Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERIA DE SOFTWARE

Una arquitectura de microservicios Python como soporte a un diccionario de datos en un sistema de aseguramiento de la calidad y asistencia al desarrollo de IAMs por grandes equipos globalmente distribuidos.

Alumno/a: David Gómez Pedriza

Tutor/es/as: Yania Crespo González-Carvajal

Agradecimientos

En primer lugar, dedico unas líneas para toda la gente que me ha apoyado y que gracias a la aportación de cada uno, el conjunto, ha supuesto un gran impulso para la realización del proyecto.

Cómo no, a mi familia, Andrea, Reyes y Jesús, que han estado siempre detrás de mí en todo momento, aportando la presión necesaria en los buenos momentos y el optimismo en los malos. Por otra parte, a mis amigos y compañeros de carrera que me han ayudado con consejos y ánimos durante muchos años. Y finalmente, gracias a los profesores de la Universidad de Valladolid, que no solo han participado en mi formación sino que han hecho posible que llevara a cabo este proyecto, en especial a Yania, que ha sido la tutora del mismo y una guía indispensable para su realización.

Resumen

Este trabajo se realiza en el marco del proyecto europeo LOCOMOTION como parte del programa Horizon 2020. En LOCOMOTION se desarrollan modelos llamados *Integrated Assessment Models* (IAMs) que permiten estudiar las altamente complejas interacciones entre las personas y el medio ambiente con el objetivo de simular políticas y comportamiento para observar sus consecuencias en el futuro, a través de técnicas de Dinámica de Sistemas y el modelado con el software de simulación Vensim.

Este proyecto se realiza de forma coordinada por 13 socios (Universidades, Centros de Investigación, Agencias de Energía, etc.) de diferentes países europeos. Participan más de 25 programadores de modelos de simulación repartidos en 9 módulos a integrar. La UVa es el socio coordinador y responsable de la coordinación técnica y el aseguramiento de la calidad.

El objetivo del proyecto es proporcionar un diccionario de datos compartido que, por una parte, sea accesible al público no integrado en el proyecto como medio de diseminación y, por otra parte, sirva de apoyo a los programadores del modelo para trabajar de forma coordinada aunque geográficamente distribuida. Este diccionario de datos explica de forma estructurada el conocimiento del modelo y una terminología común. Se integra con un sistema automático de control de calidad que revisa los modelos realizados en Vensim y coordina lo plasmado en estos modelos con la información en el diccionario de datos.

El diccionario de datos se diseña para ser implementado bajo una arquitectura basada en microservicios, capaz de escalar con despliegues flexibles y tecnología variada, pero independiente, para poder adaptarse a los cambios futuros y evolucionar con ellos.

Abstract

This work is carried out as part of the European project LOCOMOTION funded from the EU's Horizon 2020 programme.

LOCOMOTION develops models called *Integrated Assessment Models* (IAMs), they allow studying highly complex interactions between people and the environment with the aim of simulating policies and behavior to observe their consequences in the future, applying System Dynamics techniques and modeling with Vensim simulation software.

This project is carried out in coordination by 13 partners (Universities, Research Centers, Energy Agencies, etc.) from different European countries. More than 25 simulation model programmers participate in 9 modules to be integrated. Uva is the coordinator partnet and responsible for technical coordination and quality assurance.

The objective of the project is to provide a shared data dictionary that, on one hand, is accessible to the public not integrated into the project as a means of dissemination and, on the other hand, supports the programmers of the model to work in a coordinated way although geographically distributed. This data dictionary explains in a structured way the knowledge of the model and a common terminology. It is integrated with an automatic quality control system that reviews the models made in Vensim and coordinates what is shown in these models with the information in the data dictionary.

The data dictionary is designed to be implemented under a microservices-based architecture, capable of scaling with flexible deployments and varied but independent technology, in order to adapt and evolve to future changes in the project.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

En términos generales el proyecto puede ser dividido en dos pilares bien distinguidos alrededor de los cuales se ha llevado a cabo todo el proceso de planificación y desarrollo del mismo. Aunque siendo ambas fuentes de inspiración para su desarrollo de un peso similar, se deberían clasificar a su vez de maneras diferentes, por un lado, los microservicios como objetivo de cara al funcionamiento interno y planificación del sistema, su esqueleto en pocas palabras, y por otro lado, el proyecto europeo LOCOMOTION del programa Horizon 2020 (H2020)[15] como contexto de este Trabajo de Fin de Grado (TFG).

El proyecto LOCOMOTION es la fuente de creación, como entidad contribuidora de la lógica de negocio a abordar con soporte en el sistema de microservicios, actúa como cliente y entidad financiadora.

Europa y el mundo se encuentran en un momento de cambio. El clima global se está calentando. Las especies se están muriendo a un ratio sin precedentes, llevando a la pérdida masiva de de biodiversidad. Los ecosistemas están colapsando. Muchos de los recursos finitos del mundo se están sobreexplotando. Como centro de esta crisis de alcance mundial se encuentra la humanidad.

Los ciudadanos demandan cambios, mientras que la sociedad civil y políticos buscan alternativas sostenibles para un futuro bajo en carbón, con cero emisiones netas. LOCOMOTION está ayudando en este esfuerzo desarrollando modelos sofisticados para evaluar el impacto socio-económico y medioambiental de diferentes opciones políticas con el fin de ayudar a la sociedad a tomar decisiones informadas a cerca de la transición a un futuro sostenible, bajo en carbón.

LOCOMOTION es un proyecto coordinado a nivel europeo con 13 socios, entre los cuales se encuentra la Universidad de Valladolid como socio coordinador.

1.2. Objetivos

Como se acaba de indicar, el objetivo principal del sistema es proporcionar una infraestructura basada en microservicios que dé soporte a la lógica de negocio, en este caso un diccionario de datos para la compartición de conocimiento y el aseguramiento de la calidad en LOCOMOTION, un proyecto coordinado de desarrollo de *Integrated Assesment Models* (IAMs)[5].

La lógica de negocio en sí misma influye en la planificación de los microservicios, ya que el sistema necesita tener la carga balanceada para poder dar respuesta a las peticiones de forma eficiente, con esto se hace referencia a la cantidad de peticiones que maneja cada módulo de trabajo y la cercanía de la información obtenida.

1.2.1. Objetivos en cuanto a microservicios

Fundamentalmente se pueden distinguir los siguientes objetivos dentro del desarrollo y planificación basado en microservicios web:

- Reparto del trabajo que normalmente realizaría una aplicación monolítica en segmentos menores
- Escalabilidad del sistema en términos de capacidad computacional
- Despliegue sencillo del sistema e independencia entre módulos de trabajo

En cuanto a las ventajas y desventajas de este tipo de arquitectura, serán abordadas más a fondo en el Capítulo 5 de esta memoria.

