se active. Si se selecciona la validación automática, el funcionamiento continuará siendo el previsto en los requisitos iniciales de la aplicación.

En cambio, si se selecciona la externalización automática, las tareas pasarán automáticamente al listado de tareas asignables a técnicos externos.

Los criterios automáticos seleccionados en el desplegable se aplicarán hasta que un administrador desactive el Modo automático.

- RF12: Exportación de estadísticas

Se desea poder exportar tanto la página de estadísticas como la página de información de la carga de trabajo de los técnicos. Se implementará, por tanto, un botón que permita exportar esta información a un fichero descargable en formato CSV, que podrá ser leído e impreso en papel según se desee.

El botón para exportar estadísticas se insertará tanto en la página principal de estadísticas como en la página de información de carga de trabajo de los técnicos. En la página principal de estadísticas, si se aplican filtros de búsqueda, el resultado de la exportación de estadísticas deberá ser la exportación de la información filtrada.

Requisitos no funcionales

- RNF1: Adecuar código al umbral de calidad Sello de Excelencia

El código de la aplicación debe cumplir con los estándares de calidad establecidos en el Sello de Excelencia de la Generalitat Valenciana. Los criterios del Sello de Excelencia son:

- Ausencia de evidencias bloqueantes
- Ausencia de evidencias críticas
- Mínimo de 20% de cobertura de código por parte de las pruebas unitarias
- Máximo de 5% de líneas de código duplicadas
- Calificación de Seguridad A por parte del software SonarQube
- Mínimo de calificación de Mantenibilidad B por parte del software SonarQube
- Mínimo de calificación de Fiabilidad C por parte del software SonarQube

- RNF2: Seguridad

Para autenticarse en la aplicación se utiliza el sistema corporativo de Autenticación, Autorización, Auditoría y SSO, GVLOGIN.

GVLOGIN audita los accesos de los usuarios a los diferentes sistemas de información, registrando DNI, IP, fecha y hora de acceso, sistema de autenticación utilizado (usuario de dominio Generalitat, usuario de dominio Educación, usuario correo GVA, Certificado digital, Cl@ve), y resultado del acceso (correcto o fallido).

Por otro lado, para la restricción de URLs a los roles adecuados se utilizará el módulo Spring Security de Spring.

Spring Security es un marco Java que proporciona autenticación, autorización y otras funciones de seguridad a aplicaciones empresariales.

2.3 Flujo de trabajo

La nueva plataforma desarrollada funciona a modo de gestor de tiques, donde cada tique representa una tarea de traducción o revisión solicitada, y que es gestionado por un técnico del Servicio de Política Lingüística. A continuación, se analizarán las fases del flujo de trabajo de Salt.pro relacionando cada una de ellas con las atribuciones de los roles implicados en cada una de ellas. *El flujo de trabajo para técnicos y técnicos externos es idéntico.*

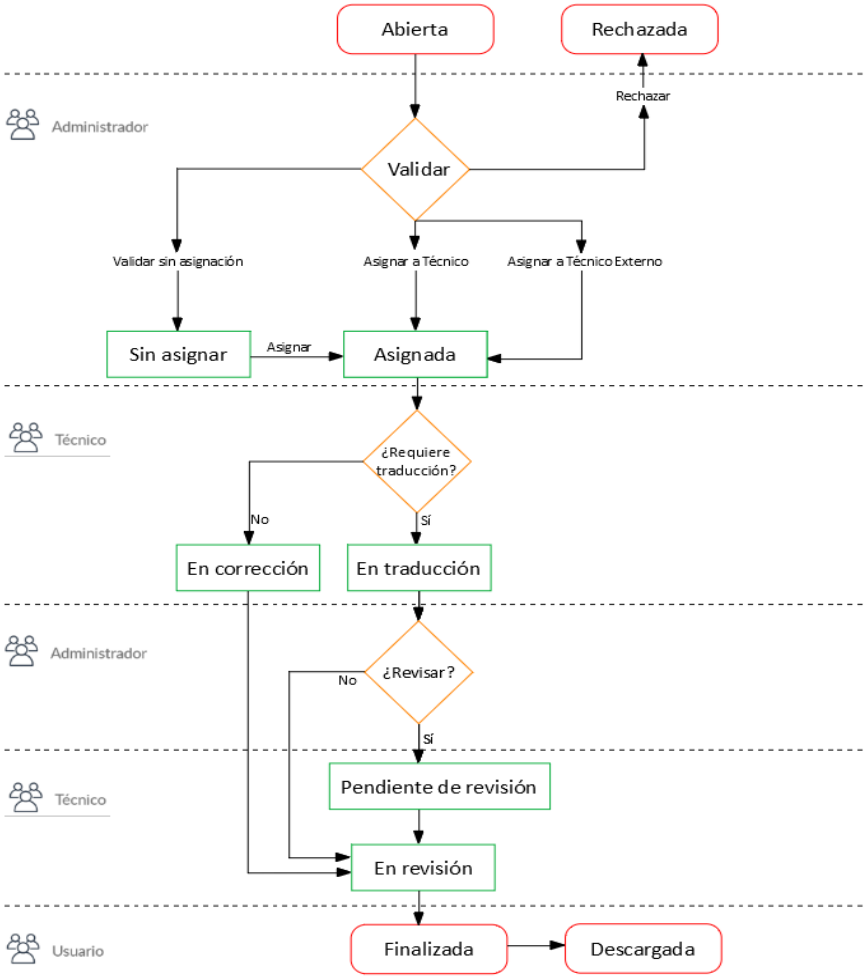

Ilustración 2.2: Diagrama de flujo de trabajo

2.3.1 Gestión de tareas

- ***Abierta: Nueva solicitud de traducción / corrección***

El flujo de trabajo comienza cuando un usuario autorizado accede a la plataforma y rellena una solicitud de traducción y/o revisión de un texto, y sube el archivo en cuestión que debe ser procesado. En dicha solicitud debe indicar:

- Tipo de tarea: traducción del castellano al valenciano, traducción del valenciano al castellano o únicamente corregir un texto que ya está en valenciano.
- El documento sobre el que se desea trabajar.
- Una temática del documento que actuará a modo de título de la tarea.
- La prioridad de su tarea, entre las disponibles.
- Opcionalmente, comentarios adicionales sobre la tarea.

Para evitar que el usuario deba estar pendiente constantemente de la plataforma, se le notifica por correo electrónico cualquier cambio de estado de su tarea solicitada y, en cualquier caso, puede consultar al momento el estado de su tarea en cualquiera de las fases de su procesamiento.

A partir de este estado, se abren dos vías de acción que dependen de los administradores de la plataforma: Validar o Rechazar.

- ***Validar***

Una vez se registra una solicitud de traducción / corrección en la plataforma, ésta pasa a un listado de tareas pendientes de validación. Un administrador, accede a la plataforma para validar las tareas que han abierto los usuarios, comprobando que el archivo se ha subido correctamente y valorando las anotaciones que haya hecho el solicitante.

En caso de que la tarea sea correcta y se deba realizar, el administrador valida la tarea. La validación podrá ser de dos tipos: sin asignación de técnico (pasa directamente a estado “Sin asignar”) o con asignación de técnico.

Si la asignación la realiza específicamente el administrador, ésta podrá ser directa (la tarea pasa a estado “Asignada”) o configurable, es decir, el administrador puede indicar un grupo de técnicos para que se hagan cargo de la tarea. En este caso la tarea pasa a estado “Sin asignar”, a la espera de que alguno de los técnicos indicados se asigne la tarea.

- ***Rechazar***

En caso de que el documente presente alguna deficiencia y no se deba realizar la tarea, un administrador rechaza la tarea, que pasa a estado “Rechazada”. Ni los técnicos lingüísticos ni los técnicos externos pueden ver estas tareas, únicamente el usuario que día de alta la tarea y que, además, recibe una notificación por correo electrónico.

- ***Sin asignar***

Si una tarea ha sido validada sin asignación de técnico, esta se encuentra en estado “Sin asignar”, a la espera de que uno de los técnicos disponibles la vea en el listado de tareas pendientes de asignar y se la asigne a sí mismo.

Si la tarea ha sido validada por un administrador con asignación configurable de un grupo de técnicos, esta únicamente aparecerá como “Sin asignar” a aquellos técnicos que el administrador haya designado.

- ***Asignada***

Una vez se asigne la tarea, ya sea por parte de un administrador o del propio técnico, pasará al estado “Asignada”, pendiente de que se empiece a trabajar con ella.

Cuando el técnico lingüístico indique en la plataforma que ya está trabajando en la tarea, esta cambiará de estado:

  - Si la tarea es de traducción, pasará a estado “En traducción”.
  - Si la tarea es sólo de corrección, pasará a estado “En corrección”.

- ***En traducción***

La tarea permanecerá en este estado mientras se realiza la traducción por parte de un técnico.

En este estado, el técnico dispondrá de la opción de descargar el documento original subido por el solicitante a la plataforma o una traducción automática del texto original, que será ofrecida por el servicio web de *Apertium*, pero únicamente para aquellos documentos cuyo formato esté soportado por *Apertium*.

Una vez se haya completado la traducción, se pulsará en la opción de cambio de estado “Cambiar estado”. El sistema le solicitará al técnico el archivo traducido y, tras esto, la tarea pasará a uno de los siguientes estados:

  - Si el administrador que validó la tarea a su entrada en el sistema indicó que era necesario valorar una revisión, ésta pasará al estado “En valoración” (etiquetado como “¿Revisar?” en el diagrama de estados).
  - Si se indicó que no era necesario valorar una revisión, la tarea pasará directamente al estado “Finalizada”.

- ***En corrección***

La tarea permanecerá en este estado mientras se realiza la corrección.

Igual que en el estado “En traducción”, se dispondrá de una opción para descargar el documento original, pero no la traducción automática pues sólo estarán en este estado aquellas tareas para las que únicamente se ha solicitado una corrección del documento.

Una vez se haya completado la corrección, se pulsará la opción de cambio de estado “Finalizar”. La plataforma le solicitará al técnico el archivo corregido y, tras esto, la tarea pasará al estado “Finalizada”.

- ***En valoración***

La tarea permanecerá en este estado hasta que un administrador valore si sobre la tarea procesada es necesario realizar una revisión o no. El administrador descargará el documento traducido/corregido y, si decide que hace falta una revisión del documento, asignará la revisión a un grupo configurable de uno o más técnicos. La tarea quedará en estado “Pendiente de revisión”, a la espera de que uno de los técnicos indicados inicie la revisión.

Si, por el contrario, decide que la revisión no es necesaria, el administrador pasará la tarea directamente a estado “Finalizada”.

- ***Pendiente de revisión***

Las tareas en estado “Pendiente de revisión” aparecerán en un listado, accesible por los técnicos. Un técnico podrá asignarse estas tareas a sí mismo, pero la tarea no cambiará de estado hasta que no seleccione la opción del menú de cambio de estado de la plataforma “Iniciar revisión”. Una vez seleccionada esta opción, la tarea pasará al estado “En revisión”.

Si la tarea ha sido asignada por un administrador directamente a un técnico, no pasará por el listado por lo que no será visible al resto de técnicos, únicamente al técnico asignado, que podrá acceder a ella a través de su listado de tareas asignadas.

- ***En revisión***

La tarea permanecerá en este estado mientras se realiza la revisión.

Igual que en el estado “En traducción”, dispondrá de una opción para descargar el documento original, la traducción automática, o la traducción del documento ya realizada en una fase anterior del flujo de trabajo.

Una vez se haya completado la revisión, se pulsará la opción de cambio de estado “Finalizar”. La plataforma le solicitará al técnico el archivo revisado y, opcionalmente, los comentarios adicionales que crea oportunos. Tras esto, la tarea pasará a estado “Finalizada”.

- ***Finalizada***

Las tareas completadas se marcarán con este estado. El sistema se encargará de notificar al usuario solicitante por correo electrónico que su solicitud de traducción/corrección ya ha sido finalizada.

Una vez notificado por parte del sistema, el usuario solicitante podrá acceder a la tarea y descargar el documento traducido o corregido en el mismo formato que subió el documento original. Todos estos documentos traducidos y corregidos de tareas finalizadas también estarán disponibles para consulta por parte de los administradores y los técnicos.

- ***Descargada***

Una vez que el usuario haya accedido a la plataforma y se haya descargado la traducción/corrección solicitada, la tarea finalizada pasará a estado “Descargada”. De esta forma se podrá controlar por parte de los administradores de Salt.pro cuándo se ha descargado un documento un usuario solicitante, con el objetivo de poder avisarle si transcurre un periodo de tiempo demasiado elevado.

- ***Cancelada***

Una tarea que, por el motivo que sea, haya sido marcada como cancelada por un administrador o por el propio usuario solicitante, pasará a estar en estado “Cancelada” y no aparecerá en ningún listado, salvo en las búsquedas en el histórico de tareas por parte de un administrador o del propio usuario solicitante.

2.3.2 Administración de la plataforma

En lo que se refiere a la administración de la plataforma, el sistema debe estar orientado a facilitar en la medida de lo posible las tareas del administrador, respondiendo a la delimitación de roles definida anteriormente.

Es por ello por lo que, aunque funcionalmente un administrador pueda realizar las mismas acciones que un técnico, se ha diseñado la plataforma para que un administrador pueda centrarse en el rol propio que se le ha definido: validar y supervisar tareas.

- ***Validación de tareas***

El administrador es el rol que tenga la visión más completa del sistema. Esto es debido a que, además de las tareas pendientes de asignar, en curso o ya finalizadas, puede ver aquellas tareas que han sido solicitadas, pero aún no han sido aprobadas. Es el encargado, pues, de aprobar o rechazar las tareas de traducción o corrección que llegan a la plataforma, antes incluso de que los técnicos se las puedan asignar.

Por otro lado, los administradores son también los únicos con la capacidad de rechazar una tarea. Desde el listado de tareas pendientes de validar, puede rechazar una solicitud de traducción bien sea porque no se ha subido bien el fichero, bien porque no está en el idioma esperado, o por el motivo que sea, y que debe ser especificado, en cualquier caso.

En caso de rechazo, el cliente será notificado del hecho y deberá abrir una nueva solicitud.

Desde el mismo listado de tareas pendientes de aprobar, un administrador puede, como se ha mencionado, aprobar una tarea, momento desde el cual estará disponible para que los técnicos se la puedan asignar. Por otro lado, también tiene la capacidad de asignar directamente al técnico que él decida en el momento de la aprobación en función de la carga de trabajo o de los motivos que se crean oportunos.

- ***Gestión de tareas aprobadas***

Un administrador puede llevar a cabo varias acciones sobre tareas aprobadas que el técnico no es capaz de realizar. Es el caso de la modificación de parámetros de la tarea en curso, como por ejemplo la prioridad.

Referente a la prioridad, se implementan los mecanismos necesarios para que el sistema soporte la priorización de tareas con 2 niveles de urgencia: el establecido por el usuario solicitante y el establecido por un administrador. Por defecto, el valor de ambos niveles de urgencia es el mismo y adopta el valor de la prioridad establecida por el usuario.

Desde el listado de tareas aprobadas un administrador también tiene la posibilidad de cancelar tareas de traducción que han sido previamente aprobadas, aunque debe justificar el motivo. Una vez marcadas como canceladas, las tareas de traducción/revisión en este estado dejan de estar disponibles para ser gestionadas por parte de los técnicos, los cuales son notificados de este hecho.

Un administrador también tiene la capacidad de cambiar el técnico asignado a una tarea, en caso de que se decida, por ejemplo, que un técnico tiene mucha carga de trabajo y se decida asignar esa tarea a otro técnico.

- ***Añadir contenido a las PMF***

La gestión de los enlaces disponibles en la sección de Preguntas Más Frecuentes es responsabilidad de los administradores. Ellos son los encargados de añadir, eliminar

y gestionar los contenidos de esta sección de ayuda que se muestra a los usuarios solicitantes.

- ***Modo vacaciones***

Cuando un administrador sepa que, por la razón que sea, no va a estar disponible y sus funciones no pueden ser realizadas por otro administrador (o administradores), que tampoco están disponibles, tiene la opción de activar el Modo vacaciones. Su cometido es que aquellas partes del flujo de trabajo en las que interviene un administrador no sean un cuello de botella cuando estas partes no puedan ser supervisadas por los administradores.

Una vez activado el Modo vacaciones, la validación de tareas y el procesamiento de traducciones que requieran valorar si es necesario o no revisar la traducción es automática. En este caso, se establecen las siguientes reglas por defecto:

  - Cuando un usuario dé de alta una tarea, ésta se valida automáticamente en el sistema, pasando automáticamente a estar disponible para cualquier técnico lingüístico.
  - Las tareas validadas en modo vacaciones automáticamente llevan activada la opción para valorar si es necesaria o no una revisión de la traducción.
  - Las tareas que lleguen a estado Pendiente de Revisión automáticamente pasan a requerir revisión y se asignan al mismo técnico que realizó la traducción.

- ***Otras tareas de administración***

Otras tareas que pueden llevar a cabo los administradores son:

  - Obtener diversas estadísticas acerca de las tareas procesadas y del uso general de la plataforma.
  - Obtener una estadística para medir la carga de trabajo de los técnicos y técnicos externos.

3. Metodología

El objetivo de este apartado es explicar en qué consiste la metodología GvLOGOS que sigue el desarrollo del proyecto Salt.pro.

La metodología GvLOGOS ha sido desarrollada por la Dirección General de Tecnologías de la Información con el objetivo de definir los procesos y métodos de trabajo que se han de emplear en la gestión de proyectos, servicios, incidencias y cambios desde el momento de la demanda hasta su entrega final, pasando por los preceptivos puntos de control de calidad y de seguridad, así como definir roles y proporcionar herramientas y plantillas para llevarlo a cabo.

A continuación, se muestra un diagrama de dicha metodología.

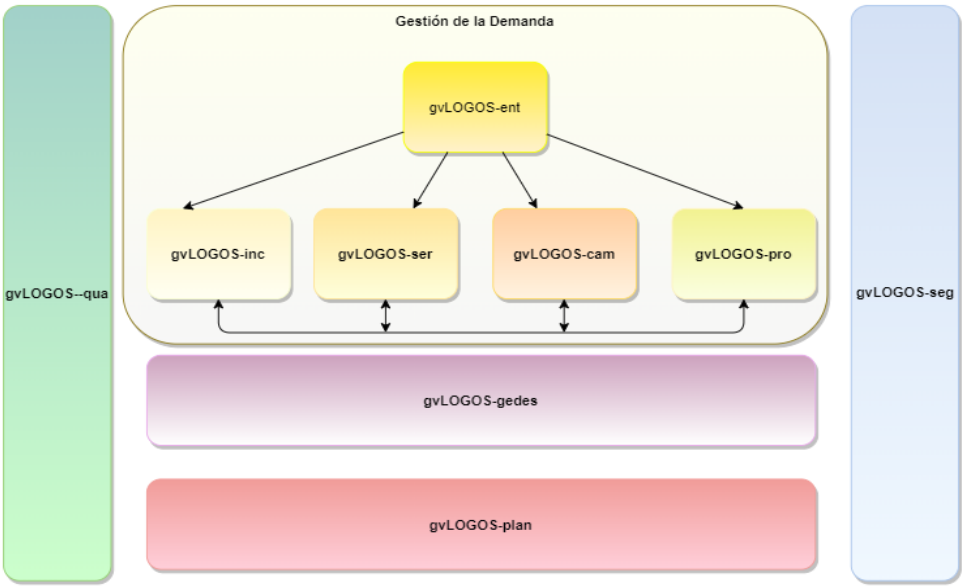

Ilustración 3.1: Metodología GvLOGOS. Fuente: DGTIC

La metodología está compuesta por:

- **Gestión de la demanda**, que abarca:
  - **GvLOGOS-ent [1]**: Gestión de entradas (CAU o nivel 1 de atención al usuario).