El anteriormente mencionado balance de carga, es algo fundamental para el desarrollo del proyecto, pero a su vez puede ser ajustado a medida que este crece, asignando en el despliegue del sistema mayor capacidad a los módulos cuya densidad sea mayor.

1.2.2. Objetivos en cuanto al contexto

El sistema es desarrollado bajo el marco de trabajo proporcionado por el proyecto LOCOMOTION. Tiene como objetivo, por una parte, dar una representación comprensible de un proyecto basado en IAM's, con bases complejas para que pueda ser comprendido por personas ajenas al proyecto. Por otra parte, el diccionario de datos servirá de soporte al conocimiento compartido por el equipo de desarrollo, al establecimiento de un vocabulario común, así como al control de calidad establecido en el proyecto (cumplimiento de reglas de nombrado y estándares). Por tanto, en su lógica de negocio se definirán un conjunto de roles en el proyecto y las tareas que las personas con dichos roles realizan con el diccionario de datos compartido.

Este diccionario de datos consta de varias partes que serán definidas en el resto de Capítulos de esta memoria, en los requisitos del sistema, y más adelante serán representadas en el análisis y diseño del proyecto.

Finalmente, se desea puntualizar que el diccionario de datos no es más que una representación abstracta del conocimiento de los IAMs de LOCOMOTION y que no influye en su simulación real, realizada en el sistema Vensim[34]. Sin embargo, sí se integra con la herramienta desarrollada para el control automático de la calidad de los modelos programados en Vensim.

1.3. Estructura de la memoria

La memoria de este TFG se estructura de la siguiente forma:

Capítulo 2 : En este Capítulo se presentan los requisitos del proyecto y las historias de usuario que lo definen, así como los roles y protocolo de gobierno del mismo.

Capítulo 3 : En este Capítulo se presenta el plan del proyecto, su distribución y organización, los riesgos previstos y sus planes de actuación y por último, se hace una previsión de costes simulada.

Capítulo 4 : En este Capítulo se presentan las herramientas y tecnologías utilizadas, se definen sus propósitos dentro del proyecto y sus características principales.

Capítulo 5 : En este Capítulo se presenta el contexto del desarrollo basado en microservicios, los conflictos encontrados en diseño, comunicación y por último despliegue.

Capítulo 6 : En este Capítulo se presentan el análisis y diseño conceptual del sistema así como los patrones utilizados para su implementación, despliegue y creación de estructuras de datos.

Capítulo 7 : En este Capítulo se presentan tanto el proceso de implementación del sistema, su instalación entre otras cosas, como las pruebas realizadas sobre el propio sistema.

Capítulo 8 : En este Capítulo se presenta el seguimiento realizado durante el desarrollo del proyecto en cada sprint, y se hace una síntesis a modo de conclusión sobre dicho desarrollo del mismo.

Capítulo 9 : En este Capítulo se presentan las conclusiones finales del proyecto y se cierra el mismo presentando varias opciones de trabajo futuro a través de las cuales se puede continuar desarrollando el sistema.

Apéndice A : En este Apéndice se presentan los manuales de despliegue, mantenimiento y usuario del sistema.

Apéndice B : En este Apéndice se presentan enlaces adicionales relacionados con el proyecto.

Bibliografía : Finalmente, tras establecer los capítulos y apéndices las referencias bibliográficas consultadas.

Capítulo 2

Requisitos e historias de usuario

Este Capítulo contiene la descripción de los requisitos funcionales, no funcionales y de información centrados alrededor del proyecto de desarrollo del diccionario de datos compartido, su gobierno, vocabulario controlado, integración con la comprobación automática de programas Vensim de acuerdo con las convenciones y reglas de nombrado establecidas y, además, con la diseminación del proyecto.

En este documento se aplica una aproximación normalizada de escritura de historias de usuario en concordancia con: Mike Cohn (Cohn, 2004). Las historias de usuario son usadas para representar tanto requisitos funcionales, como no funcionales o de información.

El modelo de historia de usuario es el siguiente: **Como** (rol/interesado) **quiero** (algo) **para** (beneficio).

El uso de la expresión "**para** (beneficio)" es empleado para priorizar las historias de usuario del backlog del proyecto y/o para determinar el valor de implementar la historia de usuario, o bien, para tener un seguimiento de los objetivos de los diferentes interesados (stakeholders).

Definición del contexto:

- Un proyecto de desarrollo de IAM
- Un equipo distribuido geográficamente
- Un conjunto grande de símbolos (variables, constantes, parámetros, etc.) que gestionar

Definición de los interesados en este contexto:

- El líder del proyecto
- El equipo de revisión de calidad

- Usuarios que no son parte del proyecto
- Miembros del proyecto con responsabilidades y tareas diferentes

2.1. Roles y protocolo de gobierno del diccionario de datos

En el diccionario de datos la información será recogida de dos formas diferentes:

1. automáticamente desde los programas Vensim mediante inyección desde el servicio API.
2. a través de un formulario parte del diccionario de datos web, como parte de la base de datos (Paquete de trabajo 2 de "LOCOMOTION")

La documentación de cada símbolo, usado en el modelo, que incluirá será al menos el nombre completo, la descripción del símbolo, las unidades, la clasificación dependiendo de múltiples criterios (módulos, categorías), el método de obtención o cálculo, etc..

El diccionario disponible en la página web incluirá también información de:

- acrónimos (para cada acrónimo: sus letras y significado).
- adjetivos (para cada adjetivo: la palabra y su uso en el modelo) .
- reglas semánticas (para cada regla: la regla para abreviar y su explicación).

La información de los acrónimos será usada en las convenciones de nombrado de los símbolos y serán revisadas de forma automática para que el acrónimo sea siempre usado en mayúsculas como parte de un nombre más largo en presencia del servicio de diccionario de datos.

La información de los adjetivos pretende ayudar en las convenciones de nombrado. Los modeladores deberían revisar la información del diccionario web cuando seleccionen un nombre para un símbolo con el objetivo de usar los adjetivos de forma consistente (ejemplos: total, final, required, ...).

La información de las reglas semánticas pretende capturar el conocimiento del dominio que debería ser preservada o respetada en el IAM. Se sugiere que las reglas semánticas sean capturadas como posibles "Reality check" (parecido a los asertos) en Vensim.