    La gestión de entradas trata de dar respuesta a un evento generado por un usuario de la DGTIC para informar de una incidencia, de la existencia de una nueva necesidad (petición de servicio) o un cambio en algún servicio(proyecto).
  - **GvLOGOS-inc [1]**: Gestión de incidencias.

    La gestión de incidencias trata de dar respuesta principalmente a los siguientes tipos de incidencia:
    - Correctivos en desarrollo del software,
    - Incidencias relacionadas con la infraestructura (suelen afectar la capacidad y la disponibilidad de los servicios),

- Quejas sobre los servicios, la infraestructura, los procedimientos etc. que pueden efectuar los usuarios.

- **GvLOGOS-ser [1]**: Petición de servicio.

  Se define como tal toda solicitud en la que un usuario indica que desea suscribirse o acceder a algún servicio del que la DGTIC es responsable.

  Generalmente las peticiones de servicio deberán ser aprobadas por un Comité de Decisión (CD).

- **GvLOGOS-cam [1]**: Gestión de cambios.

  Se define una petición de cambio cuando se solicita que un determinado servicio del que ya disfruta el solicitante y del cual es responsable la DGTIC sea modificado de algún modo con le objetivo de adaptarlo a nuevas regulaciones o mejorarlo para que se adapte mejor a sus necesidades.

- **GvLOGOS-pro [1]**: Gestión de proyectos.

  Este proceso permite la gestión de los proyectos de la DGTIC como pueden ser:

  1. El desarrollo de una nueva aplicación o módulo,
  2. Una nueva instalación que implica diferentes infraestructuras,
  3. El desarrollo evolutivo de una aplicación que afecte a varios elementos,
  4. Un cambio perfectivo en una o más aplicaciones o infraestructuras,
  5. Un cambio adaptativo del entorno tecnológico de una aplicación
  6. Una nueva explotación de datos,
  7. Un estudio de viabilidad,
  8. Gestión del conocimiento.

En este apartado se pueden distinguir las diferentes fases que abarca el desarrollo de un proyecto.

En primer lugar, está la fase de **Análisis** donde se lleva a cabo la descripción de módulos funcionales, el análisis de casos de uso y las interacciones e integración.

Se realiza también la elaboración del modelo de datos (conceptual y lógico) junto con la elaboración del modelo de procesos, la definición de interfaces de usuario, correspondencia Casos de Uso / Interfaz de Usuario, correspondencia Clases / Interfaz y correspondencia Requisito / Caso de Uso.

En segundo lugar, se encuentra la fase de **Diseño** donde se realiza la definición y diseño de la arquitectura del sistema, el diseño lógico (modelo relacional o diseño de clases), el diseño de procesos, el diseño físico de datos y el diseño de la interfaz de usuario.

También se establecen en esta fase los perfiles o roles, migraciones y dependencias con otros sistemas.

Por último, se lleva a cabo la fase de **Desarrollo** donde se implementa todo lo que se diseñó y se acordó anteriormente por un equipo de desarrollo con las tecnologías propuestas.

- **Gestión de calidad** – GvLOGOS-qua:

·La gestión de calidad tiende a garantizar que la organización o un producto sea consistente y tiene cuatro componentes:

1. Planeamiento de la calidad
2. Control de la calidad
3. Aseguramiento de la calidad
4. Mejoras en la calidad

- **Gestión de Entregas y Despliegues Software** – GvLOGOS-gedes [1]

Se define ante la necesidad de gestionar de forma homogénea las entregas de las aplicaciones de la DGTIC y establecer unos controles de calidad previos al despliegue.

Se define una política, unos procesos y procedimientos a seguir en cada entrega de software.

Para cada proceso se especifican los roles participantes, las actividades a realizar y las herramientas que van a ofrecer un soporte adecuado a dicha gestión.

Con ello se dispone de un catálogo de aplicaciones y un catálogo de librerías completo y de calidad.

Los actores principales que participan en GEDES son:

1. Usuario experto
2. Gestor de entregas
3. Oficina de entregas y calidad del SW
4. Sistemas

Se definen tres entornos en la Nueva Infraestructura de la Ciudad Administrativa NICA (desarrollo, preproducción y producción).

En el entorno de desarrollo se preparan las tareas para que sea el propio equipo de desarrollo el que se autoevalúe sin exigirle nada desde la oficina de entregas.

Es en pase a preproducción y producción donde se exigen unos controles de calidad (pruebas unitarias y análisis del código estático).

GvLOGOS-gedes comienza en fases tempranas del desarrollo con la creación del repositorio en subversión. Estableciendo en este punto unas primeras pautas como son unificar la estructura de todos los repositorios, que alberga tanto las fuentes como la documentación de la aplicación y los scripts de BBDD.

A continuación, se muestra un diagrama de la estructura de directorios para almacenar la documentación y las fuentes de aplicaciones Java según GvLOGOS.

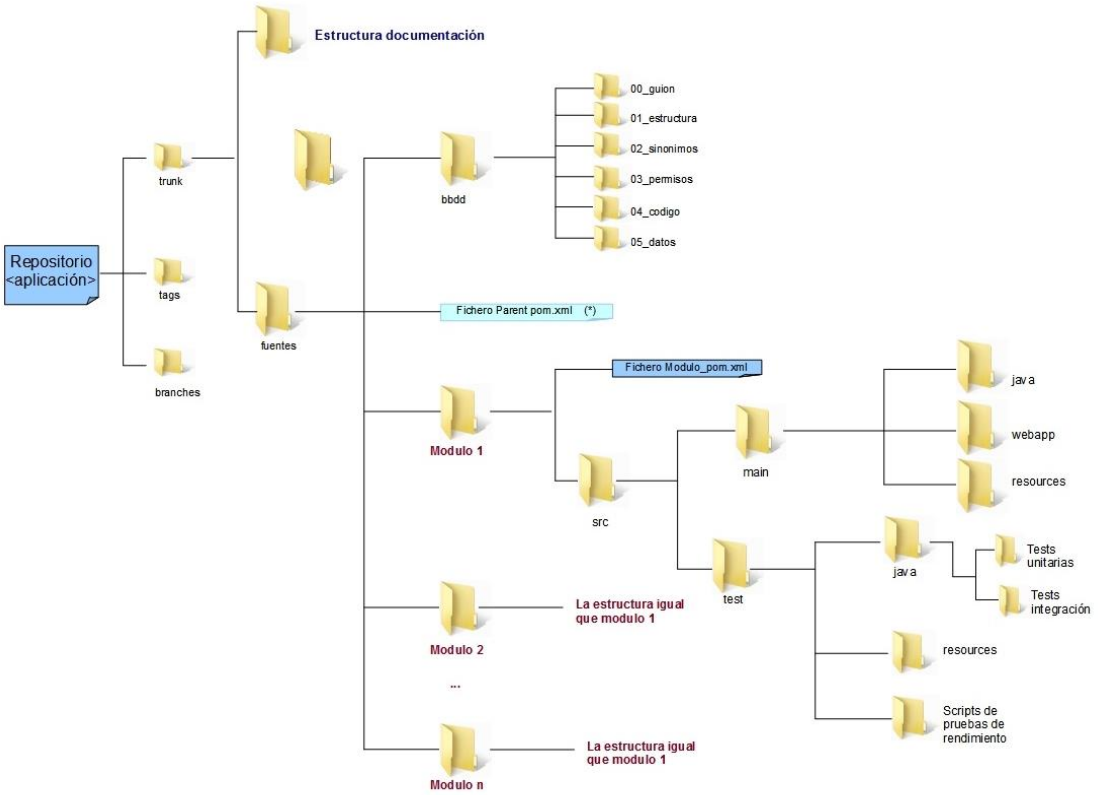

Ilustración 3.2: Estructura de directorios según GvLOGOS. Fuente: DGTIC

- **Gestión de planificación y seguimiento** – GvLOGOS-plan

La planificación de proyectos es una programación o distribución de actividades en el tiempo y una gestión de recursos para minimizar el coste del proyecto cumpliendo con los condicionantes exigidos de: plazo de ejecución, tecnología a utilizar, recursos disponibles, nivel máximo de ocupación de dichos recursos, etc. El seguimiento permite revisar el cumplimiento de plazos y objetivos, así como detectar desviaciones.

- **Gestión de seguridad** – GvLOGOS-seg

La gestión de la seguridad debe velar por que la información sea correcta y completa, esté siempre a disposición del negocio y sea utilizada sólo por aquellos que tiene autorización para hacerlo, es decir, debe preservar:

  - Confidencialidad: la información debe ser sólo accesible a sus destinatarios predeterminados.
  - Integridad: la información debe ser correcta y completa.
  - Disponibilidad: debemos tener acceso a la información cuando la necesitamos.

4. Análisis de requisitos

En este capítulo se presenta el análisis de requisitos realizado siguiendo la metodología GvLogos. Este análisis de ha realizado utilizando el modelo UML de Casos de uso y el Diagrama de clases

.

4.1 Casos de Uso y Escenarios

UML es el lenguaje de modelado de sistemas de software más utilizado actualmente, respaldado por el OMG (*Object Managment Group*). Este lenguaje sirve efectivamente para representar procesos utilizando para ello elementos gráficos.

Este lenguaje dispone de distintos tipos de diagramas de los cuales se hará uso en este apartado el diagrama de casos de uso.

El diagrama de casos de usos que se utiliza a continuación ofrece una visión general de los actores involucrados en el sistema, las funciones que emplean estos actores y como interactúan.

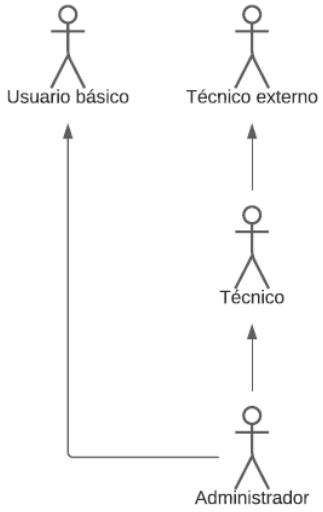

Ilustración 4.1: Diagrama de relaciones entre actores

En la Ilustración 4.1 se pueden observar los diferentes actores de este sistema que ya se han mencionado anteriormente.

A continuación, se procede a mostrar los casos de usos de cada uno de estos actores, teniendo en cuenta lo explicado anteriormente que los actores técnico y técnico externo emplean exactamente las mismas funciones.

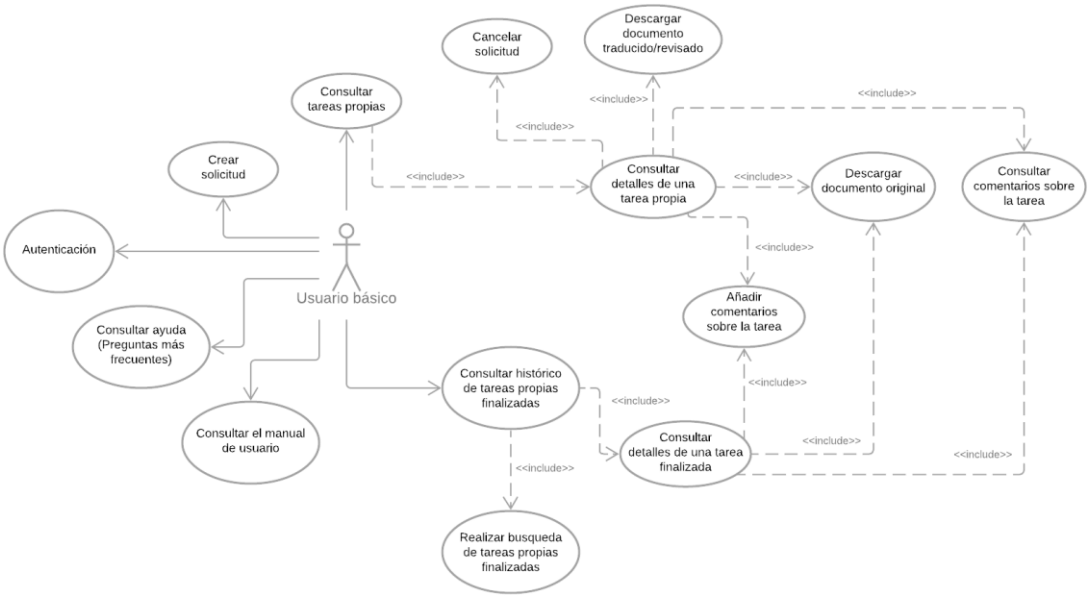

Ilustración 4.2: Diagrama de casos de uso para el usuario básico

Se puede apreciar en el diagrama el uso de la etiqueta <<*include*>> o inclusión. Esto ocurre porque algunos procesos incluyen sistemáticamente a otros. Eso es, por ejemplo, el caso de uso “Consultar detalles de una tarea propia” incluye al caso de uso “Añadir comentarios sobre la tarea” porque la instanciación de este último utiliza siempre el flujo de eventos de “Consultar detalles de una tarea propia”.

A continuación, se muestran los diferentes casos de uso para el rol de técnico y administrador.

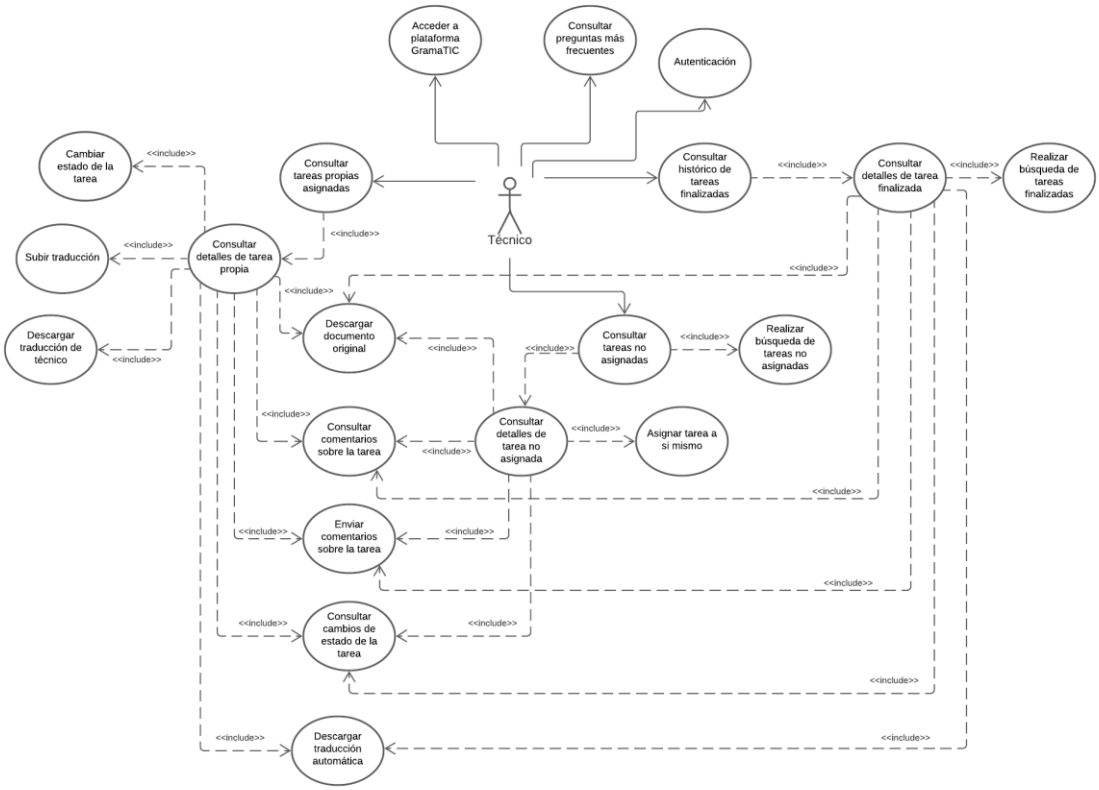

Ilustración 4.3: Diagrama de casos de uso para el rol de técnico

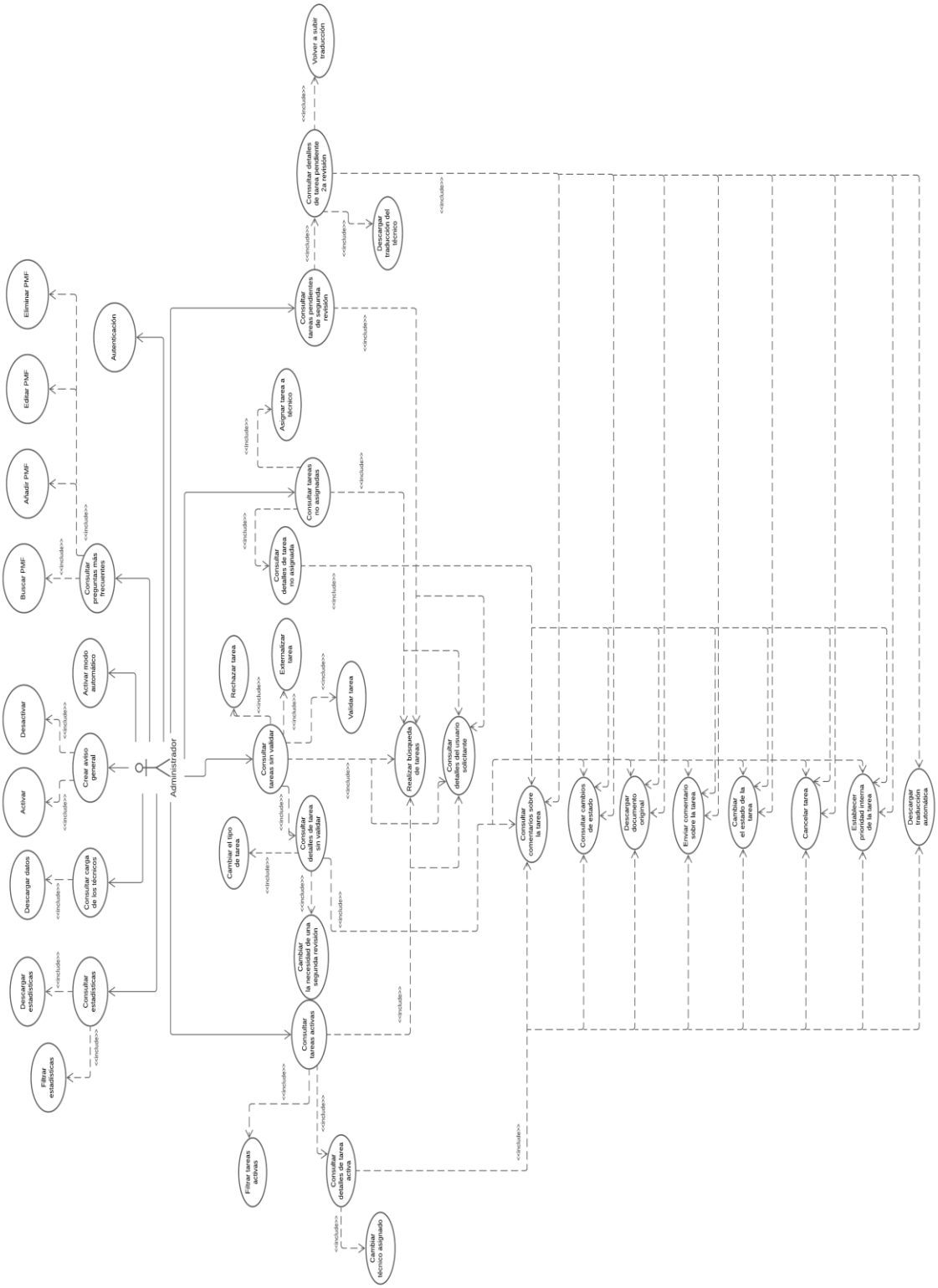

Ilustración 4.4: Diagrama de casos de uso para el rol de administrador

A continuación, se muestran algunos de los escenarios posibles más relevantes.