Proponemos un protocolo para el diccionario de datos basado en módulos y roles sobre los módulos. Cada modificación (inserción, actualización, borrado) en el diccionario de datos debería estar validada. Los usuarios deberán estar autorizados para modificar el diccionario de datos de acuerdo con la siguiente organización:

- Todos los usuarios deberán estar identificados en la página web que gobierna el diccionario de datos.
- Cada usuario tiene roles. El rol determina las operaciones que el usuario puede realizar.
- Hay tres roles diferentes: supervisor general del proyecto(General Supervisor), supervisor de módulo(Module Supervisor), programador de módulo(Module Programmer).
- Programador de módulo: un usuario puede ser asignado programador de módulo de acuerdo con uno o más módulos. Los programadores de módulo pueden insertar y actualizar información en el diccionario de datos de símbolos (variables...) relacionadas con el módulo en el que están autorizados. Cada módulo puede tener varios programadores de módulo asociados. Cualquier modificación realizada en el diccionario de datos debe ser validada por un supervisor.
- Supervisor de módulo: un usuario puede ser asignado supervisor de módulo en un módulo en particular. El supervisor de módulo puede (y debe) validar la información que se encuentra pendiente de ser validada. El supervisor de módulo puede hacer las mismas modificaciones que los programadores de módulo. Adicionalmente, el supervisor de módulo puede borrar información. Solo hay un supervisor de módulo por cada módulo.
- Supervisor general del proyecto: un usuario puede ser designado supervisor general del proyecto. Es posible que haya más de un supervisor general por proyecto. Los supervisores generales pueden hacer las mismas tareas que un supervisor de módulo, están autorizados en todos los módulos(All).
- La organización debe garantizar que hay al menos un supervisor general, al menos un supervisor de módulo por módulo y al menos un programador por módulo.
-

El gobierno del diccionario de datos debería ser aplicado de acuerdo al siguiente protocolo:

1. La información es introducida generalmente por programadores de módulo y se queda pendiente de validación.
2. Un supervisor de módulo o supervisor general de proyecto puede revisar la información y validarla si es posible. Los supervisores pueden editar la información con el objetivo de ajustarla y mejorar su descripción.

Cuando la página web del diccionario de datos y sus servicios API relacionados estén disponibles, la automatización de las tareas del mantenimiento de calidad comprobará que cada programa Vensim sea consistente con la información ya validada del diccionario de datos. En el protocolo anterior, el paso 1 puede ser sustituido por una inyección automática desde programas Vensim (archivos .mdl). Esta inyección automática, tanto como la inserción manual a través de la aplicación web permanecerán pendientes de validación (paso 2).

Estos roles y protocolo de gobierto del diccionario de datos se definen en [6].

2.2. Historias de usuario

Como se ha mencionado anteriormente, los requerimientos del sistema serán descritos en historias de usuario, que principalmente serán separadas en tres categorías, esta separación esta basada en la necesidad de implementación de las propias historias, si la historia es necesaria o influye en el funcionamiento del sistema.

De esta forma, si una historia de usuario no es requerida para la ejecución normal del sistema, esta será incluida en la categoría "no funcional", al igual que una que sí sea requerida para dicha ejecución será incluida en la categoría "funcional", finalmente los requisitos de información están orientados a la información que el sistema debe almacenar.

2.2.1. Requisitos funcionales como historias de usuario

FR-1. Como líder de proyecto, quiero que los usuarios registrados en el sistema sean de tres tipos: programador de módulo, supervisor de módulo y supervisor general, para que el diccionario de datos centralizado ayude en la implantación del protocolo establecido en el proyecto para la definición y validación de símbolos.

FR-2. Como programador de módulo quiero ser capaz de introducir metadatos en el diccionario de datos para hacer más fácil el compartir información a cerca de los símbolos definidos en los módulos con otros miembros del proyecto y adicionalmente para facilitar la coordinación y asegurar la calidad, la diseminación del modelo, y su comprensión. Los metadatos registrados por un programador de módulo quedarán pendientes de revisión y validación por el supervisor de módulo o supervisor general, para mantener el diccionario de datos controlado de acuerdo al protocolo de revisión de calidad.

FR-3. Como programador de módulo quiero modificar los metadatos introducidos previamente en el diccionario de datos para que las decisiones tomadas a lo largo del desarrollo del proyecto estén actualizadas. Los metadatos que un programador de módulo modifica pasan a estar pendientes de revisión y validación por el supervisor de módulo o supervisor general, para mantener el diccionario de datos controlado de acuerdo a los protocolos que aseguran la calidad.

FR-3.1. Como supervisor de módulo quiero borrar los metadatos introducidos previamente en el diccionario de datos para que las decisiones tomadas a lo largo del desarrollo del proyecto estén actualizadas. El sistema deberá comprobar que no haya módulos secundarios asociados al símbolo a borrar y en caso de que los haya evitará el borrado. Un supervisor general puede eliminar del símbolo el módulo principal y promocionar uno secundario como principal. El efecto será como eliminar el símbolo de un módulo y mantenerlo en otros módulos.

FR-3.2. Como supervisor general quiero añadir un índice para que las decisiones tomadas a lo largo del desarrollo del proyecto estén actualizadas en el diccionario de datos. El sistema debe permitir asociar elementos a los índices, que representarán sus valores, estos no tienen porqué ser únicos.

FR-3.3. Como supervisor general quiero poder modificar la información de un índice y la de sus elementos para que las decisiones tomadas a lo largo del desarrollo del proyecto estén actualizadas en el diccionario de datos.

FR-3.4. Como supervisor general quiero borrar un índice para que las decisiones tomadas a lo largo del desarrollo del proyecto estén actualizadas en el diccionario de datos. Se mostrarán y comprobarán los símbolos asociados al índice seleccionado, y se impedirá el borrado del índice en caso de existir tal asociación. Por usabilidad se mostrará la lista de nombres de símbolos asociados a dicho índice para facilitar su búsqueda.

FR-4. Como supervisor de módulo quiero supervisar y validar los metadatos introducidos para asegurar la integridad, coordinación y la calidad de la información en el diccionario de datos.

FR-5. Como líder de proyecto quiero que los metadatos que un programador de módulo puede introducir o modificar en un módulo sea relativo a los módulos en los cuales el programador participa de forma que el diccionario de datos se mantiene controlado y se asegura el cumplimiento del protocolo de calidad.

FR-6. Como líder de proyecto quiero que los metadatos que los supervisores de módulo pueden validar y/o modificar sea relativo al módulo bajo su responsabilidad de forma que se asegure que el diccionario de datos se mantiene controlado bajo el protocolo de calidad.

FR-7. **(No se realiza en este TFG)** Como supervisor de módulo quiero recibir alertas cuando los metadatos sean introducidos o modificados para que pueda revisarlos y validarlos. Estas alertas serán mostradas en la aplicación web. Para evitar enviar emails de forma invasiva con cada modificación, un resumen semanal será enviado por email, solo en caso de que existan nuevas alertas.

FR-8. Como supervisor de módulo quiero poder realizar todas las tareas que pueda hacer un programador de módulo, referidas al módulo del que soy responsable para que pueda actuar en caso de indisposición o sobrecarga de trabajo de los programadores, o en caso de modificación sin que la validación sea requerida.

FR-9. Como supervisor de módulo quiero crear acrónimos, adjetivos y reglas semánticas en el diccionario de datos para facilitar a los miembros del proyecto el trabajo de nombrado de símbolos y las tareas de modelado, al mismo tiempo que faciliten la comprensión del proyecto para personas externas.

FR-10. Como supervisor de módulo quiero modificar o borrar acrónimos, adjetivos y reglas semánticas en el diccionario de datos para que las decisiones tomadas a lo largo del desarrollo del proyecto sean guardadas actualizadas en el mismo diccionario de datos.

FR-11. **(No se realiza en este TFG)** Como supervisor de módulo quiero ser capaz de obtener la información pública del diccionario de datos como un archivo Word con formato legible por humanos para facilitar la compartición de información con los stakeholders, y la generación de entregables del proyecto.

FR-12. Como supervisor general quiero realizar todas las tareas que un supervisor de módulo puede, referidas a todos los módulos, para que pueda actuar en caso de indisposición

o sobrecarga de trabajo de un supervisor de módulo, o en caso de modificación sin que se requiera validación.

FR-13. Como supervisor general quiero añadir módulos al proyecto para que la estructura del mismo quede delimitada, y las responsabilidades relativas a estos módulos puedan ser asignadas.

FR-13.1. Como supervisor general quiero añadir categorías al proyecto para que la estructura del mismo quede delimitada.

FR-14. Como supervisor general quiero modificar la información de los módulos para que las decisiones tomadas a lo largo del desarrollo del proyecto sean guardadas actualizadas en el diccionario de datos.

FR-14.1. Como supervisor general quiero modificar la información de las categorías para que las decisiones tomadas a lo largo del desarrollo del proyecto sean guardadas actualizadas en el diccionario de datos.

FR-15. Como supervisor general quiero borrar un módulo para mantener el diccionario de datos actualizado en base a las decisiones tomadas a lo largo del desarrollo del proyecto. No se permitirá borrar en caso de que haya símbolos o usuarios asignados al módulo.

Los borrados en cascada de roles de usuario o de símbolos no serán permitidos en ningún caso.

FR-15.1 **(No se realiza en este TFG)** En caso de que el módulo que va a ser borrado tenga usuarios con un rol asignado y símbolos asociados como módulo principal, el sistema permitirá reasignar a estos usuarios, sus roles, y los símbolos a otro módulo o borrarlos uno a uno de forma controlada.

FR-15.2. Como supervisor general quiero borrar una categoría para mantener el diccionario de datos actualizado en base a las decisiones tomadas a lo largo del desarrollo del proyecto. No se permitirá borrar en caso de que haya símbolos asociados a la categoría.

Los borrados en cascada de símbolos asociados no serán permitidos en ningún caso.

FR-15.2.1. **(No se realiza en este TFG)** En caso de que una categoría vaya a ser borrada y tenga símbolos asociados, el sistema permitirá reasignar estos símbolos a otra categoría.

FR-16. Como supervisor general quiero añadir un usuario, asignando al usuario al menos un rol para que los miembros del proyecto puedan ejecutar acciones sobre el diccionario de datos de acuerdo a las tareas definidas para cada rol. El rol de supervisor de módulo requiere indicar el módulo del que es responsable. El rol de programador de módulo requiere indicar el módulo o módulos en los que participa. El rol de supervisor general no está asociado con ningún módulo de forma particular, si no con todos al mismo tiempo.

FR-17. Como supervisor general quiero asignar un módulo a un usuario programador para que de esta forma pueda participar en varios módulos.

FR-18. Como supervisor general quiero asignar un nuevo rol a un usuario para que pueda participar en uno o varios módulos como programador y supervisor de módulo al mismo tiempo.

FR-19. Como supervisor general quiero ser capaz de borrar un usuario para que cualquier cambio en la plantilla del proyecto quede actualizado en el diccionario de datos.

FR-20. Como supervisor general quiero modificar o borrar el rol de un usuario para que cualquier cambio en la plantilla del proyecto quede actualizado en el diccionario de datos, o que se mitiguen los riesgos que conciernen la indisposición de los miembros del proyecto.

FR-21. **(No se realiza en este TFG)** Como supervisor general quiero añadir/modificar/borrar los valores asignados a "ProjectTypeOfValue", "ProgrammingLanguageSymbolType" y su asociación para que pueda adaptar el diccionario de datos al contexto del proyecto (ver IR-03).

FR-22. Como usuario registrado del sistema quiero poder consultar toda la información existente del diccionario de datos referida a módulos, símbolos y su caracterización, categorías (y subcategorías), acrónimos, adjetivos y reglas semánticas para que las tareas del proyecto se vean facilitadas. La información deberá ser mostrada filtrada de acuerdo a si está o no validada.

FR-23. Como líder de proyecto quiero que las personas externas al proyecto puedan consultar toda la información del diccionario de datos sin tener que estar registrados como usuarios, evitando así almacenar datos personales para que la transparencia y diseminación del proyecto queden garantizadas al mismo tiempo que se cumple la ley de protección de datos GDPR. La información visible a los usuarios no registrados **excluye** todos los aspectos relacionados con los usuarios participantes en el proyecto y sus roles así como cualquier información **NO** validada.

Nota: Los requisitos identificados como X.Y o X.Y.Z se han introducido en varias iteraciones durante el seguimiento del proyecto. Con el objetivo de mantener los identificadores de cara a la trazabilidad en el seguimiento, pero a la vez facilitar la comprensión de la funcionalidad se han introducido en la lista final de requisitos en la mejor posición de acuerdo a los otros requisitos relacionados y se ha optado por este sistema de identificación.

2.2.2. Requisitos de información

El sistema deberá guardar información a cerca de:

IR-01. Usuario: todos los usuarios tendrán un nombre de usuario, nombre completo, y dirección de email. Un usuario tiene al menos un rol en el protocolo de gobierno del diccionario de datos. Un usuario puede tener uno o dos roles diferentes. Un usuario puede ser programador en uno o varios módulos y puede ser supervisor de un módulo. Un usuario que es supervisor general solo tendrá un rol (supervisor general) debido a que los niveles de autorización de este usuario incluyen los otros.

IR-02. Módulo: un módulo tendrá un nombre que será único en el proyecto. Los símbolos definidos en el módulo y los usuarios autorizados en él, como programadores o supervisores de módulo, serán asociados al mismo.

IR-03. Símbolo: los símbolos son los elementos principales del diccionario de datos. Cada símbolo tendrá un nombre, el cuál será único en el proyecto, una explicación, las unidades y tipos de símbolo (su rol en el proyecto: "ProjectTypeOfValue" y el tipo de símbolo en el lenguaje en el que se programa el modelo: "ProgrammingLanguageSymbolType"). Cuando los símbolos no tienen unidades, serán indicados como carentes de dimensión.

IR-04. El tipo de símbolo en el proyecto indica el rol que juega cada símbolo: variable, variable auxiliar, serie histórica de datos, parámetro, constante, escenario de parámetro, variable desagregada o variable multidimensional, índice o caso de índice.

IR-05. Símbolos, índices y sus casos, al mismo tiempo que categorías, deben estar validadas y su estado debe reflejar si ya han sido validadas.