- Escenario 1: Crear una solicitud

  Cualquier usuario básico o administrador autenticado en el sistema puede dar de alta una tarea creando una solicitud de traducción o revisión.

  1. El usuario accede a la página “Crea una solicitud”.
  2. Se muestra un formulario a rellenar con varios campos.
     a. Tipo de tarea
     b. Fichero
     c. Tema del documento
     d. Prioridad
     e. Comentarios adicionales
  3. El usuario lo rellena y lo envía.
  4. El sistema guarda en la base de datos los detalles de la tarea nueva.
  5. Finalmente, el sistema redirige al usuario a la página de detalle de la tarea recién creada.

- Escenario 2: Buscar una tarea procesada

  Cualquier usuario con el rol Administrador puede buscar en el listado de tareas procesadas o finalizadas una o varias tareas mediante una búsqueda por filtrado.

  1. El usuario accede a la página de “Histórico”.
  2. Pulsa el botón de “Mostrar los filtros de búsqueda”.
  3. Se muestra al usuario un formulario con varios filtros.
  4. El usuario lo rellena y lo envía.
  5. El sistema obtiene la información de acuerdo con los filtros introducidos y la prepara previo a mostrarla.
  6. El sistema muestra un listado con las tareas encontradas. Si ninguna tarea se encuentra se le informa al usuario.

- Escenario 3: Asignarse una tarea a uno mismo.

  Cualquier técnico puede asignarse una tarea del listado de “Tareas No Asignadas”.

  1. De la barra del menú el técnico selecciona la opción “No asignadas”.
  2. Elige en el listado la tarea que desea asignarse.
  3. Pulsa el botón de “Asigna” y se asigna la tarea.
  4. El sistema asigna al técnico la tarea y redirige a la página del detalle de tarea.

- Escenario 4: Finalizar una tarea

  Cualquier técnico después de iniciar y realizar la traducción o revisión de un documento puede dar por finalizada la tarea asociada al documento.

1. Accede al listado de “Mis tareas” desde la barra del menú.
2. Selecciona la tarea a finalizar.
3. Pulsa el desplegable “Cambia el estado” y elige “Termina”.
4. En la ventana que aparece sube el fichero traducido y pulsa “Termina”.
5. El sistema guarda el fichero subido y marca la tarea con el estado “Finalizado”.

- Escenario 5: Descargar fichero de estadísticas

  Un usuario con el rol Administrador puede descargar un fichero en formato CSV con las estadísticas del sistema.
  1. El administrador selecciona de la barra del menú el desplegable “Supervisión”.
  2. Elige la opción “Estadísticas”.
  3. Desplazarse al final de la página.
  4. Pulsar el botón “Descarga las estadísticas”.

- Escenario 6: Mostrar un aviso general en el sistema.

  Cualquier usuario con el rol Administrador puede hacer aparecer un mensaje de aviso general en la aplicación.
  1. El administrador pulsa sobre el desplegable “Supervisión” que se muestra en la barra del menú.
  2. Elige la opción “Configuración”.
  3. Activa el aviso general en el caso de que estuviera desactivado.
  4. Escribe el aviso y lo guarda.
  5. El sistema guarda la información y procede a mostrarla en el panel de “Aviso General” visible para todos los usuarios del sistema.

4.2 Diagrama de clases

A continuación, se muestra el diagrama de clases que ayuda a mostrar las distintas entidades que forman parte del sistema y como se relacionan entre ellas. Una clase se compone por sus atributos y métodos. Todos los objetos que provienen de la misma clase tendrán el mismo comportamiento y conjunto de atributos. En este caso, un objeto podría ser un usuario, una tarea o un documento. Se indican en el diagrama todos los atributos de todas las entidades.

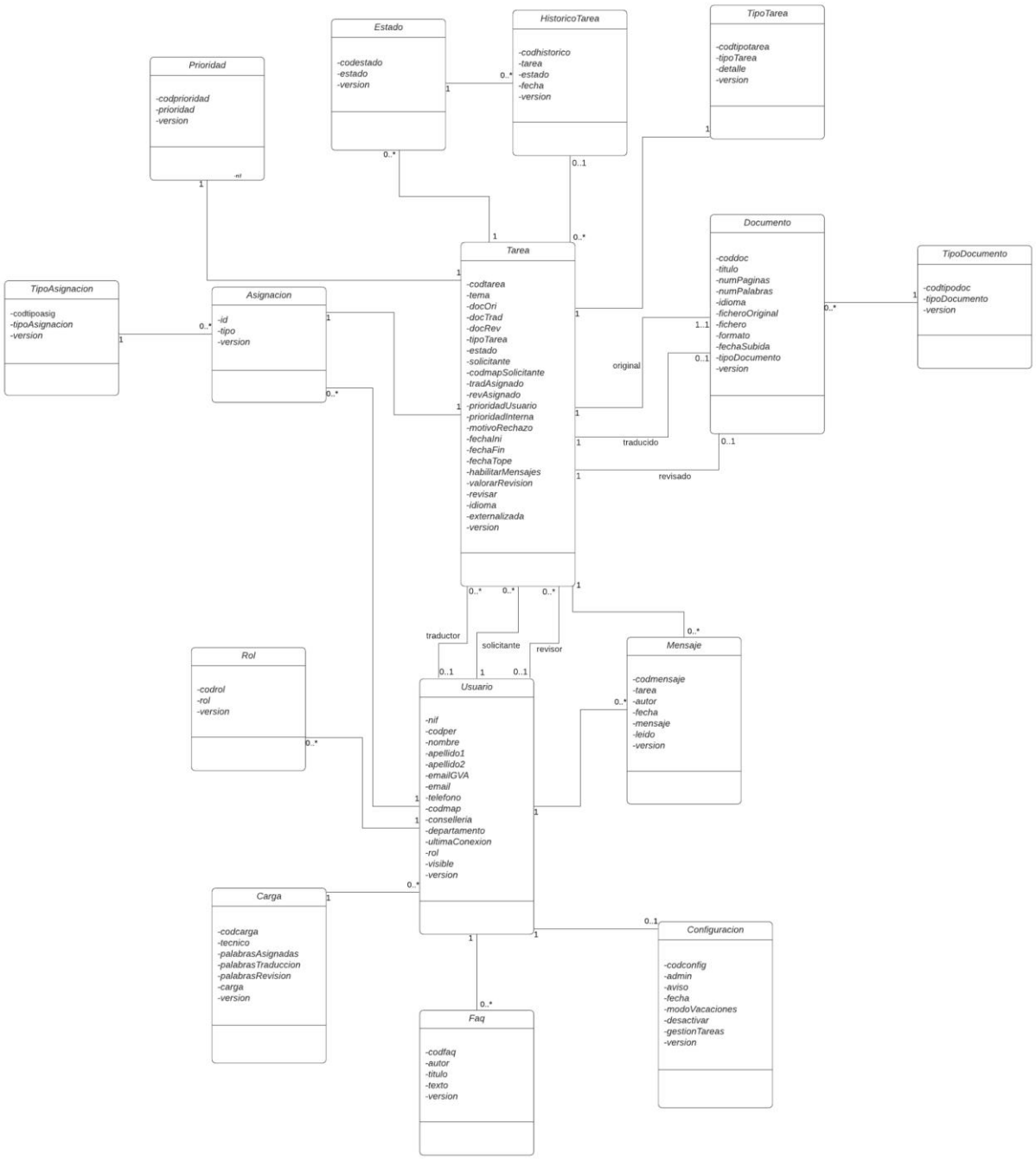

Ilustración 4.5: Diagrama de clases

Se procede a explicar en qué consisten algunas de las clases implementadas:

- Tarea: representa todas las tareas de traducción o revisión que un usuario solicitante puede solicitar en los idiomas castellano o valenciano.
- Usuario: representa los usuarios solicitantes, administradores y técnicos lingüísticos.
- Documento: representa el documente presente en la tarea sobre el que se realiza la traducción o revisión solicitada. Además, representa también los documentos traducidos o revisados por los técnicos.
- Rol: representa los distintos roles que pueden tener los usuarios de Salt.pro.

- Asignación: representa la asignación de las tareas a los técnicos lingüísticos
- Carga: representa la carga de trabajo que tiene cada técnico lingüístico.
- Faq: representa las preguntas frecuentes que pueden registrar los usuarios.
- Configuración: representa la configuración que pueden realizar los administradores de algunos aspectos de la plataforma como por ejemplo los avisos generales que aparecen o la gestión automática de tareas.

5. Diseño

En este capítulo se describe el modelo de la base de datos acompañado con el diagrama de la base de datos. Además, se presentan los prototipos iniciales de las distintas páginas que conforman el diseño del sistema.

5.1 Modelo de la Base de Datos

En este apartado se comentará la parte de la arquitectura más alejada de la visión del usuario y que representa la base de toda la estructura: la base de datos, la encargada de almacenar los datos de la aplicación.

Se trata de una base de datos *PostgreSQL* a la que se puede añadir mediante el cliente *pgAdmin* o en este caso accedemos mediante el cliente *DBeaver* comentado más adelante en las herramientas utilizadas.

La ilustracoón 5.1 muestra una representación de las tablas de la base de datos. Se pueden observar de esta forma las 15 tablas que componen la base de datos y sus relaciones:

- **salt_prioridad:** Mantiene la información relacionada con que tareas están asignadas a que técnico.
- **salt_estado:** Contiene información sobre los diferentes estados en los que se puede encontrar una tarea.
- **salt_historico_tarea:** Para cada tarea mantiene la información de todos los estados por los que ha pasado.
- **salt_tipo_tarea:** Contiene la información de los distintos tipos de tarea que existen (traducción de castellano a valenciano y viceversa o revisión)
- **salt_tipo_asignacón:** Almacena información sobre el tipo de asignación eso es, de traducción o revisión
- **salt_asignacion:** Mantiene la información relacionada con que tareas están asignadas a que técnico.
- **salt_tarea:** Contiene toda la información de cada tarea del sistema.
- **salt_documento:** Contiene toda la información de todos los documentos que han entrado el sistema.
- **salt_tipo_documento:** Contiene información sobre los distintos tipos de documentos que nos encontramos (original, traducido, revisado, obsoleto).
- **salt_rol:** Contiene información sobre los distintos roles (usuario, técnico, técnico externo o administrador)
- **salt_usuario:** Almacena la información de todos los usuarios que tienen acceso al sistema.
- **salt_mensaje:** Guarda todos los mensajes de cada una de las tareas.
- **salt_carga:** Almacena información sobre la carga de los técnicos.
- **salt_faq:** Guarda información acerca de las preguntas frecuentes que se observan en la plataforma.

- **salt_configuracion:** Guarda solamente la información de la configuración actual. Si la configuración se cambia, entonces la nueva configuración sobreescribe a la antigua en la base de datos.

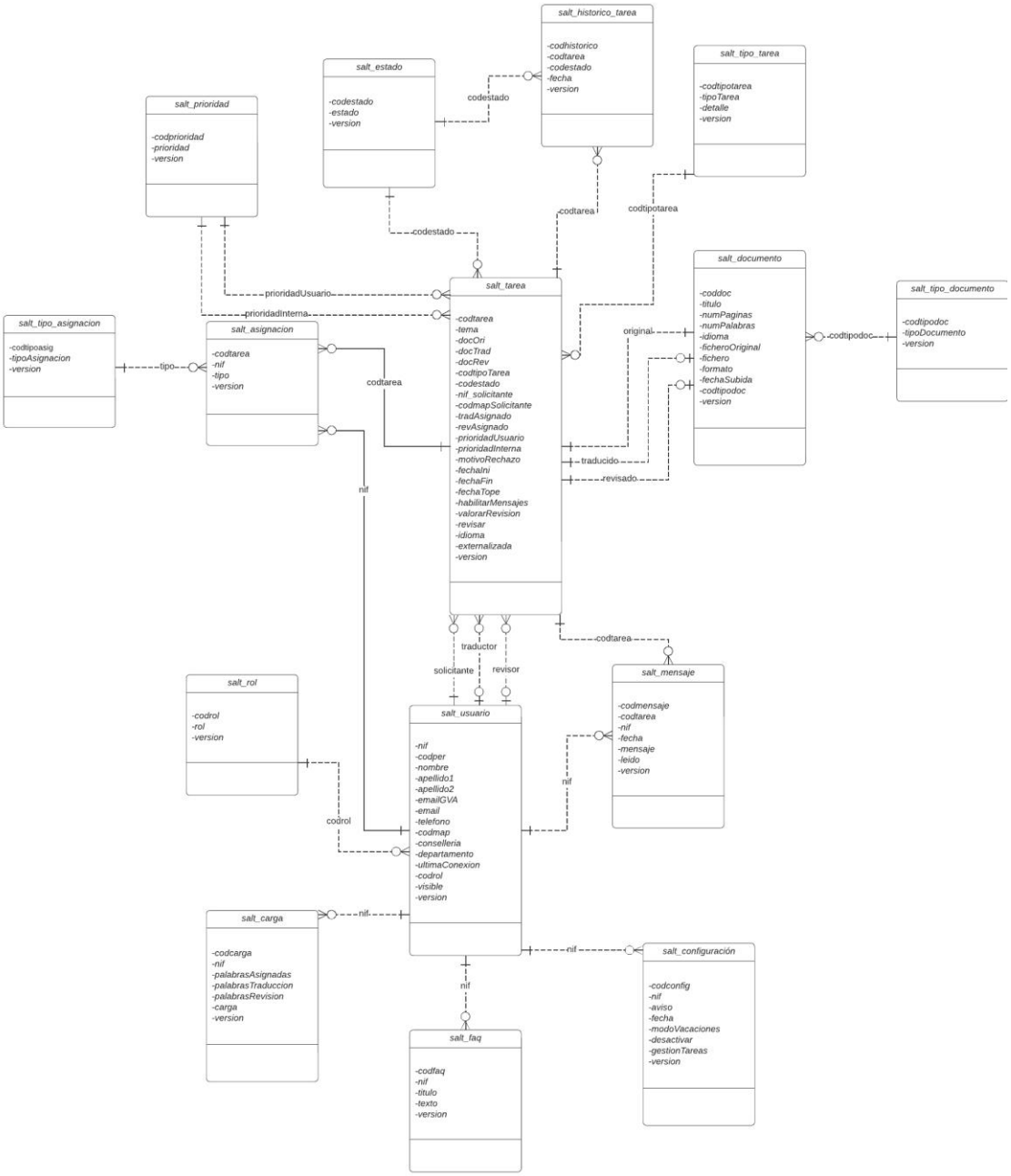

Ilustración 5.1: Modelo de la base de datos

5.2 Prototipos de la interfaz de la aplicación

Un prototipado es una representación de un sistema interactivo. Su objetivo es explorar los aspectos interactivos del sistema, incluyendo usabilidad y funcionalidad.

A continuación, se mostrará el prototipado realizado mediante *Bootstrap*. Dicho prototipado inicial ha ido evolucionando y variando a pequeña escala según las peticiones del cliente y según los requisitos adicionales que se han ido estableciendo.

En la siguiente ilustración se muestra el prototipado de la página del listado de tareas propias para el rol de usuario. Podemos destacar en esta pantalla el listado de tareas y el menú que llevará a las demás pantallas de la aplicación.

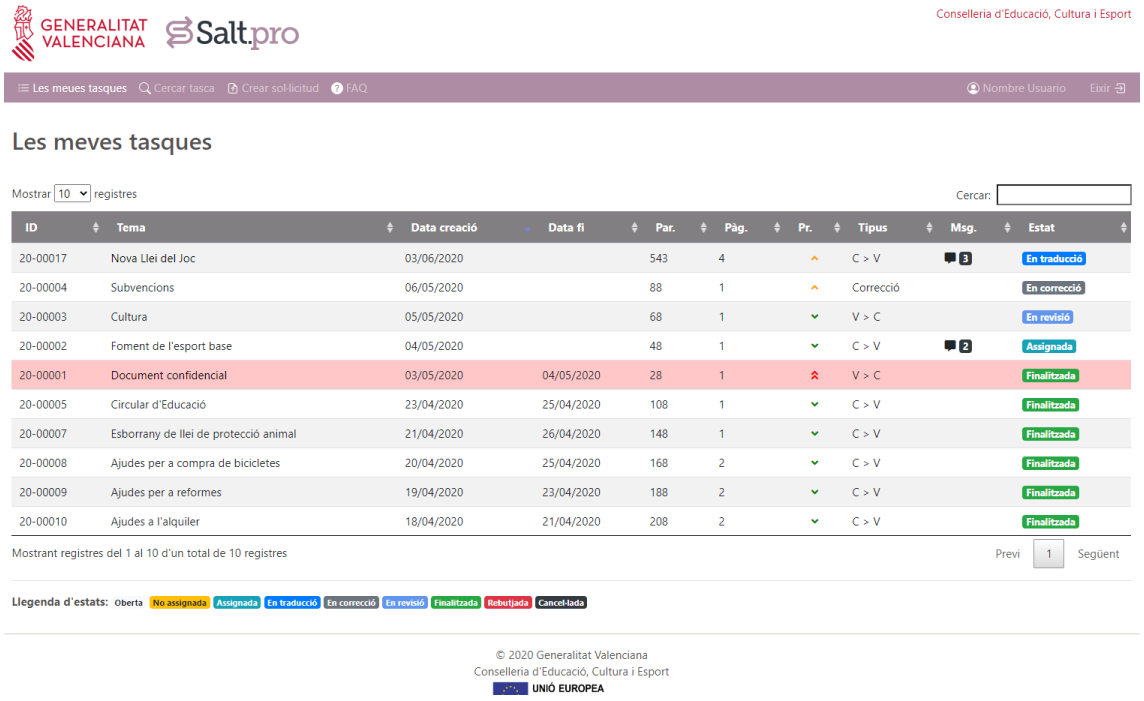

Ilustración 5.2: Prototipado de la pantalla "Les meues tasques"

Siguiendo con otra pantalla importante en el rol de Usuario, se expone a continuación la pantalla del formulario de creación de tarea.

Ilustración 5.3: Prototipado de la pantalla "Crear sol·licitud

A continuación, observamos el prototipado inicial para dos pantallas de interés para el rol de técnico, la de Tareas no Asignadas y cuando una tarea propia del técnico se encuentra en estado "En traducción".

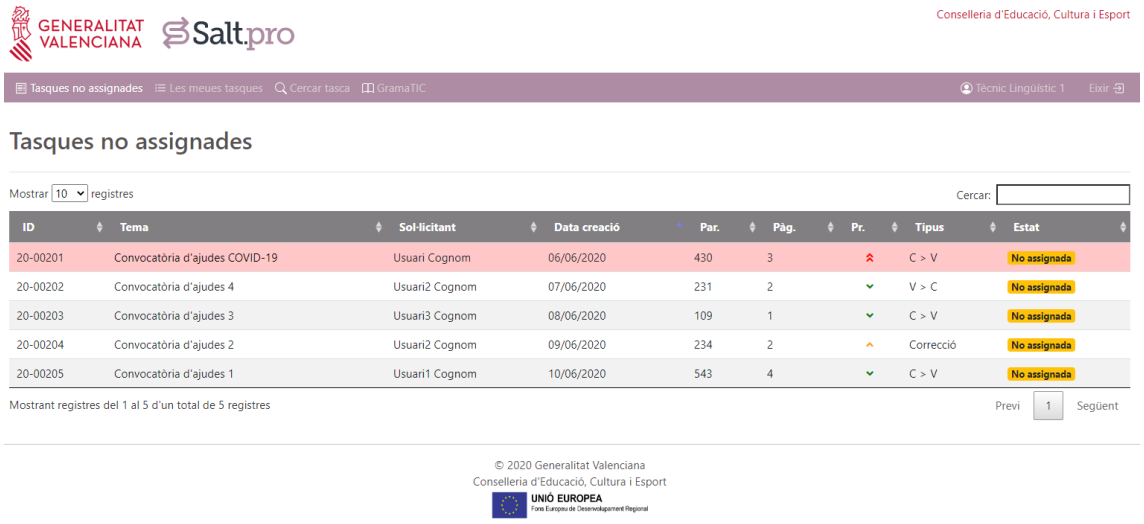

Ilustración 5.4: Prototipado de la pantalla "Tasques no assignades"

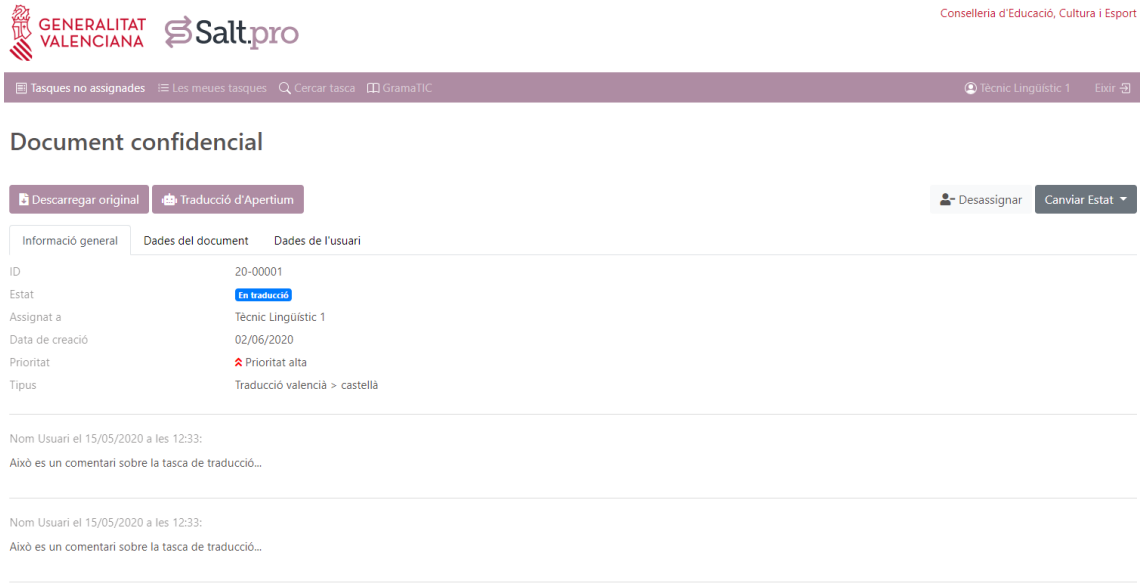

Ilustración 5.5: Prototipado de la pantalla de información de una tarea

Para el rol de administrador se han seleccionado cuatro pantallas de especial interés para el cliente y a continuación se muestra el prototipado inicial de dichas pantallas.

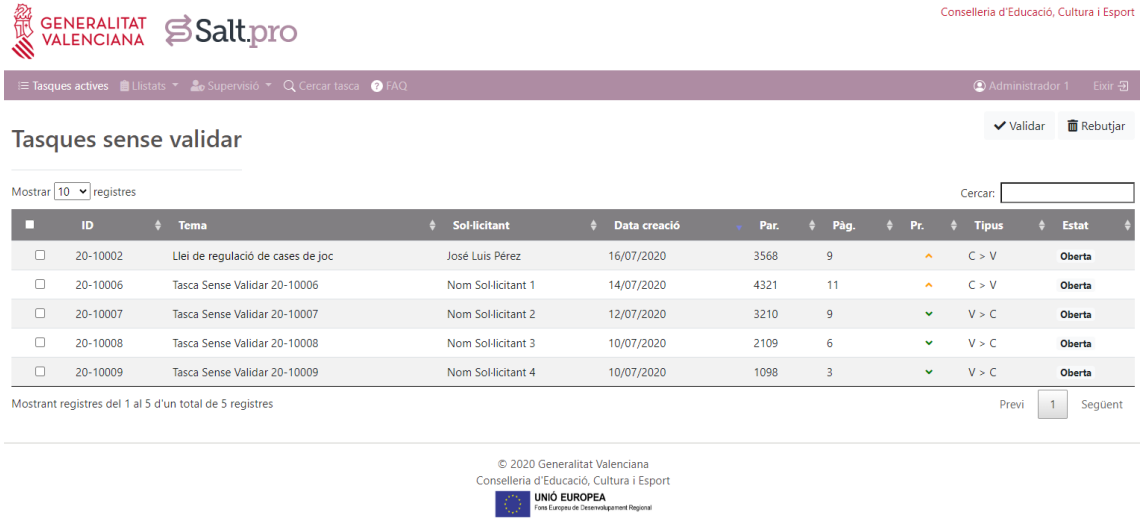

Ilustración 5.6: Prototipado de la pantalla "Tasques sense validar"

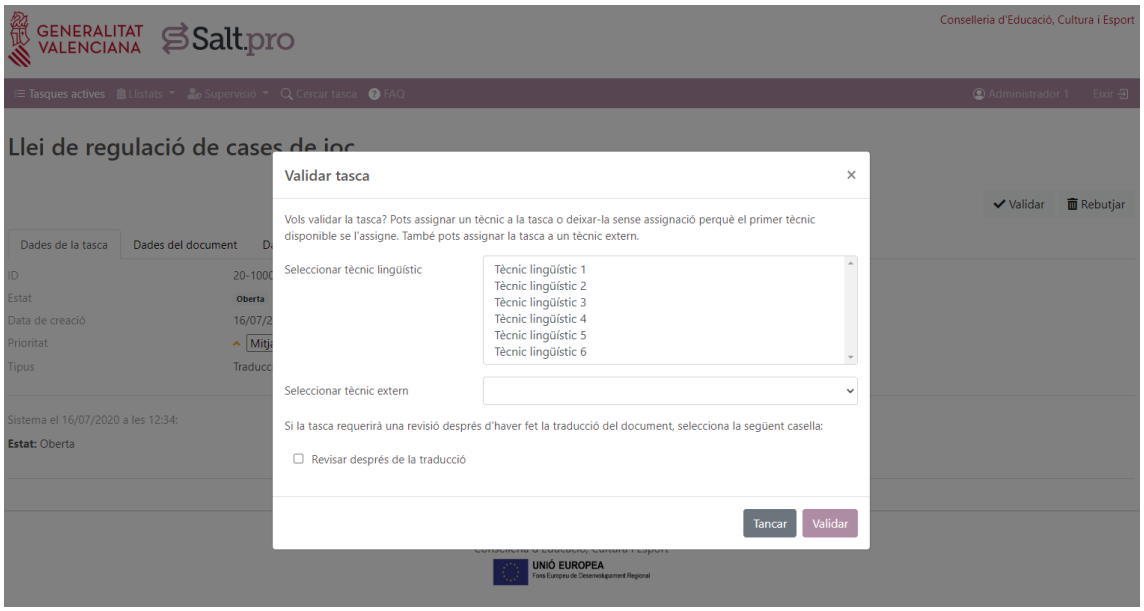

Ilustración 5.7: Prototipado de la ventana modal "Validar tasca"

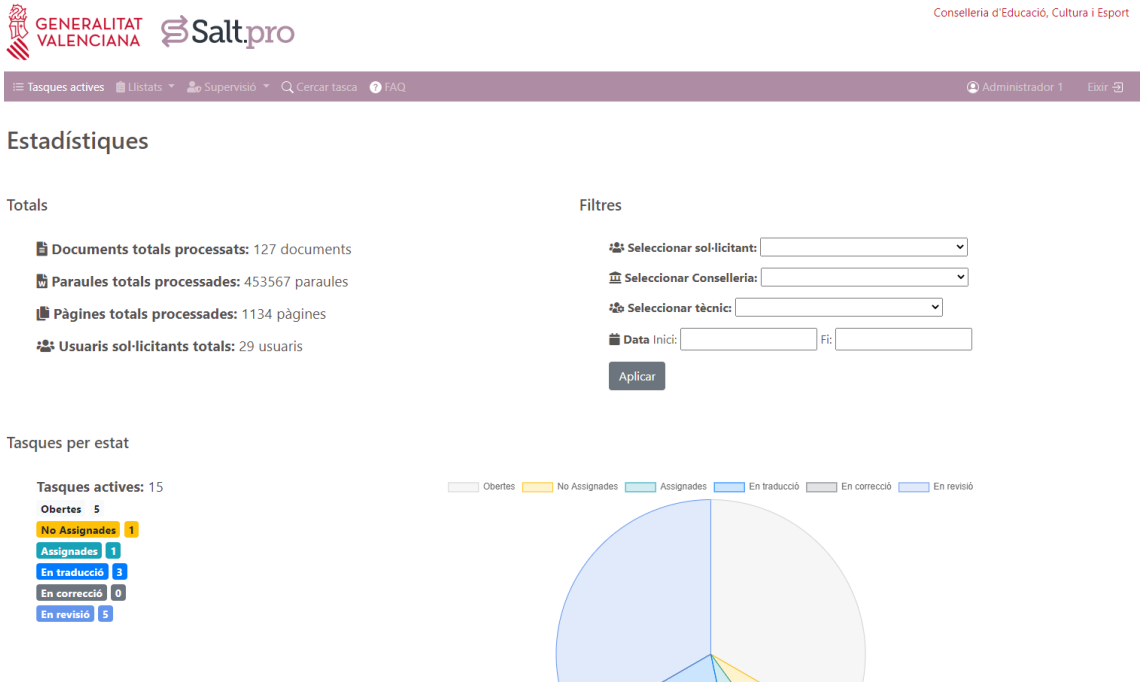

Ilustración 5.8: Prototipado de la pantalla "Estadístiques"

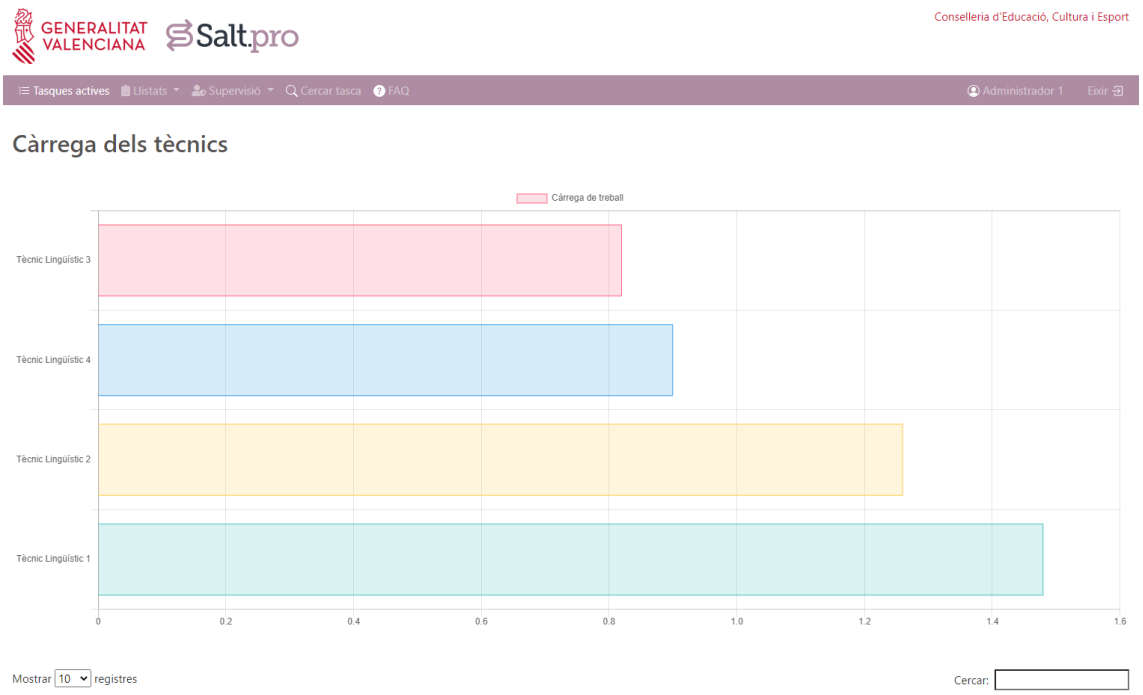

Ilustración 5.9: Prototipado de la pantalla "Càrrega dels tècnics"

En apartados posteriores se verá la pequeña evolución del diseño de la aplicación en comparación con estos prototipados iniciales. Cabe destacar que se han mantenido los aspectos más importantes y característicos de la aplicación.

6. Desarrollo de la solución

En este capítulo de la memoria se detallan las etapas de desarrollo de la aplicación, así como las soluciones que se han aplicado para resolver cualquier incidencia.

6.1 Arquitectura

Uno de los principios de diseño más útiles es hacer y mantener una buena separación de los diferentes bloques funcionales de una aplicación y una de las maneras más comunes de llevar este principio a buena práctica es dividir la aplicación en tres capas: presentación, lógica de negocio y acceso a datos.

- **Presentación**. Encargada de manejar o gestionar las peticiones y generar el mensaje de respuesta.
- **Negocio**. Todo lo relacionado con la lógica del negocio.

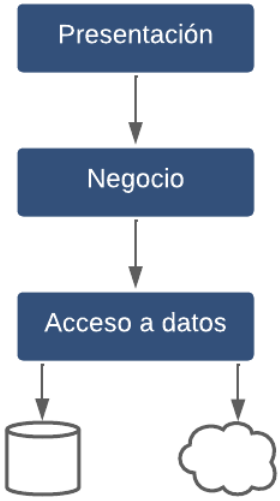

Ilustración 6.1: Arquitectura 3-capas

- **Acceso a datos**. Responsable de la persistencia de datos en base de datos.

Este tipo de arquitectura ha demostrado ser una forma efectiva para organizar las aplicaciones en módulos o capas. Sus principales beneficios son:

- Focaliza el trabajo de desarrollo, ya que permite centrar la atención en cada capa funcional por separado.
- Independencia entre módulos, permite substituir diferentes implementaciones de una misma capa sin afectar al resto de capas ni duplicar la lógica.
- Mayor capacidad para implementar pruebas de código gracias a que las capas definen su frontera o interfaz de forma clara.

Estos beneficios son críticos para la productividad en cualquier proyecto, motivo por el que habitualmente se buscan variantes que aumenten estos beneficios para conseguir una mayor productividad. Las variantes más comunes son subdividir la capa de presentación y la capa de negocio e implementar la capa de acceso a datos utilizando patrones de diseño como *Repository*:

- **Presentación**
  - Capa **Vista**: generación del mensaje resultado de la petición.
  - Capa **Control**: gestión de peticiones HTTP.
- **Negocio**
  - Capa **Servicios**: flujos de trabajo y procesos del negocio.
  - Capa **Modelo**: lógica y estructuras de datos del dominio.
- **Acceso a datos**
  - Capa **Repositorio**: Código de construcción de consultas.
  - Capa **Conversión de datos**: Aísla los objetos del dominio de los detalles del código de acceso a la base de datos.

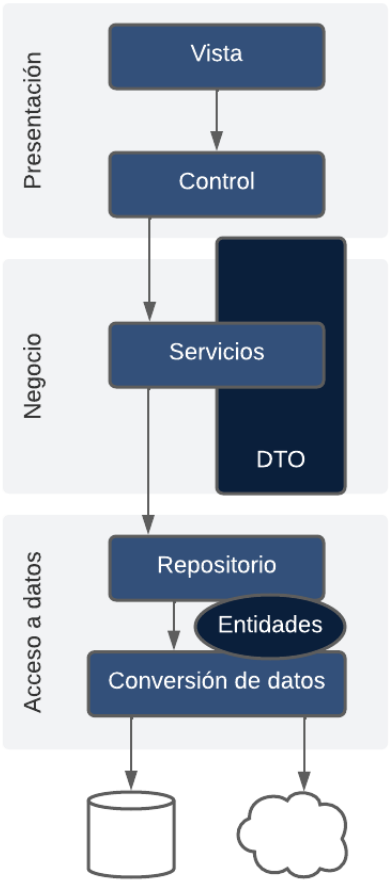

Ilustración 6.2: Arquitectura n-capas

Como se observa en el diagrama anterior, además de la división en capas, el modelo de arquitectura propone una dependencia entre capas que especifica que la ejecución va desde la capa superior hasta la capa más inferior pasando por todas de la pila: la

vista depende de la capa de control, que depende de la capa de servicios, que depende de la capa repositorio y que depende de la capa de conversión de datos.

El modelo es una capa transversal. El modelo en ningún caso depende de la capa, es decir, aunque la lógica del dominio puede invocarse por la capa de servicio, esta no podrá invocar elementos de la capa repositorio, sólo podrá delegar en servicios de la propia capa de negocio.

En adición a los beneficios indicados anteriormente para una arquitectura de 3 capas, en una arquitectura de n capas podemos encontrar algunos otros beneficios:

- Permite utilizar distintas tecnologías de vista sin necesidad de duplicar código de la capa de control.
- Mayor facilidad para desarrollar las pruebas de la capa de control ya que define su interfaz.
- Participación de perfiles más especializados, por ejemplo, al separar la vista del control perfiles de diseño pueden participar en el desarrollo de la vista de los proyectos y no necesitan conocimientos de programación.