IR-06. Todos los símbolos tienen una categoría asociada a ellos. Una jerarquía de categorías de 2 niveles como máximo puede ser definida. En caso de la jerarquía categoría-subcategoría, el símbolo debe reflejar la subcategoría a la que pertenece.

IR-07. Cuando los símbolos son multidimensionales (variables desagregadas) o series históricas, tendrán uno o más índices asociados a ellos.

IR-08. Cada índice tiene asociados diferentes valores o casos que el propio índice representa. Por ejemplo, "REGIONS_I" sería un índice, y sus valores o casos serían "ASIA, USA, EU, ...".

IR-09. Para completar la definición del vocabulario controlado del proyecto el diccionario de datos almacenará acrónimos (letras y significado) tales como "EROI": "Energy Return on Investment", y adjetivos (la palabra y su forma de uso en el proyecto), por ejemplo "Consumed": "Use of a flow of materials or energy with consequence of disposal or dissipation and release of heat. In the model it is used for the consumption of an available amount of materials or energy (valid for both primary and final). For primary resources the adjective land is used for a similar use.". Estos acrónimos y adjetivos podrán ser usados en los nombres de símbolos.

IR-10. Para una mejor explicación de las restricciones del dominio del proyecto el sistema guardará reglas semánticas (un nombre para la regla y la explicación). Por ejemplo, "conservation of energy": "the total energy of an isolated system remains constant; it is said to be conserved over time. This law means that energy can neither be created nor destroyed; rather, it can only be transformed or transferred from one form to another." o una expresión matemática.

IR-11. Los módulos del proyecto son programados usando algún lenguaje de simulación. El diccionario de datos guarda los tipos de símbolos que pueden ser usados en el lenguaje de programación. Vensim es el lenguaje usado en el proyecto Locomotion.

IR-12. El diccionario de datos guarda una relación entre los tipos de símbolos en el proyecto y los tipos de símbolos en el lenguaje de programación. Por ejemplo, teniendo Vensim

como lenguaje de programación, un índice esta definido a través de un "subscript", una serie de datos histórica puede ser definida con una "lookup table" o una "subscripted variable", un parámetro puede ser definido como un "switch" (símbolo con, por ejemplo, 0, 1, 2, 3, como posibles valores que son usados en funciones "IF_THEN_ELSE(..., IF_THEN_ELSE(...), IF_THEN_ELSE(...))".

2.2.3. Requisitos no funcionales como historias de usuario

NFR-01. Como líder de proyecto quiero que la interfaz del diccionario de datos sea accesible vía una app web para facilitar tanto el acceso, como el mantenimiento (principalmente actualización).

NFR-02. Como líder de proyecto quiero que la interfaz de la app web tenga un diseño "responsive" para que se adapte bien a las diferentes pantallas de los dispositivos y sus tamaños.

NFR-03. Como líder de proyecto quiero que los usuarios con roles en el proyecto deban mostrar sus credenciales para llevar a cabo las tareas asociadas con sus roles para garantizar la seguridad de la información contenida en el diccionario de datos.

NFR-04. Como líder de proyecto quiero que los usuarios no puedan ejecutar otras tareas para las cuales no están autorizados para que el protocolo de definición y validación de símbolos en el proyecto sea cumplido.

NFR-05. Como líder de proyecto quiero hacer accesible las funcionalidades del diccionario de datos a través de una API RESTful, para facilitar la automatización de tareas.

NFR-06. Como equipo de aseguramiento de la calidad queremos que la funcionalidad de añadir símbolos vía la API RESTful sea una prioridad. Debería estar disponible al mismo tiempo que el servicio de diccionario de datos esté ejecutándose para que esta información pueda ser inyectada de forma automática desde programas Vensim ya existentes.

NFR-07. Como equipo de aseguramiento de calidad queremos tener disponible un servicio de API RESTful que, dada una lista de nombres de símbolos, devuelva (con formato JSON pendiente de definición precisa) toda la información concerniente a los símbolos nombrados, si ya están guardados y validados en el diccionario de datos para que los programas Vensim puedan ser automáticamente revisados.

NFR-08. Como líder de proyecto quiero tener un sistema de autenticación basada en tokens entre servicios para que la seguridad de la API RESTful sea asegurada y por lo tanto la integridad del diccionario de datos.

NFR-09. Como líder de proyecto quiero que ambos, usuarios registrados y usuarios externos no registrados en el proyecto, puedan filtrar la información del diccionario de datos o bien por coincidencia exacta o por similitud (basado en "substrings") para mejorar la usabilidad cuando se desea localizar un símbolo con conjunto de símbolos.

NFR-10. Como líder de proyecto quiero que el sistema esté disponible al menos 340-350 días al año, 24 horas al día, para que su uso por usuarios con diferentes zonas horarias sea facilitado, pero al mismo tiempo permitir la realización de tareas de mantenimiento que requieran parar los servicios el sistema.

NFR-11. Como líder de proyecto quiero que el sistema entero (código fuente, interfaces, notificaciones, mensajes y la API RESTful) esté escrito en Inglés para facilitar la comunicación entre los investigadores y en general otras personas interesadas en un entorno internacional y europeo, en particular.

Este conjunto de requisitos se basa en una especificación original realizada por Yania Crespo para el Deliverable 9.1 [6].

Capítulo 3

Plan de proyecto

3.1. Scrum

Utilizaremos Scrum como marco de trabajo para desarrollo de software ágil. En Scrum se definen eventos y roles. En este caso el Scrum Master, rol encargado de dirigir el desarrollo del proyecto, es la tutora del mismo, es decir, Yania Crespo. Por otro lado el rol de desarrollador, o mejor dicho de equipo, le corresponde al alumno David Gómez. La tutora también actuará como product owner, haciendo de intermediario con los diferentes stakeholders del proyecto. En cuanto a los eventos que se definen en el marco de Scrum encontramos los sprints, es decir, periodos de trabajo cuya característica principal en Scrum es su duración fija a lo largo del proyecto. Los sprints, comúnmente, duran entre una y cuatro semanas. En este proyecto la duración del sprint se define en dos semanas. Para cada sprint se determina la carga de trabajo antes de comenzar y se evalúa lo realizad al terminar. Para ello se definen una serie de eventos o reuniones.

Para cada sprint se realizará un sprint review & retrospective en forma de reunión, para, como se ha comentado anteriormente, analizar los resultados de cada tarea planificada hecha, incompleta o que quede por abordar. En periodos de dos semanas y coincidiendo con las reuniones de sprint review, realizaremos el sprint planning, es decir, planificaremos las siguientes dos semanas de trabajo en base a los resultados observados en los sprint reviews. Las reuniones diarias habituales en Scrum se adaptan al contexto definiendo reuniones semanales. Por tanto, por cada sprint habrá 3 reuniones, planning, weekly y review & retrospective.

Se intentará que el número de sprints sea el menor posible intentando planificar el trabajo de la forma más conveniente para ajustar el contenido de los sprints al trabajo requerido, contando con un exceso de trabajo en caso de que todo vaya bien y se pueda abordar, o en caso contrario pasarlo al siguiente sprint.