Además de esta división funcional la aplicación de Salt.pro cuenta con un conjunto de elementos adicionales como:

- Seguridad: componentes que permiten proteger la aplicación. En este caso destacan la segmentación funcional por tipo de usuario y el sistema de control de acceso. Estos componentes sirven para evitar el uso indebido por parte de los usuarios que ponga en riesgo la información que gestiona la aplicación.
- Integración: elementos que permiten a las aplicaciones interactuar con otras aplicaciones como por ejemplo Apertium o GvLogin.
- Inyección de dependencias.

6.1.1 Framework base

El proyecto de Salt.pro tiene una organización modular basada en el sistema multimódulo de Maven que hace más fácil el mantenimiento del proyecto.

Se utiliza una organización basada en las capas de la arquitectura de la aplicación web distribuida en 7 módulos:

- Módulo *raíz (fuentes)*: agregador del resto de módulos.
- Módulo *application*: framework base y capa de presentación.
- Módulo *dto*: capa de DTOs del modelo del dominio.
- Módulo *model*: capa de entidades del modelo del dominio.
- Módulo *repository*: capa de acceso a datos.
- Módulo *service-api*: API o interfaz de los servicios de la capa de servicios.
- Módulo *service-impl*: implementación de los servicios de la capa de servicios.

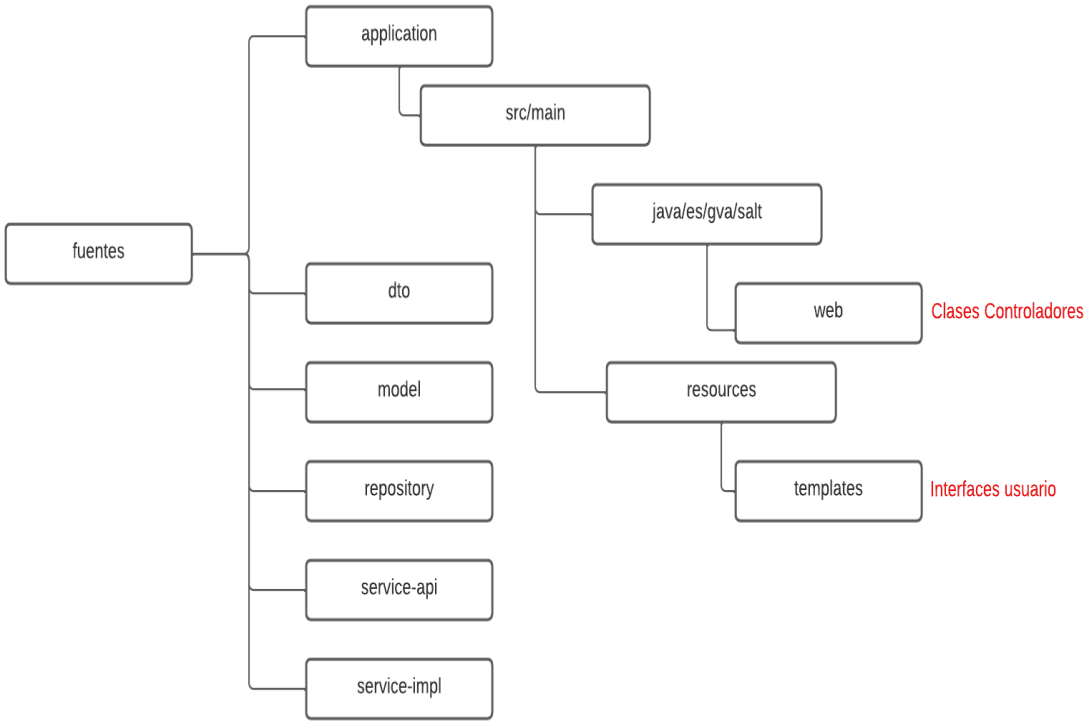

Ilustración 6.3: Diagrama de la organización en módulos de la aplicación

El proyecto desarrollado es identificado por las coordenadas GAV (groupId, artifactId, version), las cuales permiten identificar un artefacto software de forma única.

Las coordenadas GAV de un proyecto se definen en el archivo pom.xml de cada módulo Maven, por ejemplo:

```
<!-- tag:Datos Maven -->
<groupId>es.gva.salt</groupId>
<artifactId>salt-pro</artifactId>
<version>01.00.00-SNAPSHOT</version>
<!-- end:Datos Maven -->
```

Ilustración 6.4: Coordenadas GAV en Maven

Además, los proyectos de la GVA de la oficina java tienen configurado un Maven común denominado super pom que debe ser el pom padre de todos los proyectos que cumplen el estándar. Contiene ciertas configuraciones comunes para todos los proyectos del estándar para evitar repetir las mismas configuraciones en cada proyecto.

```xml
<!-- tag:Import Super POM -->
<parent>
  <groupId>es.gva.dgtic</groupId>
  <artifactId>super-pom</artifactId>
  <version>03.01.01</version>
</parent>
<!-- end:Import Super POM -->
```

Ilustración 6.5: Inclusión del super pom en el pom.xml del módulo raíz

Este super pom incluye mediante herencia el super pom de Spring Boot.

Las aplicaciones basadas en Spring Boot incluyen una clase principal. Esta clase es muy importante para Spring Boot ya que la utiliza como punto de entrada para ejecutar toda la funcionalidad de autoconfiguración de Spring Boot.

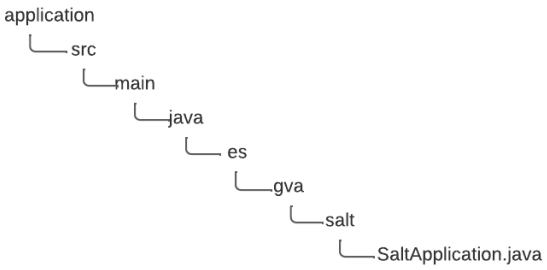

Ilustración 6.6: Ubicación clase principal de Salt.pro

6.1.2 Acceso a datos

En esta capa se incluyen tres de los módulos de mencionados anteriormente.

- Módulo *model*

Este módulo contiene todas las entidades de la capa del modelo.

- Módulo repository

Módulo que contiene los elementos de la capa de acceso a datos, es decir, la capa de repositorio de entidades y la capa de conversión de datos.

- Módulo *dto*

Este módulo contiene todos los objetos de transferencia de datos.

6.1.3 Negocio

Esta capa contiene los módulos relacionados con los servicios.

- Módulo *service-api*

Este módulo contiene el *API* de la capa de servicios, es decir, sólo contiene interfaces de servicios y clases de la capa de servicios que pueden utilizarse en la capa de control.

- Módulo *service-impl*

Módulo donde se ubica la implementación de los servicios de la capa de negocio.

La lógica de la aplicación se implementa en servicios de la aplicación y el conjunto de estos servicios constituyen la capa de servicios dentro de la capa de negocio.

Se establecen los siguientes principios de dependencia entre capas:

1. El modelo (entidades y DTOs) en ningún caso utiliza ninguna otra capa, las entidades del modelo sólo interactúan con otras entidades del dominio o con ciertos tipos de DTOs como pueden ser tipos enumerados y los DTOs solo interactúan con otros tipos de DTOs.
2. Los servicios invocan o delegan en las interfaces de otros servicios propios de esta capa, en las interfaces de los repositorios de la capa de acceso a datos y por supuesto, en las entidades y *DTOs* del modelo del dominio.

6.1.4 Control

En esta capa podemos encontrar el módulo principal de la aplicación, módulo *application.* La capa de control contiene los componentes que gestionan la entrada y tratamiento de las peticiones *HTTP*. El *API HTTP* o interfaz *RESTful* de la aplicación esta formada por diferentes elementos entre los que destacan los **recursos** y las **acciones**.

Un **recurso** es cualquier elemento del dominio o de la lógica de la aplicación a la que se le pueda asignar un nombre que lo identifique: un archivo, una entidad, una operación de negocio, etc.

Una vez se han identificado los recursos, se modelan las operaciones o **acciones** necesarias sobre cada recurso utilizando los métodos *HTTP* disponibles:

- **GET**: Obtiene información de un recurso.
- **POST**: Crea nueva información en un recurso.
- **PUT**: Actualiza información de un recurso.
- **DELETE**: Elimina información de un recurso.

Para la creación de la capa de control se ha hecho uso del patrón MVC, que se comentará posteriormente, con Spring MVC.

Spring Boot proporciona un *starter* con nombre spring-boot-starter-web que integra Spring MVC en un proyecto Spring incluyendo la dependencia en el archivo pom.xml del módulo *application*.

Un criterio que facilita las labores de mantenimiento y resolución de incidencias es agrupar todos los métodos manejadores de acciones relativas a un recurso en un único

controlador. Por ejemplo, los métodos que llevan a cabo acciones sobre el recurso “Tarea” se agrupan en un único controlador denominado “TareaController”. En un apartado posterior se exponen ejemplos de dicho recurso y su controlador.

Los controladores se crean en el módulo *application* dentro del sub paquete *web*.

6.1.5 Vista

- Vista Servidor

La capa de vista a nivel de servidor contempla todos los elementos relacionados con la generación de la vista de usuario realizada por la aplicación en funcionamiento en el servidor. Un sistema de plantillas permite separar los datos de la aplicación de su visualización al usuario final. Las plantillas se ubican dentro del módulo *application* en la carpeta */src/main/resources/templates*.

- Vista Cliente

La capa de vista a nivel de cliente recoge aquellos elementos que participan en la creación de la vista y que se ejecutan directamente sobre el cliente web.

6.2 Contexto tecnológico

En este apartado vamos a presentar y explicar brevemente las tecnologías utilizadas para el desarrollo de esta aplicación. Para hacer la lectura más intuitiva se seguirá una orden comenzando por el usuario y la interfaz hasta llegar a la base de datos.

Aunque en este apartado explicaremos las tecnologías que se han utilizado, la aportación de código de ejemplo no se realizará hasta el apartado 6.4 Ejemplos de código de este documento.

A continuación, se muestra un pequeño diagrama de las tecnologías utilizadas y se muestra el orden en el que se explicarán.

Ilustración 6.7: Diagrama de tecnologías

Bootstrap [2]: Es una de las librerías *frontend* más utilizadas y permite diseñar un *frontend responsive* accesible desde cualquier navegador y dispositivo, utilizando para ello HTML, CSS y Javascript. El principal objetivo es desarrollar una plataforma más intuitiva y sencilla de utilizar. Además, el uso de una librería de *frontend* permite reducir el tiempo de desarrollo al poder reutilizar todos los componentes que ya proporciona.

- **HTML**: Lenguaje de marcado cuyo nombre significa Lenguaje de Marcas de Hipertexto. Es el componente más básico de la Web. «Define el significado y la estructura del contenido Web». Estructuralmente, HTML se basa en un sistema de elementos y estilos definidos por marcas como '<*head*>', '<*body*>' o '<*title*>'.
- **CSS**: Lenguaje de estilos cuyo nombre significa Hojas de Estilo en Cascada. Se utiliza en combinación con HTML para definir «como debe ser renderizado el elemento estructurado en la pantalla, en papel, en el habla o en otros medios» [3].
- **Javascript**: Lenguaje de programación que destaca por su compilación durante la ejecución, por ser «basado en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa» [4].

Thymeleaf [5]: Para la implementación de la capa vista-servidor se utiliza, siguiendo las recomendaciones de la Oficina Java, Thymeleaf. Se trata de una librería Java que implementa un motor de plantillas para procesar y crear HTML, XML, Javascript, CSS y texto. Puede ser utilizado tanto en entornos web como en entornos ajenos a la web. Spring permite la integración completa del motor de plantillas Thymeleaf que añade atributos al lenguaje HTML para facilitar algunas tareas al programador.

Spring Boot [6][7]: La elección de Java como lenguaje de programación para el desarrollo de la aplicación no es casual, pues se trata de uno de los lenguajes de referencia de diversos proyectos de la Generalitat Valenciana. Como asistente de desarrollo Java se utiliza Spring Boot, que ya se emplea en múltiples proyectos de la GVA.

El objetivo de Spring Boot es proporcionar un conjunto de herramientas para construir rápidamente aplicaciones de Spring que sean fáciles de configurar. Esa aceleración se basa en anteponer una serie de convenciones que predominan sobre la configuración. Es decir, Spring Boot establece algunas configuraciones por defecto. Su objetivo no es sustituir a Spring pues trabaja en base a Spring y por eso Spring Boot no se considera un *framework* tal y como lo es Spring.

El problema principal de una aplicación basada en Spring consiste en que lleva mucho trabaja configurarlo. Spring es un conjunto de infraestructuras que, para funcionar correctamente, requieren una configuración muy cuidadosa. La solución a ese problema es Spring Boot que toma una visión dogmática de la plataforma de Spring y de las bibliotecas de terceros para que pueda comenzarse con el mínimo revuelo. La poca configuración que se necesita está en forma de anotaciones.

La Oficina Java publica multitud de material y manuales para que los desarrollos para la GVA basados en Java cumplan con los estándares establecidos por la DGTIC, esto permite, entre otros:

- Utilizar el super pom de la GVA, unificando las versiones de los componentes a utilizar y asegurando que su compatibilidad esté comprobada.
- Emplear una base de proyecto generada con Spring Boot.
- Modularizar el desarrollo de la aplicación.
- Facilitar la integración continua con Jenkins para agilizar su despliegue en los entornos de desarrollo, preproducción y producción.

Jenkins [8] es un servidor *open source* para la integración continua. Es una herramienta que se utiliza para compilar y probar proyectos de software de manera continua, lo que facilita integrar cambios en un proyecto y entregar nuevas versiones. Mediante Jenkins conseguimos acelerar el proceso de desarrollo y entrega de software a través de la automatización.

Spring [6]: La historia de Spring comienza en 2003 de la mano de Rod Johnson quien fue su principal desarrollador. Su objetivo fue crear una plataforma para desarrollar en Java de código abierto. Su uso comenzó a extenderse hasta convertirse en el principal *framework* para Java en el ámbito empresarial.

Al comienzo, la ejecución de proyectos Spring seguía un flujo simple y secuencial. Cuando aparecieron bibliotecas de código reutilizable apareció el problema del acoplamiento entre componentes. Es entonces cuando Spring incorpora la inyección de dependencias. La ventaja principal es que el propio *framework* proporciona objetos a una clase, en lugar de ser la propia clase quien cree esos objetos. Eso es debido a que Spring posee un contenedor donde se gestionan las instancias de los objetos de la aplicación y su respectivo ciclo de vida.

Spring Security: Es otro módulo de Spring que se ha utilizado para gestionar la seguridad de la aplicación. Mediante los mecanismos que proporciona se protege la aplicación de una manera sencilla y sin afectar a la lógica de negocio. Los mecanismos principales que proporciona son:

- Protocolos de seguridad
- Roles para los accesos a recursos
- Autenticación y autorización

Hibernate [9]: Se trata de una herramienta de mapeo objeto-relacional (ORM). Su objetivo principal es despreocupar al programador de establecer cómo se guarda, recupera o elimina la información que persiste. Hibernate permite acceder a los datos y mapear las entidades con los objetos, así como también facilita consultas y operaciones sobre la base de datos. Cuenta con integración completa con Spring.

PostgreSQL [10]: Es un sistema gestor de bases de datos relacionales libre y de código abierto. Es considerado el motor de base de datos más avanzado hoy en día.

Para la administración y el desarrollo de la base de datos se ha utilizado DBeaver, un IDE de administración que aporta un gran número de funcionalidades.

Para el desarrollo de la aplicación se ha utilizado principalmente, uno de los entornos de desarrollo más usados actualmente, IntelliJ.

6.3 Patrones de diseño

En este apartado se explicará detalladamente el patrón de arquitectura de software MVC que emplea la aplicación Salt.pro, cómo funciona, así como sus componentes.

La función principal del patrón MVC [11] es la de separar la lógica de negocio de una aplicación, es decir los datos, de su representación y el módulo que gestiona los eventos y las comunicaciones.

Se lleva a cabo dicha separación mediante tres componentes distintos que son el modelo, la vista y el controlador.

La siguiente ilustración es una demostración de las relaciones entre estos tres componentes y el cliente o usuario del sistema:

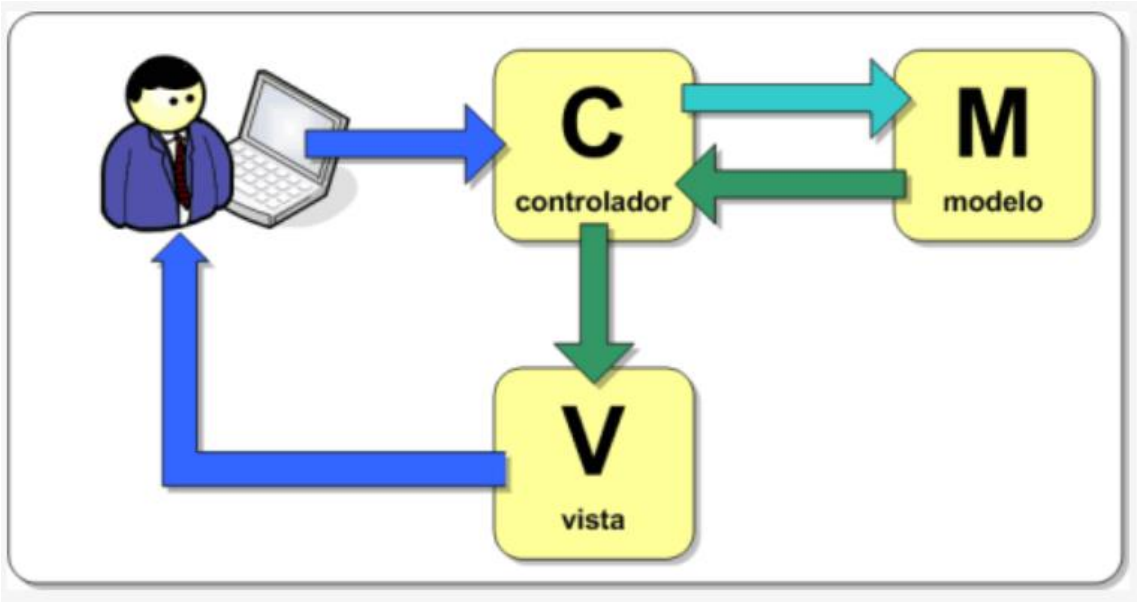

Ilustración 6.8: Diagrama patrón MVC. Fuente: campusmvp

Como se demuestra en el diagrama, las peticiones o la información que proviene directamente del usuario las recoge el controlador de modo que ningún otro componente tendrá acceso a los datos provenientes del usuario. Así como la vista es el único componente encargado de generar interfaces de usuario para transmitir la información de retorno. Destaca en este patrón el componente de controlador especialmente porque es tanto el encargado de enviar datos a la vista para más adelante transmitirlos al usuario como por su acceso bidireccional al modelo, eso significa que puede actualizar su estado mediante la invocación de métodos que se encuentran en su lógica de negocio como recibir información para completar sus tareas. El modelo se considera la capa de la representación de los datos, la lógica del sistema y los mecanismos de persistencia. Es recomendable que esta capa sea independiente al sistema de almacenamiento. Spring diferencia entre modelos y entidades. El modelo engloba los objetos que se envían y se reciben del controlador mientras que las entidades son las que utiliza Hibernate para llevar a cabo consultas en la base de datos. Para trabajar con ambos se hace uso de los conversores que se dedican a la transformación entre modelo y entidades. Estos conversores son clases de transferencia de datos o DTO (*Data Transfer Objects*).

6.4 Ejemplos de código

Para finalizar este apartado, se mostrarán algunos ejemplos de código de interés de la aplicación Salt.pro. Se dará una breve explicación de cada parte comenzando, como se ha hecho hasta ahora por la interfaz de usuario.

6.4.1 Nueva tarea - plantilla

Para comenzar se muestra un ejemplo de código en el lenguaje que define los elementos a mostrar en la página web: HTML.

Las ilustraciones 6.5 y 6.6 del fichero *nova-tasca.html* muestran un ejemplo de cabecera y pie de la página donde importamos distintos elementos de los que se hace uso en la página web, por ejemplo: Bootstrap, jQuery o Select2. También importamos ficheros javascript y css para cargar estilos y funciones que se utilizan. Todo lo incluido en la cabecera se cargará antes de la propia página como pueden ser por ejemplo los estilos "*theme.css*". Lo que se incluye en el pie de la página son principalmente ficheros javascript de los que se hará uso después de que la página se haya cargado por completo.

No todas las funciones javascript que se utilizan en la página se encuentran en ficheros separados, algunos se incluyen en el propio fichero *nova-tasca.html* como se mostrará a continuación.

```
<head>
  <meta charset="UTF-8" />
  <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />
  <title data-layout-title-pattern="$CONTENT_TITLE - $LAYOUT_TITLE"
   data-th-text="#{label_aplicacion_titulo}"></title>

  <link rel="stylesheet" type="text/css"
   data-th-href="@{/webjars/bootstrap/4.5.0/dist/css/bootstrap.css}" />
  <link rel="stylesheet" type="text/css"
   data-th-href="@{/webjars/datatables/1.10.19/media/css/jquery.dataTables.css}" />
  <link rel="stylesheet" type="text/css"
   data-th-href="@{/webjars/select2/4.0.7/dist/css/select2.css}" />
  <link rel="stylesheet" type="text/css"
   data-th-href="@{/webjars/datetimepicker/2.5.20/build/jquery.datetimepicker.min.css}" />
  <link rel="stylesheet" type="text/css"
   data-th-href="@{/public/css/theme.css}">
   <link rel="stylesheet" type="text/css"
   data-th-href="@{/public/css/quill.min.css}">
  <link rel="stylesheet" type="text/css"
   data-th-href="@{/public/libs/fontawesome/css/all.min.css}">
</head>
```

Ilustración 6.9: Código - Cabecera HTML

```
<!-- Código JavaScript de la página para agilizar la carga -->
<script type="text/javascript"
 data-th-src="@{/webjars/jquery/3.5.1/dist/jquery.min.js}">
</script>
<script type="text/javascript"
 data-th-src="@{/webjars/popper.js/1.16.0/dist/umd/popper.min.js}">
</script>
<script type="text/javascript"
 data-th-src="@{/webjars/bootstrap/4.5.0/dist/js/bootstrap.min.js}">
</script>
<script type="text/javascript" charset="utf-8"
 data-th-src="@{/webjars/datatables/1.10.19/media/js/jquery.dataTables.js}">
</script>
<script type="text/javascript" charset="utf-8"
 data-th-src="@{/webjars/datatables.net-select/js/dataTables.select.js}">
</script>
<script type="text/javascript"
 data-th-src="@{/public/js/moment.min.js}">
</script>
<script type="text/javascript"
 data-th-src="@{/public/js/datetime-moment.js}">
</script>
</script>
<script type="text/javascript"
 data-th-src="@{/webjars/select2/4.0.7/dist/js/select2.full.js}">
</script>
<script type="text/javascript"
 data-th-src="@{/webjars/datetimepicker/2.5.20/build/jquery.datetimepicker.full.min.js}">
</script>
```

Ilustración 6.10: Código – Pie HTML

A continuación, se muestra el código de algunos elementos de la página. En concreto veremos el elemento para subir un fichero y por introducir el tema de una tarea que forman parte del formulario de creación de nuevas tareas.

```
<!-- Control para seleccionar el archivo a subir -->
<div class="form-group row">
  <label class="col-sm-3 col-form-label">
    <span data-th-text="#{label_nueva_tarea_archivo}"></span>
    <span class="fas fa-info-circle"
     data-th-title="#{label_nueva_tarea_archivo_ayuda}"></span>
  </label>
  <div class="col-sm-4">
    <input type="file" name="archivo" id="archivo"
     th:max-size="#{label_tamano_maximo_archivo}"/>
  </div>
  <div class="col-sm-3">
    <span class="error" id="error_archivo"></span>
  </div>
</div>

<!-- Control para introducir el tema de la tarea -->
<div class="form-group row">
  <label class="col-sm-3 col-form-label">
    <span data-th-text="#{label_nueva_tarea_tema}"></span>
    <span class="fas fa-info-circle"
     data-th-title="#{label_nueva_tarea_tema_ayuda}"></span>
  </label>
  <div class="col-sm-7">
    <input type="text" class="form-control" id="tema" name="tema"
     data-th-field="*{tema}" />
  </div>
  <div class="col-sm-2">
    <span class="error" id="error_tema"></span>
  </div>
</div>
```

Ilustración 6.11: Código – elementos HTML

Aquí se puede observar el uso de Thymeleaf, el motor de plantillas comentado anteriormente en el apartado de las tecnologías, como forma de acceder a las variables que se le proporciona a la plantilla como entrada como, por ejemplo:

```
data-th-title="#{label_nueva_tarea_tema_ayuda}"
```

Aquí recoge el valor de la variable "*label_nueva_tarea_tema_ayuda*" y lo muestra mediante un <*span*>.

Bootstrap permite utilizar algunos estilos predefinidos en sus elementos, de todas formas, en la aplicación se han personalizado algunos estilos sobrescribiendo a los estilos de Bootstrap o se han definido nuevos estilos. A continuación, se muestran algunos estilos como por ejemplo los que se le han aplicado al logo de la aplicación de Salt.pro.

```css
body {
    color: #565353;
    font-size: 0.95rem;
}

/* Logotipo de la aplicacion Salt */
.logo-salt{
  background-image: url('../img/logo.pro.svg');
  background-position: left center;
  background-repeat: no-repeat;
  background-size: auto 45px;
  display: inline-block;
  height: 45px;
  width: 280px;
}

#seleccionar-rol{
    margin-left: 10px;
    margin-top: 5px;
}
```

Ilustración 6.12: Código - ejemplos CSS

Utilizando CSS se pueden distinguir 3 maneras de distinguir entre diferentes atributos:

El primer bloque que se muestra hace referencia a los elementos del HTML como puede ser en este caso el "<*body*>".

El segundo bloque hace uso del punto *"."* y afecta a todos los elementos que incluye esa clase en este caso "*class=logo-salt*".

Y, por último, el bloque que hace uso de la almohadilla *"#"* pues con ello aplica los estilos a los elementos cuyo identificador sea "*id=seleccionar-rol*".

6.4.2 TareaController

En primer lugar, se muestra cómo se inyecta un servicio del que hará uso el controlador para realizar distintas acciones. El servicio tiene un rol intermedio entre el controlador que se comunica con la vista y el repositorio que se comunica directamente con la base de datos. Para desacoplar el código como se ha comentado anteriormente, el controlador hará uso de los métodos de la interfaz del servicio para que si luego hay cambios en la implementación de dicho servicio esto no provoque inconvenientes en la implementación del controlador. La inyección de dicha dependencia se hace mediante la anotación @Autowired de Spring Boot.

```java
/** API del servicio para la entidad {@link Tarea} */
@Autowired
private TareaService tareaService;

/** API del servicio para la entidad {@link TipoTarea} */
@Autowired
private TipoTareaService tipoTareaService;

/** API del servicio para la entidad {@link Documento} */
@Autowired
private DocumentoService documentoService;

/** API del servicio para la entidad {@link HistoricoTarea} */
@Autowired
private HistoricoTareaService historicoTareaService;

/** API del servicio para la entidad {@link Estado} */
@Autowired
private EstadoService estadoService;

/** API del servicio para la entidad {@link Usuario} */
@Autowired
private UsuarioService usuarioService;

/** API del servicio para la entidad {@link Asignacion} */
@Autowired
private AsignacionService asignacionService;
```

Ilustración 6.13: Código - Inyección de dependencias

En la Ilustración 6.10 se muestra un método del controlador que se encarga de obtener una tarea a visualizar. Veremos que se hace uso de la clase *ModelAndView* [14] para poder pasar el objeto a visualizar (la Tarea) a la vista. Esta clase da soporte tanto para Modelo como para Vista en el patrón MVC, aunque hay que tener en cuenta que son componentes completamente distintos. Esta clase contiene ambos simplemente para hacer posible que un controlador devuelva tanto el modelo como la vista en un único valor de retorno. También se puede observar que este método lo que devuelve es la entidad Tarea.

Una entidad de persistencia es una clase de Java ligera, cuyo estado es persistido de manera asociada a una tabla en una base de datos relacional.

```java
/**
 * Obtener Tarea por codtarea
 *
 * @param tarea Tarea a visualizar
 * @return Entidad {@link Tarea}
 */
@GetMapping(value = "/{codtarea}")
public ModelAndView mostrar(@ModelAttribute Tarea tarea) {

  Tarea result = tareaService.findTareaByCodtarea(tarea.getCodtarea());

  //Lista de mensajes a mostrar en la pestaña de información de la tarea
  List<Mensaje> mensajes =
      mensajeService.listMensajesByCodtarea(tarea.getCodtarea());

  List<Mensaje> msgSistema =
      mensajeService.listCambiosEstadoByCodtarea(tarea.getCodtarea());

  //Comprobamos si la tarea está asignada
  int asignaciones =
    asignacionService.findAsignacionByCodtarea(result.getCodtarea()).size();
  boolean asignacionConfig = asignaciones > 1;

  //Obtenemos la fecha de inicio de la traducción, en caso de haberse iniciado
  Date fechaIniTrad =
      historicoTareaService.getFechaIniTrad(result.getCodtarea());

  ModelAndView modelo = new ModelAndView(VISTA_TAREA);
  modelo.addObject(
    "usuario", this.loggedUser.getLoggedUser().getDisplayName()
  );
  modelo.addObject("rol", this.loggedUser.getLoggedUser().getRol().getRol());
  modelo.addObject("asignacionConfig", asignacionConfig);
  modelo.addObject(VISTA_TAREA, result);
  modelo.addObject("mensajes", mensajes);
  modelo.addObject("msgSistema", msgSistema);
  modelo.addObject("msgSize", mensajes.size());
  modelo.addObject(
    "msgUnread",
    mensajeService.mensajesNoLeidos(
      tarea.getCodtarea(), this.loggedUser.getLoggedUser().getNif()
    )
  );
  modelo.addObject("fechaIniTrad", fechaIniTrad);
  modelo.addObject(
    "nombreCompleto",
    result.getSolicitante().getNombreCompleto()
  );
  if(result.getTradAsignado() != null) {
    modelo.addObject(
      "nombreTrad",
      result.getTradAsignado().getNombreCompleto()
    );
  }
  if(result.getRevAsignado() != null) {
    modelo.addObject(
      "nombreRev",
      result.getRevAsignado().getNombreCompleto()
    );
  }
  modelo.addObject(
    "estaCaducada", tareaService.isCaducada(result.getCodtarea())
  );

  return modelo;
}
```

Ilustración 6.14: Código - método de controlador

Nuestras entidades JPA son mapeadas acorde al esquema de la base de datos. En la siguiente ilustración se puede observar una parte del código de esa entidad.

```java
@Entity
@Table(name = "salt_tarea")
public class Tarea implements Serializable {

  private static final long serialVersionUID = -1760761312362124421;

  @Id
  @Column(name = "codtarea")
  @Size(max = 10)
  @NotNull
  private String codtarea;

  @Column(name = "tema")
  @NotNull
  private String tema;

  @OneToOne(fetch = FetchType.EAGER, targetEntity = Documento.class)
  @JoinColumn(name = "coddoc_ori", referencedColumnName = "coddoc")
  private Documento docOri;

  @OneToOne(fetch = FetchType.EAGER, targetEntity = Documento.class)
  @JoinColumn(name = "coddoc_trad", referencedColumnName = "coddoc")
  private Documento docTrad;

  @OneToOne(fetch = FetchType.EAGER, targetEntity = Documento.class)
  @JoinColumn(name = "coddoc_rev", referencedColumnName = "coddoc")
  private Documento docRev;

  @ManyToOne(fetch = FetchType.EAGER, targetEntity = TipoTarea.class)
  @JoinColumn(name = "codtipo_tarea", referencedColumnName = "codtipotarea")
  @NotNull
  private TipoTarea tipoTarea;

  @ManyToOne(fetch = FetchType.EAGER, targetEntity = Estado.class)
  @JoinColumn(name= "codestado", referencedColumnName = "codestado")
  @NotNull
  private Estado estado;

  @ManyToOne(fetch = FetchType.EAGER, targetEntity = Usuario.class)
  @JoinColumn(name = "nif_solicitante", referencedColumnName = "nif")
  @NotNull
  private Usuario solicitante;

  @Column(name = "codmap_solicitante")
  private Integer codmapSolicitante;
```

Ilustración 6.15: Código - ejemplo entidad

Las anotaciones a las que se hacen uso en esta clase como, por ejemplo: *@Table, @Entity, @Column, @OneToMany* son anotaciones JPA que se utilizan para mapear una tabla en Java.

JPA (*Java Persistence API*) [13] es la API de persistencia desarrollada para la plataforma Java EE. Es un *framework* del lenguaje JAVA que maneja datos relacionales en aplicaciones. Viene definida en el paquete *javax.persistence*. El objetivo principal de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos siguiendo el patrón de mapeo objeto-relacional.

Para concluir con los ejemplos, a continuación, se muestra el ejemplo de una clase repositorio que opera sobre la base de datos. Estos están implementados mediante Spring Data. Los *querys* se implementan directamente mediante la anotación *@Query*. Esta es una de las maneras de realizar una operación o consulta sobre la base de datos mediante JPA en Spring.

```
@Transactional(readOnly = true)
public interface TareaRepository
  extends TareaRepositoryCustom, JpaRepository<Tarea, String> {

  /**
   * Realiza la proyección del listado de todas las tareas sobre un listado de
   * DTOs de cierto tipo (p.e. TareaInfo)
   */
  <T> List<T> findBy(Class<T> type);
  <T> Page<T> findBy(Pageable pageable, Class<T> type);

  Tarea findByCodtarea(String codtarea);

  /**
   * Obtención del listado de tareas activas filtradas según: codtarea, tema,
   * solicitante, estado, tipoTarea, fechaIni, numPaginas, numPalabras
   */
  @Query("SELECT t " + "FROM Tarea t "
      + "WHERE (t.estado.codestado BETWEEN '1' AND '7') "
      + "AND (UPPER(t.tema) LIKE %:search% "
      + "OR t.codtarea LIKE %:search% "
      + "OR UPPER(t.solicitante.nombre)||' '||"
      + "UPPER(t.solicitante.apellido1) LIKE %:search% "
      + "OR UPPER(t.estado.estado) LIKE %:search% "
      + "OR UPPER(t.tipoTarea.tipoTarea) LIKE %:search% "
      + "OR FUNCTION('to_char', t.fechaIni, 'DD-MM-YYYY') LIKE %:search% "
      + "OR CAST(t.docOri.numPaginas AS string) LIKE %:search% "
      + "OR CAST(t.docOri.numPalabras AS string) LIKE %:search%)")
  Page<Tarea> findTareasActivasPageBySearch(@Param("search") String search,
      Pageable pageable);
```

Ilustración 6.16: Código - ejemplo repositorio

7. Producto desarrollado

Este apartado se dedicará a mostrar cómo se puede utilizar la plataforma desarrollada para cumplir con los requisitos y los escenarios propuestos en apartados anteriores.

Se acompañará a cada escenario con diferentes imágenes de la página para así poder demostrar la interacción entre el usuario y la plataforma. Comenzaremos por los escenarios a realizar por usuario más básico de la aplicación, seguido por aquellos escenarios que realizaría un técnico lingüístico y finalizaremos con el rol de administrador.

7.1 Escenario 1: Crear una solicitud

El escenario comienza cuando, una vez autenticado, el usuario se encuentra en la página inicial de la aplicación. Para el rol de usuario esa página es la de "*Les meues tasques*".

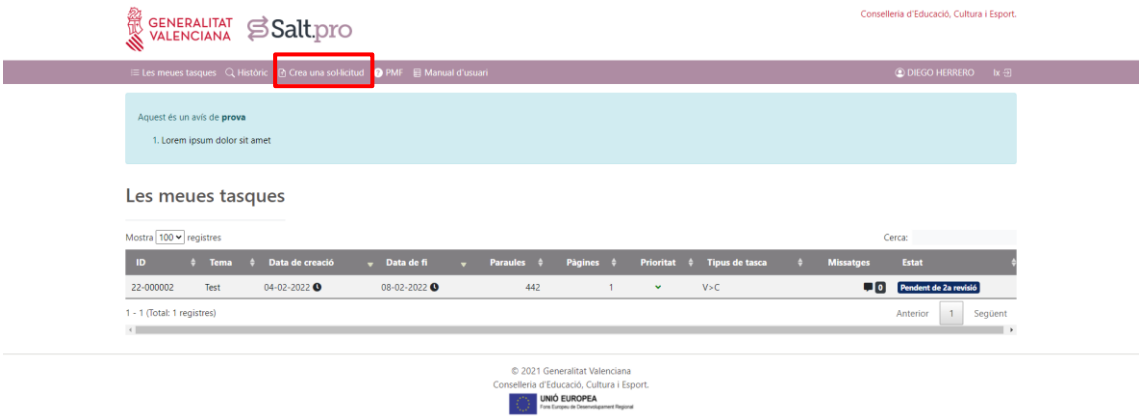

Ilustración 7.1: Pantalla "Les meues tasques" – usuario

En la barra del menú se encuentra el botón "*Crea una sol·licitud*" que está enmarcado en color rojo en la ilustración. Este botón lleva directamente a la página desde donde el usuario rellenará el formulario para crear una nueva tarea de traducción o revisión.