En general se planificará en base a 30 horas o más por sprint y se irán añadiendo retrasos a medida que avance el tiempo, al mismo tiempo, se define un sprint inicial (sprint 0) que

engloba el proceso de investigación y pruebas realizado para su desarrollo. Se pueden resumir los datos en los siguientes puntos:

- 40 horas de media por sprint repartidas en (30 horas + 10 horas extras estimadas por retrasos durante el desarrollo).
- 10 sprints.
- Un sprint 0 inicial de preparación del proyecto.

En cuanto a las tareas planificadas a la hora de abordar el desarrollo del proyecto, entra en juego un nuevo concepto relacionado con Scrum: el "Product Backlog", este representa el conjunto de tareas que han de ser realizadas para que la conclusión del proyecto sea efectiva, y son, en resumidas cuentas, las actividades extraídas de los requisitos del sistema desglosadas en conceptos más básicos e inmediatos sobre los que se puede trabajar.

Dentro de un sprint planning son asignadas ciertas tareas del "Product Backlog" a nuestro "Sprint Backlog", este no es ni más ni menos que un subconjunto de tareas extraídas del "Product Backlog" que son asignadas para su consecución próxima en la duración del sprint. El "Sprint Backlog" será definido a fondo en la sección de seguimiento de la memoria.

3.1.1. Calendarización

Siendo la fecha de inicio del proyecto, el día 1 de Noviembre de 2019, se van a realizar las previsiones necesarias para estimar la duración del proyecto, al mismo tiempo que se tendrá en cuenta que durante un periodo de 2 semanas de duración, se tendrá que reducir la actividad de este proyecto debido a la necesidad de aprobar un examen pendiente por el integrante del equipo David Gómez, al mismo tiempo que coinciden las vacaciones de Navidad.

En cuanto a la fecha de finalización prevista, se prevé que alrededor de finales de junio el proyecto esté cerca de su conclusión. Numéricamente, se estiman unas 500 horas, mucho más que las 300 horas previstas asignadas al período inicial de la beca (sin prórroga) o al desarrollo de un TFG. Adicionalmente, se trabajará como parte de las prácticas en empresa sobre el TFG hasta el 6 de Marzo, por lo que se intentarán cubrir las necesidades previas del proyecto, pertenecientes a los sprints 0 y 1. De esta forma se agilizará el comienzo del mismo y se podrá tener un contacto más frecuente entre los participantes en el proyecto. En la Tabla 3.1 se muestra la definición de los sprints previstos.

3.2. Análisis de riesgos

Como es lógico todo trabajo tiene su riesgo, y debemos planificar la gestión potencial de los mismos en caso de darse en algún momento puntual durante su desarrollo, para no ver

Calendarización		
Sprint	**Fecha de inicio**	**Fecha de fin**
Sprint 0	1 de Noviembre de 2019	26 de Febrero de 2020
Sprint 1	26 de Febrero de 2020	11 de Marzo de 2020
Sprint 2	11 de Marzo de 2020	25 de Marzo de 2020
Sprint 3	25 de Marzo de 2020	8 de Abril de 2020
Sprint 4	8 de Abril de 2020	22 de Abril de 2020
Sprint 5	22 de Abril de 2020	6 de Mayo de 2020
Sprint 6	6 de Mayo de 2020	20 de Mayo de 2020
Sprint 7	20 de Mayo de 2020	3 de Junio de 2020
Sprint 8	3 de Junio de 2020	17 de Junio de 2020
Sprint 9	17 de Junio de 2020	1 de Julio de 2020
Sprint 10	1 de Julio de 2020	15 de Julio de 2020

Tabla 3.1: Calendarización.

retrasado su progreso.

Actualmente existen diversas clasificaciones de riesgos[4], aunque en este caso se seleccionará la clasificación más básica aceptada de los mismos[24]:

- Riesgos de proyecto: Ponen en peligro al plan de proyecto. Afectan al esfuerzo y coste de producción que aumentarán considerablemente.
- Riesgos técnicos: Ponen en peligro la calidad resultante. Afectan a la complejidad estructural del sistema que puede llegar a escalar excesivamente.
- Riesgos de negocio: Ponen en peligro la realización del proyecto. En general se refieren a aspectos administrativos o económicos. Al ser un Proyecto de Fin de Grado no los tendremos en cuenta.

Para actuar contra los riesgos y minimizar su impacto existen dos aproximaciones estratégicas:

- Estrategia reactiva, esperar a que el riesgo sea real para afrontarlo, se analizan y reserva presupuesto previamente.
- Estrategia proactiva, analizar los riesgos antes de empezar, se miden las posibilidades de su ocurrencia, se priorizan y planifica entorno a su prevención y contingencia.

Se realizará un estudio proactivo de riesgos. Se han reservado 10 horas de trabajo por sprint para paliar posibles retrasos, mal funcionamiento del software y rediseños de interfaz.

Como se ha indicado anteriormente, los riesgos tienen cierta probabilidad de suceder, y por lo tanto es necesario medir el impacto de los más probables en el proyecto, al mismo tiempo que un plan para evitar el peor de los casos, la cancelación de este [27].

Impacto	Probabilidad				
	Improbable	**Poco probable**	**Posible**	**Muy probable**	**Casi seguro**
Mínimo	Bajo	Bajo	Bajo	Medio	Medio
Menor	Bajo	Bajo	Medio	Medio	Medio
Moderado	Medio	Medio	Medio	Alto	Alto
Mayor	Medio	Medio	Alto	Alto	Muy alto
Máximo	Medio	Alto	Alto	Muy alto	Muy alto

Tabla 3.2: Matriz de riesgos.

De la misma forma que existen los riesgos, existen ciertas técnicas de respuesta para minimizar sus impactos [29]:

- Evasión del riesgo
- Atenuación del riesgo
- Transferencia del riesgo
- Aceptación pasiva, no hacer nada, asumir el riesgo
- Aceptación activa, prever contingencias y hacer reservas

Generalmente, se intentará evitar cualquier tipo de riesgo, pero en caso de que sucedan, se intentará actuar de forma atenuante ante las consecuencias inevitables que conlleva cada riesgo y en caso de ser permanente, se intentará prever el transcurso futuro del proyecto en base a las medidas previstas para el mismo.

3.2.1. Gestión de riesgos

Tal y como se ha comentado anteriormente, vamos a realizar un enfoque proactivo de riesgos, para ello se emplearán tablas que describan el riesgo, sus características e impacto, y un plan de acción como se muestra en las Tablas de la 3.3 a la 3.9.

Riesgo 01		
Nombre: Retrasos en la planificación		
Tipo: Riesgo de proyecto	Probabilidad: Muy probable	Impacto: Mayor
Descripción: Retrasos en la planificación debido a la estimación incorrecta de las actividades a realizar		
Plan de acción: Se realizará una nueva planificación aumentando las horas dedicadas para la realización de las actividades que se han visto retrasadas, y llegar a alcanzar la planificación original.		