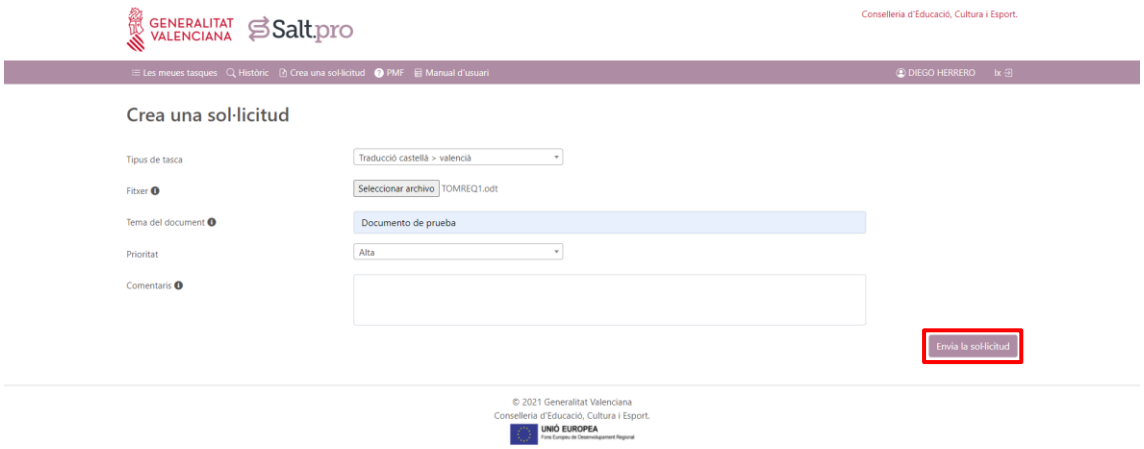

Ilustración 7.2: Pantalla "Crea una sol·licitud"

El usuario deberá rellenar los distintos campos del formulario y una vez haya finalizado mediante el botón "*Envia la sol·licitud*" podrá crear la tarea.

Algunos de los campos son desplegables que contienen varias opciones donde el usuario solo puede elegir una.

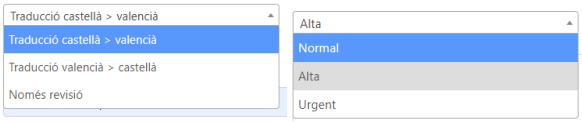

Ilustración 7.3: Desplegables

7.2 Escenario 2: Buscar una tarea procesada

Para buscar una tarea que el usuario creó y actualmente se encuentra como procesada (por ejemplo, cualquiera de los estados: Descargada, Rechazada, Cancelada o Finalizada) deberá pulsar el botón "*Històric*" que se encuentra en la barra del menú.

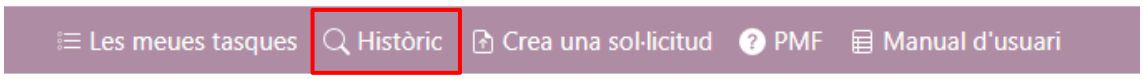

Ilustración 7.4: Barra de menú – usuario solicitante

Este botón llevara a una página donde se muestra un listado de todas las tareas procesadas de este usuario.

Ilustración 7.5: Pantalla "Històric" – usuario

Para realizar una búsqueda sobre dicho listado existe un formulario de búsqueda al que se accede pulsando el botón "*Mostra els filtres de cerca*".

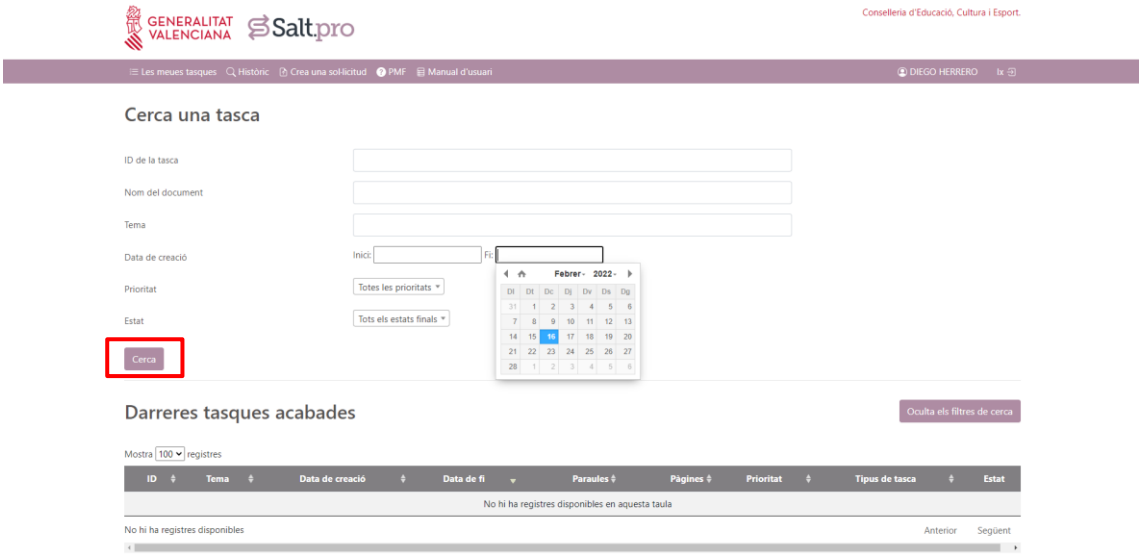

Ilustración 7.6: Pantalla "Històric" - Filtro de búsqueda

Para realizar una búsqueda concreta el usuario tendrá que rellenar los campos sobre los que desea realizar la búsqueda y pulsar el botón "*Cerca*". En caso de una búsqueda exitosa aparecería en el listado la tarea o las tareas deseadas. En caso contrario aparecería el mensaje "*No hi ha registres disponibles en aquesta taula*".

7.3 Escenario 3: Asignarse una tarea a uno mismo

Un técnico, una vez autenticado en la aplicación, puede asignarse una tarea no asignada a si mismo. La página inicial que un técnico verá al entrar a la plataforma será la de "Tasques no assignades". En caso de que no se encontrara en la página inicial, para acceder a ella lo único que tendría que hacer sería pulsar sobre el bóton "No assignades" de la barra del menú tal y como se muestra a continuación.

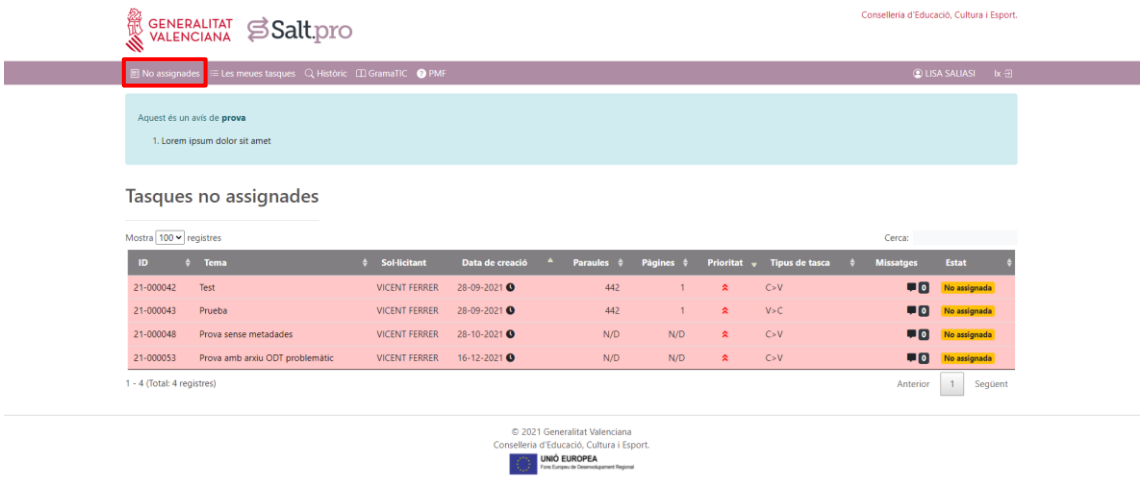

Ilustración 7.7: Pantalla "No assignades"

A continuación, pulsaría sobre cualquier tarea que desea asignarse y eso le llevaría a la página del detalle de tarea.

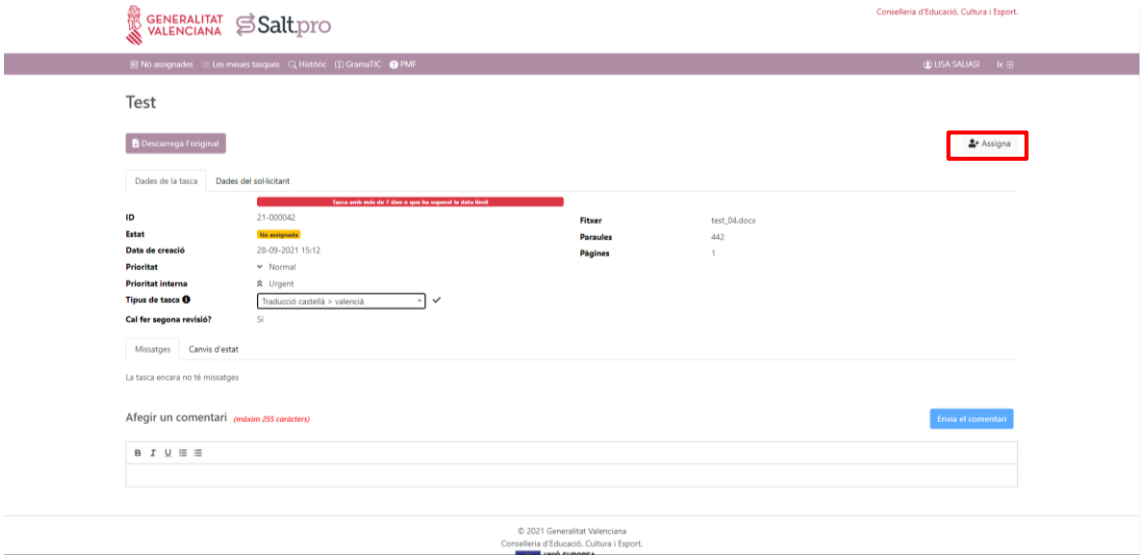

Ilustración 7.8: Pantalla tarea

En esta página se encontrará, además de los detalles de la tarea y del solicitante entre otras cosas, con un botón "*Assigna*" que deberá pulsar. A continuación, saltaría una pequeña ventana modal para confirmar la operación.

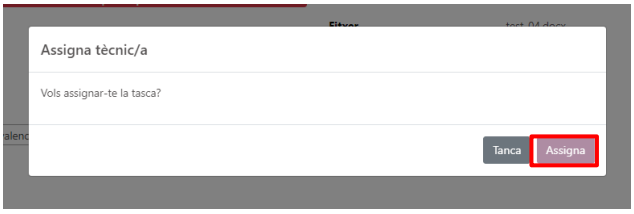

Ilustración 7.9: Pantalla ventana modal "Assigna tècnic"

Esta tarea ya sería propia de dicho técnico tras la asignación y aparecería en la ventana "*Les meues tasques*".

7.4 Escenario 4: Finalizar una tarea

Para que un técnico concreto pueda dar por finalizada una tarea tendrá que acceder a la página "*Les meues tasques*" pulsando para ello el botón del mismo nombre en la barra de menú.

Ilustración 7.10: Barra de menú - técnico

Eso le llevará a la página donde se encuentra el listado de tareas propias de dicho técnico.

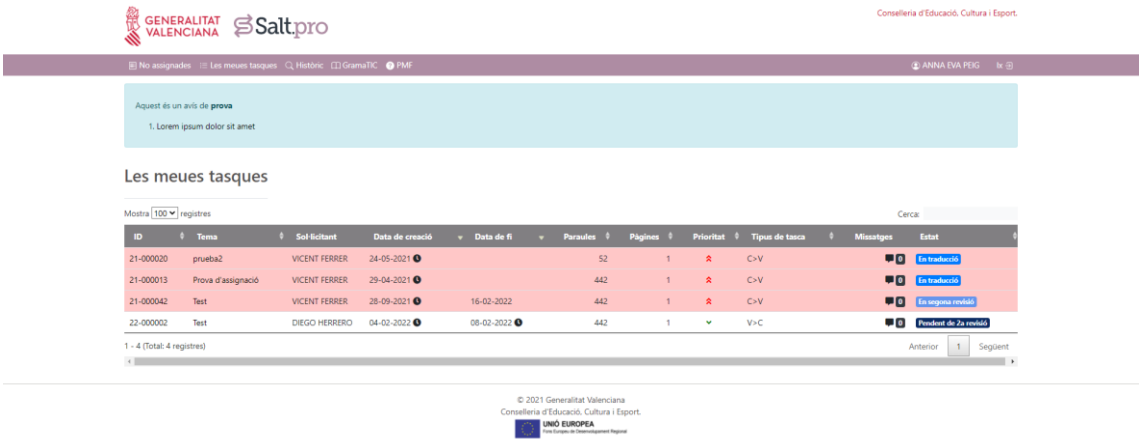

Ilustración 7.11: Pantalla "Les meues tasques" - técnico

Ahí elegirá y pulsará sobre la tarea que desea finalizar y eso le llevará a la página de detalle de dicha tarea donde encontrará un desplegable con la opción "*Acaba*".

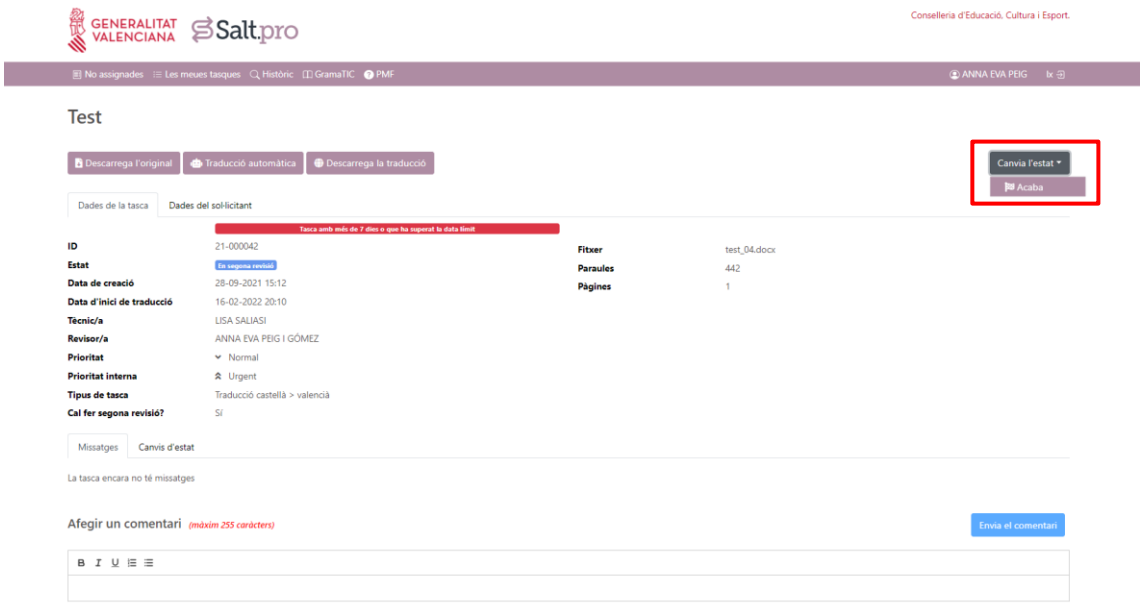

Ilustración 7.12: Pantalla tarea - técnico

Tras pulsar sobre esta opción, saltaría una ventana modal que pide, antes de poder finalizar la tarea, subir el fichero de la tarea traducido o revisado.

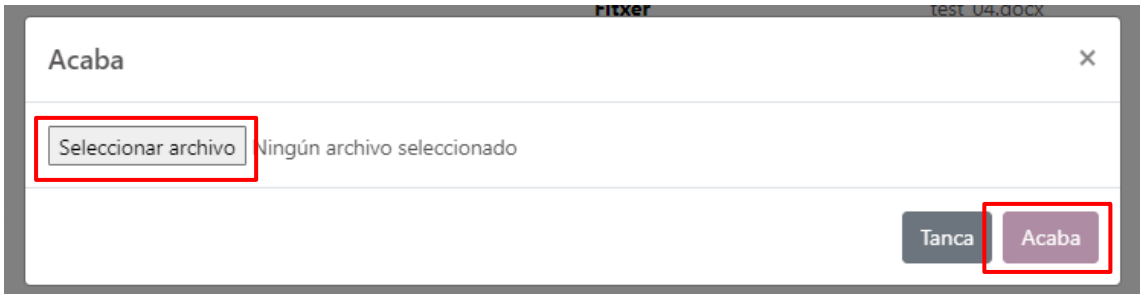

Ilustración 7.13: Pantalla - ventana modal "Finalizar tarea"

Tras seleccionar el fichero y terminar la tarea, dicha tarea cambiará de estado y aparecerá a partir de ese momento en el listado de tareas procesadas.

7.5 Escenario 5: Descargar fichero de estadísticas

Este es uno de aquellos escenarios que solo puede llevar a cabo un usuario con el rol administrador. Pues como se ha explicado anteriormente, los administradores son aquellos que más operaciones pueden realizar sobre el sistema incluyendo tanto las operaciones que lleva a cabo un usuario solicitante y las operaciones de un técnico lingüístico como otras operaciones exclusivas de un administrador.

Para consultar y descargar el fichero en formato CSV de las estadísticas un administrador tendrá que navegar hasta la página de estadísticas. Para ello pulsará

sobre el desplegable "*Supervisió*" en la barra del menú y elegirá la opción "*Estadístiques*".

Ilustración 7.14: Barra de menú - Administrador

Tendrá que navegar hasta el final de dicha página y ahí encontrará el botón "*Descarrega les estadístiques*" que deberá pulsar para bajarse el fichero.

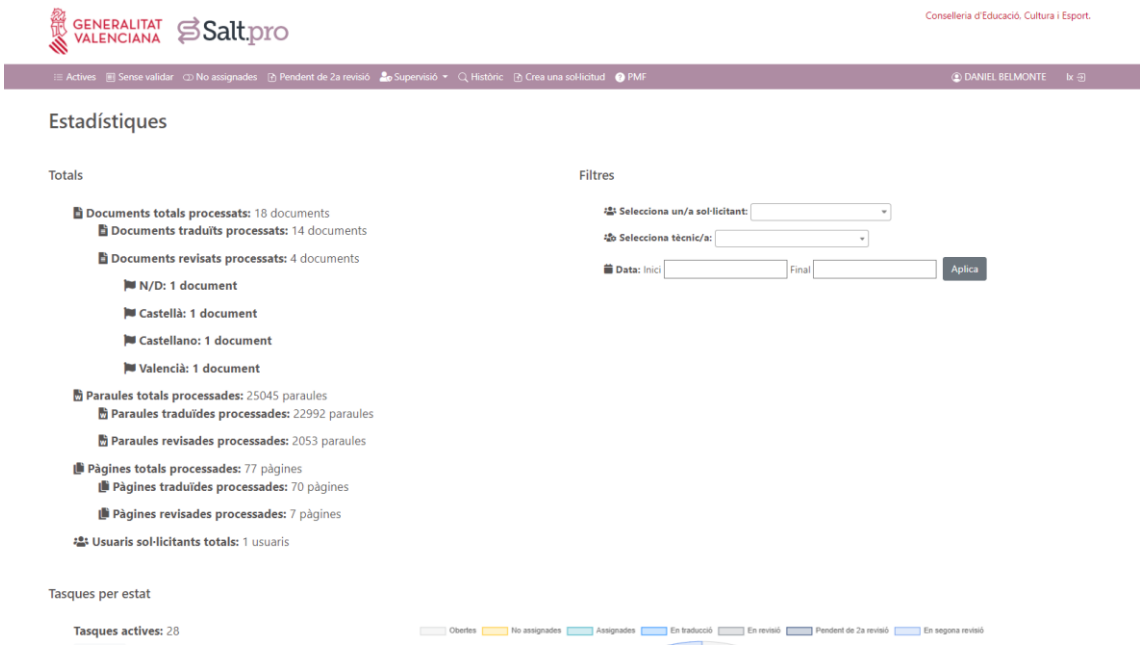

Ilustración 7.15: Pantalla "Estadístiques" - Inicio

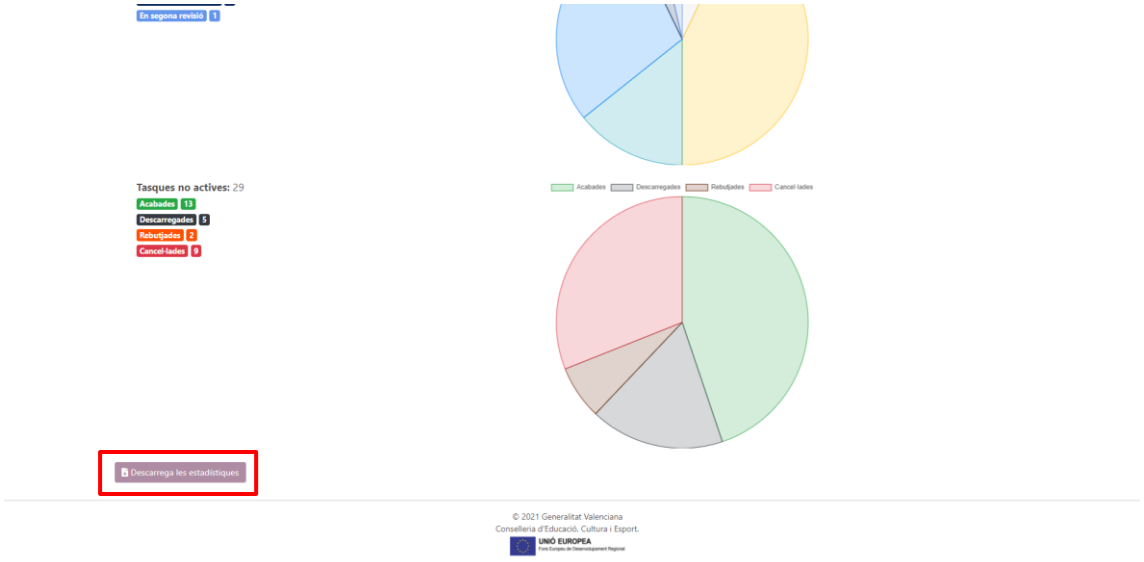

Ilustración 7.16: Pantalla "Estadístiques" - Final

7.6 Escenario 6: Mostrar un aviso general en el sistema

Para concluir este apartado, se explicará a continuación otro escenario exclusivo de un administrador.

Este escenario se considera una configuración del sistema por lo tanto se realiza desde la página de "*Configuració*" a la que el administrador tiene acceso y para acceder tendría que seleccionar la opción "*Configuració*" que se encuentra en el desplegable "*Supervisió*" de la barra de menú.

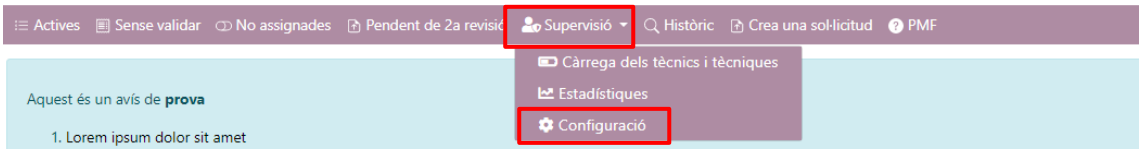

Ilustración 7.17: Barra de menú - Administrador

Una vez en la página, el administrador tendría que rellenar el campo de "*Avís general*" y guardaría la configuración pulsando el botón "*Guarda*".

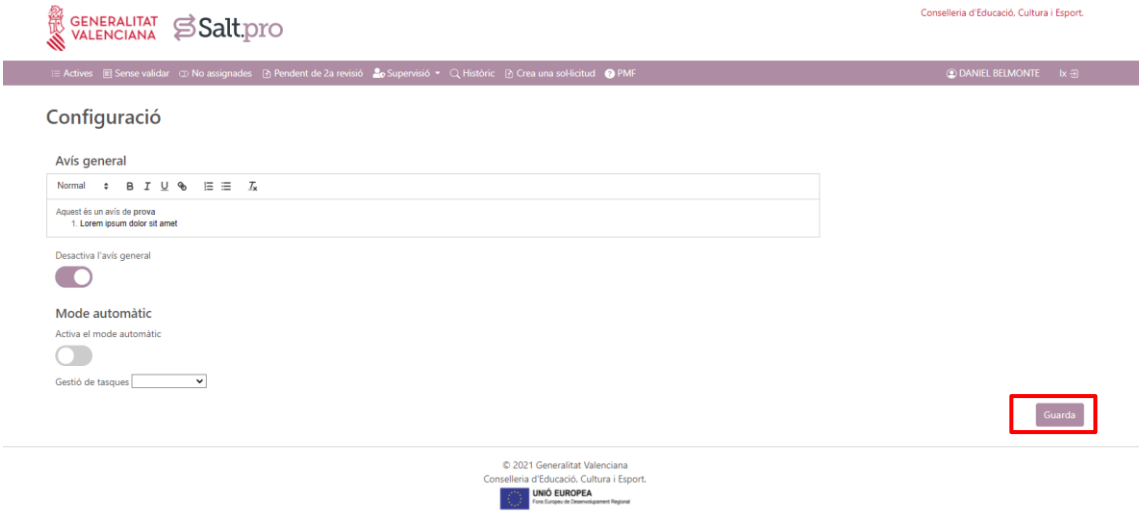

Ilustración 7.18: Pantalla "Configuració"

El aviso está pensado para que esté siempre activo, pero en caso de que estuviera desactivado para volver a activarlo habría que pulsar sobre el botón de "*Desactiva l'avís general*".

Ilustración 7.19: Botón "Desactiva l'avís general

8. Pruebas unitarias

Un aspecto muy importante de un proyecto es la calidad de software. La calidad de software es un conjunto de cualidades que caracterizan al software y que determinan su utilidad y existencia. Para medir la calidad del software se miden características como eficiencia, flexibilidad, corrección, confiabilidad, mantenibilidad, portabilidad, usabilidad, seguridad e integridad. Estas características del software pueden ser medibles y, por tanto, fácilmente comparables.

Los sistemas de software son cada vez más importantes en la sociedad actual y crecen rápidamente en tamaño y complejidad, por lo que desarrollar software de calidad, basado en estándares con funcionalidad y rendimiento ajustado a las necesidades y exigencias del cliente, son aspectos fundamentales para asegurar el éxito del producto software.

El Sello de Excelencia consiste en una serie de criterios de calidad que permiten evaluar la Seguridad, Fiabilidad y Mantenibilidad de las aplicaciones desarrolladas por y para la DGTIC.

La Oficina de Certificación del Software de la GVA es quien define los criterios que componen el Sello de Excelencia y es quien trabaja en el mantenimiento y la mejora continua del Sello de Excelencia.

Uno de los criterios que componen el Sello de Excelencia es el de la cobertura de código mediante pruebas.

Las pruebas unitarias son aquellas encargadas de validar, independientemente del resto de elementos, el funcionamiento específico de un método. El objetivo es comprobar si cada unidad funcional, por ejemplo, un método público, se comporta como se espera y devuelve el resultado esperado.

Quedan excluidos de las pruebas unitarias los métodos *getter* y *setter* los cuales solo proporcionan acceso a las propiedades de una clase.

Algunos casos típicos de validaciones mediante pruebas unitarias son los métodos con la lógica de la aplicación la capa de servicios.

En las pruebas unitarias, es muy importante aislar las pruebas de elementos externos que podrían distorsionarlas. Por este motivo, si el método a probar tiene dependencias con otros elementos, por ejemplo, con un repositorio de la capa de acceso a datos o un servicio web de la capa de integración, deben reemplazarse con objetos ficticios que simulan su comportamiento de forma controlada.

Para la realización de dichas pruebas en la aplicación Salt.pro se utilizan las siguientes tecnologías:

- **JUnit [15]**: Es uno de los *framework* Java más utilizados para el desarrollo de pruebas.

- **AssertJ [16]**: Librería cuyas utilidades permiten crear de forma sencilla reglas de validación.
- **Mockito [18]**: *Framework* que facilita y simplifica el aislar las pruebas en aquellos casos en los que la clase a probar tiene dependencias con otros elementos.
- **Spring Test**: Utilidades de Spring para las pruebas.

En las clases de pruebas, cada método en el que se desarrollen las pruebas se debe anotar con la anotación *@Test* de Junit.

```
@Test
public final void findTareaByCodtareaTarea() {
```

Ilustración 8.1: Código - Ejemplo de uso de la anotación @Test

Se utiliza un patrón de 3 fases que define los pasos a seguir para cada prueba:

- Preparación: Define la precondición inicializando los datos de la prueba.
- Ejecución: Ejecuta el comportamiento del código a validar en la prueba.
- Validación: Define la postcondición de la prueba para verificar su resultado.

En la fase de preparación se creará la instancia de la clase a validar y aquellos valores que con necesarios para establecer su estadio previo a ejecutar la prueba.

Esto se llevará a cabo en un método global que lleva la anotación *@Before* de Junit, el cual se ejecutará de forma automática antes de cada método de la prueba de la clase:

```
@Before
public void setUp() {
  TipoDocumento tipoDocOri = new TipoDocumento();
  tipoDocOri.setCodtipodoc((short) 1);
  tipoDocOri.setTipoDocumento("Original");
  tipoDocOri.setVersion(0L);

  Documento docOri = new Documento();
  docOri.setCoddoc(0L);
  docOri.setTitulo("TEST");
  docOri.setNumPaginas(3L);
  docOri.setNumPalabras(500L);
  docOri.setIdioma("TEST");
  docOri.setFicheroOriginal("test.txt");
  docOri.setFichero("test_ori.txt");
  docOri.setFormato("txt");
  docOri.setFechaSubida(new Date());
  docOri.setCodtipodoc(tipoDocOri);
  docOri.setVersion(0L);
```

Ilustración 8.2: Código - Ejemplo de uso de la anotación @Before

En el ejemplo anterior, se crea un nuevo objeto TipoDocumento y Documento inicializando algunas de sus propiedades, estos objetos junto a otros se utilizarán en los métodos de prueba para hacer validaciones y simulaciones.

En la fase de ejecución se ejecutará el comportamiento que se quiere validar invocando el método de la clase a probar.

```
//Ejecución del Test
Tarea resultado = service.findTareaByCoddoc(0L);
```

Ilustración 8.3: Código - Ejemplo ejecución del test

En el ejemplo, se llama al método findTareaByCoddoc de la clase TareaServiceImpl.

En fase de validación se verifica si el comportamiento del método en prueba es el esperado.

La validación de la prueba se fundamenta en el uso de aserciones (*asserts*). En la prueba, la aserción debe ser un predicado que compruebe qué ha sucedido en la fase de ejecución, por ejemplo, validando que el comportamiento es el esperado.

Si el predicado se cumple, es decir se evalúa a cierto (*true*), la prueba se ha comportado de la forma esperada y por tanto es satisfactoria. Por el contrario, si el predicado no se cumple, es decir se evalúa a falso(*false*) no es el comportamiento esperado y la prueba no es satisfactoria. JUnit recoge este resultado negativo y marca la prueba como fallida en el informe de ejecución de las pruebas [15].

```
//Validación del Test
assertThat(resultado).as(
        "El objeto recibido no es el esperado"
).isEqualTo(tarea);
```

Ilustración 8.4: Código - Ejemplo validación del test

En el ejemplo anterior, se verifica que el objeto Tarea devuelto como resultado de la ejecución del método en el ejemplo anterior es el correcto.

Como se ha comentado anteriormente en este apartado, para probar clases que dependen de otros elementos se crean objetos ficticios. Los objetos ficticios tienen que ser inicializados antes de ser utilizados. Para ello, se declara en la clase de prueba una propiedad anotada con *@Rule [17]*.

```
/**
 * Regla JUnit para inicializar Mocks y validar el uso de Mockito
 */
@Rule
public MockitoRule mockito = MockitoJUnit.rule();
```

Ilustración 8.5: Código - Ejemplo Regla JUnit

En las pruebas realizados en Salt.pro se han definido dos tipos de objetos ficticios.

- Dummy: Son aquellos en los que su estado no es relevante para las pruebas. Son necesarios únicamente porque son requeridos como parámetro para la invocación de algún método.

```
@Mock
TareaRepository repository;
```

Ilustración 8.6: Código - Ejemplo "dummy"

  En este ejemplo, en lugar de implementar manualmente la creación de una instancia de la clase *TareaRepository*, se genera automáticamente un objeto ficticio mediante la propiedad *repository* anotada con *@Mock [17]*.
- Stub: El objetivo de estos es devolver valores controlados al objeto que se prueba, estableciendo valores de respuesta concretos.

```
/* Stub del método findTareaByCoddoc */
when(repository.findTareaByCoddoc(0L)).thenReturn(tarea);
```

Ilustración 8.7: Código - Ejemplo "Stub"

  Los *stubs* se crean a partir de objetos *dummy* con el método *when [17]* de Mockito. En el ejemplo anterior, cuando el elemento que esté siendo probado realice una llamada al método *findTareaByCoddoc* del repositorio con el valor *0L* como parámetro, la respuesta del método será el objeto tarea.

9. Conclusiones

De acuerdo con lo expuesto en este documento podemos afirmar que el desarrollo de la nueva aplicación en comparación con la antigua versión ha tenido efectos positivos en cuanto a mejoras de rendimiento y la monitorización del flujo de trabajo de Salt.pro.

Se ha conseguido también mejorar y pasar de una interfaz, aunque simple y efectiva, confusa y poco intuitiva a una interfaz que mejora la facilidad de uso de la herramienta utilizando tecnologías más modernas que permiten ampliar el abanico de posibilidades de diseño y funcionamiento.

El proyecto actualmente se encuentra finalizado, aunque en estado de mantenimiento y es posible incorporar nuevas mejoras si el cliente lo requiere como ha ocurrido desde la finalización del desarrollo del proyecto y su pase a producción.

Personalmente este proyecto ha sido un gran reto, comencé con conocimientos básicos teóricos sobre las tecnologías utilizadas. Si tenemos en cuenta las asignaturas cursadas a lo largo de mi paso por el grado, se abarca una multitud de materias estudiadas como por ejemplo, base de datos en asignaturas como BDA y TBD; Desarrollo Web y diseño de interfaces estudiadas en DEW e IPC; todo lo relacionado con los análisis de requisitos, casos de uso y buenas prácticas estudiadas en la asignatura ISW e incluso leyes sobre la protección de datos vistas en DYP.
Con todo lo visto en estas materias que sirven de base para este trabajo, además de mucha investigación personal y aprendizaje por cuenta propia he podido conseguir ampliar mis conocimientos y experiencia e incorporarme en el mundo de desarrollo web con grandes tecnologías que desconocía.

Ahora tengo muchas ganas de seguir aprendiendo y seguir por este camino.

Bibliografía

[1] DGTIC. Metodología gvLOGOS. https://dgtic.gva.es/va/metodologia-gvlogos
[2] Rock Content. Bootstrap: guía para principiantes de qué es, por qué y cómo usarlo. https://rockcontent.com/es/blog/bootstrap/
[3] MDN Web Docs. CSS. https://developer.mozilla.org/es/docs/Web/CSS
[4] MDN Web Docs. JavaScript. https://developer.mozilla.org/es/docs/Web/JavaScript
[5] Baeldung. Introducción al uso de Thymeleaf en Spring. https://www.baeldung.com/thymeleaf-in-spring-mvc
[6] NTTDATA.Spring Framework: Qué es y por qué usarlo. https://ifgeekthen.nttdata.com/es/spring-framework
[7] Java desde 0. Diferencias entre Spring y Spring Boot. https://javadesde0.com/diferencias-entre-spring-y-spring-boot/
[8] Sentrio Blog. Introducción a Jenkins: ¿qué es, para qué se sirve y cómo funciona? https://sentrio.io/blog/que-es-jenkins/
[9] NTTDATA. ¿Qué es Hibernate? ¿Por qué usarlo? https://ifgeekthen.nttdata.com/es/que-es-java-hibernate-por-que-usarlo
[10] Platzi. Curso de PostgreSQL. https://platzi.com/cursos/postgresql/
[11] Aguilar, J. M. ¿Qué es el patrón MVC en programación y por qué es útil? https://www.campusmvp.es/recursos/post/que-es-el-patron-mvc-en-programacion-y-por-que-es-util.aspx
[12] Microsoft. Arquitecturas de aplicaciones web comunes. https://docs.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures
[13] Wikipedia. Java Persistence API. https://es.wikipedia.org/wiki/Java_Persistence_API
[14] spring.io https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/servlet/ModelAndView.html
[15] javadoc.io. JUnit http://junit.sourceforge.net/javadoc/org/junit/Assert.html#assertThat(T,%20org.hamcrest.Matcher)
[16] github.io. AssertJ https://assertj.github.io/doc/
[17] javadoc.io. Mockito https://www.javadoc.io/doc/org.mockito/mockito-core/1.10.19/index.html
[18] Mockito.org https://site.mockito.org/
[19] Desarrollo de aplicaciones mediante el framework de Spring / Eugenia Pérez Martínez. ISBN: 978-849-964-556-8
[20] Hibernate: Persistencia de objetos en JEE / Eugenia Pérez Martínez ISBN: 978-849-964-558-2
[21] Código limpio: Manual de estilo para el desarrollo ágil de software / Robert C. Martin ISBN:978-84-415-3210-6

Anexo

Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**				X
ODS 5. **Igualdad de género.**				X
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**	X			
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades**.				X
ODS 11. **Ciudades y comunidades sostenibles**.				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**		X		

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este TFG trata sobre el desarrollo de una nueva plataforma web más eficiente y eficaz por lo que estaría directamente relacionado con los ODS de "Trabajo decente y crecimiento económico", "Industria, innovación e infraestructuras" y "Alianzas para lograr objetivos". A continuación, se explica detalladamente su relación con dichos ODS.

- **ODS 8. Trabajo decente y crecimiento económico.**

Se percibe la existencia de un vínculo alto ya que dicha herramienta promueve directamente el teletrabajo utilizando tecnologías innovadoras, así como también promueve el empleo y los entornos de trabajo seguros.

- **ODS 9. Industria, innovación e infraestructuras.**

El problema principal con el que nos encontramos es que las infraestructuras básicas como pueden ser las tecnologías de la información y las comunicaciones siguen siendo escasos en muchos países en desarrollo. Con herramientas desarrolladas con las últimas tecnologías promueve una innovación tecnológica, así como el crecimiento sostenible de los países.

- **ODS 17. Alianzas para lograr objetivos.**

Existe un vínculo directo especialmente por la meta 17.8 de dicho ODS *"…aumentar la utilización de tecnologías instrumentales, en particular la tecnología de la información y las comunicaciones".*