Tabla 3.3: Riesgo-01.

Riesgo 02		
Nombre: Baja médica del equipo de desarrollo		
Tipo: Riesgo de proyecto	Probabilidad: Poco probable	Impacto: Mayor
Descripción: Debido a que el equipo de desarrollo está formado por una persona, la baja médica llevaría al retraso del proyecto		
Plan de acción: Replanificar las actividades retrasadas. Aumentar la prioridad de las actividades afectadas para recuperarlas.		

Tabla 3.4: Riesgo-02.

Riesgo 03		
Nombre: Problemas con software de terceros		
Tipo: Riesgo de proyecto	Probabilidad: Posible	Impacto: Moderado
Descripción: Debido a que el proyecto está compuesto por múltiples elementos software, estos pueden llegar a no ser compatibles o a presentar ciertos errores.		
Plan de acción: Emplear software probado y consultar los manuales oficiales y foros de confianza para resolver los problemas. Realizar un trabajo de preparación en el sprint 0.		

Tabla 3.5: Riesgo-03.

Riesgo 04		
Nombre: Incertidumbre técnica y tecnologías desconocidas		
Tipo: Riesgo técnico	Probabilidad: Casi seguro	Impacto: Mayor
Descripción: El proyecto está compuesto por múltiples elementos software que no se han utilizado con anterioridad por el equipo de desarrollo.		
Plan de acción: Emplear software conocido en medida de lo posible y consultar los manuales oficiales y foros de confianza para resolver los problemas. Realizar un trabajo de preparación en el sprint 0.		

Tabla 3.6: Riesgo-04.

Riesgo 05		
Nombre: Falta o cambio de requisitos		
Tipo: Riesgo de proyecto	Probabilidad: Posible	Impacto: Moderado
Descripción: El proyecto se ha planificado sobre la marcha por lo que es posible que falten condiciones o se cambien por los socios.		
Plan de acción: Desacoplar software creado de forma que se puedan adaptar los cambios rápidamente.		

Tabla 3.7: Riesgo-05.

Riesgo 06		
Nombre: Incertidumbre arquitectónica		
Tipo: Riesgo técnico	Probabilidad: Posible	Impacto: Máximo
Descripción: El modelo arquitectónico de microservicios elegido puede no ser el óptimo y no existe una sola convención en su diseño.		
Plan de acción: Buscar la mejor versión arquitectónica posible o más consistente que se pueda encontrar y partir de ella. Realizar un trabajo de preparación en el sprint 0.		

Tabla 3.8: Riesgo-06.

Riesgo 07		
Nombre: Usabilidad de la interfaz		
Tipo: Riesgo técnico	Probabilidad: Casi seguro	Impacto: Menor
Descripción: La interfaz del proyecto puede no ser óptima para representar toda la información o acceder a las acciones presentadas.		
Plan de acción: Emplear una biblioteca de estilos para la página y llevar a cabo demostraciones periódicas en los sprint reviews.		

Tabla 3.9: Riesgo-07.

3.3. Presupuesto

Esta sección representa el coste simulado y real del proyecto en caso de ser desarrollado en una empresa de producción de software. Finalmente se calcula el total simulado y más adelante se comparará este valor con el obtenido tras el desarrollo del proyecto. Ver **8.12.3**.

3.3.1. Presupuesto simulado

Este sería el coste previsto para una empresa de desarrollo software.

Coste material

Para el desarrollo del proyecto se planean utilizar los siguientes elementos clave:

- Ordenador portátil para el desarrollo del código, coste 800 €.
- Un total de 3 máquinas virtuales con una capacidad de 10 GB de almacenamiento en total para usarlas de nodos de despliegue, cuyo coste es de 0,009 €, la hora por máquina[17]. 13,5 Euros en total.
- Kubernetes como plataforma de despliegue de los servicios desarrollados, 0 €.
- GitLab como plataforma de almacenamiento del código del proyecto, 0 €

El coste total del Hardware y Software es de 813,5 Euros ya que el software empleado es libre y el Hardware es alquilado en caso de los servidores de máquinas virtuales y comprado en caso del portátil.

Coste laboral

Para la búsqueda del coste laboral, recursos humanos, se ha consultado la página `es.indeed.com` a fecha de julio de 2020.

El coste de recursos humanos lo tendremos en cuenta en base a los salarios medios de un desarrollador de software en España, tomando como referencia el salario medio anual (27.147€), y una jornada completa (8 horas), obtenemos que cobran 9'3€ la hora.

Por lo tanto, contando con la previsión de 500 horas en la realización del proyecto, el coste inicial estimado sería de: 500 horas x 9,3€/hora = 4.650€.

Presupuesto total previsto

La beca es un pago que sustituye al sueldo de un profesional, por lo tanto, se omitirá su valor y se pondrá el del sueldo simulado debido a que esto es una previsión en base al coste que tendría el proyecto en una empresa de desarrollo software.

Por lo tanto, el coste total sería de 813,5€+ 4.650€, siendo el primero el coste material simulado y el segundo el coste de recursos humanos simulado.

Ascendería finalmente a 5.463,5€.

3.3.2. Presupuesto real

Este sería el coste que realmente se prevé para el proyecto, siendo este desarrollado como trabajo de fin de grado.

Coste material

En total, las herramientas utilizadas serían las siguientes:

- 1 ordenador portátil.
- 3 máquinas virtuales.
- 1 servicio de despliegue distribuido Kubernetes.
- 1 servicio de almacenamiento de versiones.
- 12 herramientas software libres, sin contar GitLab y Kubernetes.

Todo ello supone un coste de 0€, ya que en realidad las máquinas virtuales las proporciona la universidad y se usará el portátil personal del alumno.

Costes de la beca

De forma complementaria al desarrollo del proyecto, la realización del mismo presenta una beca para el alumno desarrollador. Esta beca consta de pagos mensuales de 300 €, y una duración de 6 meses prorrogables.

Por lo tanto, su coste añadido previsto es de 1.800 €, sobre el total.

Sin embargo, el proyecto está planificado para acabar en julio, por lo que en caso de prorrogarse la beca, esto supondría un coste de 2.700€.

Capítulo 4

Herramientas y tecnologías utilizadas

Actualmente hay muchas tecnologías y herramientas disponibles en el mercado, esto presenta tanto ventajas como desventajas a la hora de abordar el desarrollo de un proyecto software.

Empezando por las fortalezas de la amplia gama disponible de tecnologías, en primer lugar, debe ser mencionado que aunque el proyecto estaba pensado para ser realizado con Python como lenguaje principal de programación, podría ser sustituido por otro lenguaje. Como ejemplo, podríamos poner Node js, basado en javascript. De todas formas, un lenguaje base tenía que ser adoptado para poder enfocar la búsqueda de herramientas al mismo y que coincidan con los intereses y funcionalidad deseadas.

Cada lenguaje y proyecto posee sus propias particularidades. Por ello, no todas las herramientas o tecnologías son compatibles entre sí. Sin embargo, contamos con un gran número de posibilidades a la hora de elegir, cada una con un enfoque diferente de la anterior que puede diferir ligeramente o drásticamente.

Por lo tanto, y más concretamente hablando sobre Python, es un lenguaje con mucha flexibilidad a la hora de elegir herramientas que ayuden a desarrollar con él. Gracias a su extendido uso a lo largo de toda la comunidad, es fácil de aprender gracias a la gran cantidad de ejemplos disponibles.

En el lado contrario, el hecho de que haya tanta diversidad de herramientas y versiones dentro de cada una, dificulta enormemente la elección de las mismas. Esto es así, debido a que no siempre son compatibles entre ellas e incluso es posible que las diferentes versiones generen errores imprevistos o dependencias externas a mayores de las necesarias. Evidentemente no podemos elegir herramientas que reporten errores frecuentes o mal-funcionen en ciertas versiones. Por ello, las hemos elegido por su utilidad, consistencia y familiaridad, debido a que ya habíamos trabajado con varias de ellas.

Concluyendo el balance de rasgos anteriormente mencionado, en base a la variedad de herramientas y versionado, es positivo en la gran mayoría de casos exceptuando quizás algunos lenguajes o herramientas más específicas o de nicho. Por ejemplo el caso de BPEL[9] fue considerado como lenguaje de programación para llevar a cabo la orquestación de los microservicios, pero debido al poco uso que tenía y a que su utilidad era fácilmente sustituible por un despliegue en Kubernetes, no ha llegado a ser utilizado.

4.1. Herramientas de organización

4.1.1. Rocket

Canal de comunicación seleccionado para realizar consultas en torno a las actividades del proyecto. Principalmente es una vía rápida de mensajería a través de chats para poder obtener respuestas de forma ágil.

Esta es la herramienta seleccionada debido a que ambos integrantes del grupo la conocíamos y aprovechamos así el servidor que ya tiene disponible la propia Escuela de Ingeniería Informática. La interfaz es parecida a la de WhatsApp o Telegram en sus versiones web pero algo más agradable a la vista.

4.1.2. Skype

En principio no iba a ser utilizada esta herramienta, pero finalmente tras la declaración del estado de alarma y sucesiva cuarentena por culpa de la propagación de una pandemia sin precedentes, CoVid-19, se tuvo que plantear un sistema para comunicarse por voz y llevar a cabo las reuniones semanales que plantean los diferentes sprints del proyecto.

Skype es el canal de comunicación seleccionado con este propósito, como método de reunión virtual en la etapa CoVid-19, también empleado para realizar demostraciones en tiempo real y corregir aspectos de la interfaz pendientes o plantear nuevos. Ha sido una herramienta fundamental para el desarrollo del proyecto durante dicha cuarentena y eficaz ante el distanciamiento social y confinamiento.

4.1.3. Overleaf

Overleaf es la herramienta elegida para tanto redactar la documentación, como anotar las actividades que se van realizando y planificando, y que a continuación se apuntan en los tableros de GitLab convertidas ya en incidencias (*issues*).

Esto último se hace en cada reunión semanal, y la documentación, por otro lado es escrita de forma periódica una vez se obtiene la información necesaria para las diferentes secciones o se realizan las tareas que pretende cubrir.

Esta herramienta de escritura está basada en LaTeX, y permite ver, una vez compilado, el resultado de las diferentes versiones de la documentación que se van realizando a medida que es escrita.

4.1.4. GitLab

En todo proyecto software, el versionado es uno de los aspectos más relevantes a la hora de desarrollar el producto en cuestión. Por ello, git, en este caso GitLab, que está basado en él, ofrece un mecanismo de control de versiones de archivos basado en ramificaciones del proyecto y mediante comandos propios, se puede gestionar de manera muy conveniente para llevarlo a cabo con éxito.

Para empezar, definimos una licencia en el repositorio, que debe ser público al igual que el proyecto en sí mismo. De entre las licencias revisadas y consideradas, podemos destacar la licencia MIT, enfocada a ser de carácter débil, más permisiva, bajo esta licencia el proyecto no tendría copyright, pero no está recomendada su uso para proyectos en términos generales, incluso bajo las premisas de software de código abierto.

Algo contrapuestas, las licencias GPL, siguen siendo licencias de código abierto, pero en este caso están enfocadas a proteger a los usuarios finales del software. Como consecuencia de esto, el software generado bajo este tipo de licencias tiene que perpetuar la propia licencia aplicada al código original. En nuestro caso, la versión seleccionada es la 3: GPLv3, debido a que es la más actual, recomendada y la que mejor encaja con las características del proyecto.

Aunque no se haya empleado realmente el desarrollo basado en DevOps, se planteó en un principio hacerlo a través de las pipelines que proporciona GitLab, en estas pipelines se llevaría a cabo del proceso de CI/CD (Integración Continua / Despliegue Continuo). En otras palabras, al actualizar nuestro repositorio GitLab, analizaremos el código producido, y en caso de ser apto para ser usado de forma inmediata, esta nueva versión sería desplegada de forma automática.

Por último, se va a utilizar el seguimiento de incidencias mediante las funciones de tableros de GitLab, y se asignará la carga de sus tareas con el peso que tengan en desarrollo mediante un tipo de medición llamado “T-shirt sizing”. En otras palabras, se definirá la talla de las tareas a realizar.

4.2. Herramientas para la programación

4.2.1. Visual Studio Code

Para llevar a cabo el proceso de codificación del programa necesitamos establecer una herramienta que pueda soportar todos los lenguajes a ser posible y que sea cómoda y visualmente ligera, que no sature de información al programador mientras trabaja. Visual Studio Code es el IDE elegido, ya que a parte de cumplir lo anteriormente mencionado, posee un

sistema de plugins mediante el cuál podemos personalizar el entorno a nuestro gusto. Entre otras funcionalidades cabe destacar que posee una terminal linux integrada en el propio entorno y un sistema de visualización de carpetas para poder acceder a los archivos fuente de forma cómoda y estructurada.

4.2.2. Python

Python es el lenguaje de programación base elegido para el proyecto y en el que se debía desarrollar desde un principio. Como puntos a destacar de Python, está su flexibilidad, en cuanto a que es un lenguaje multiplataforma, se puede desarrollar tanto para Windows como para Linux u otros sistemas operativos, multiparadigma, se pueden implementar otros paradigmas diferentes al orientado a objetos aunque este sea su fortaleza, y cuyo tipado es dinámico, las variables se adaptan al valor que contienen a medida que evolucionan.

En Python será programada toda la lógica de negocio del sistema, es decir, el backend del mismo. Por lo tanto, los microservicios y la gateway, que son los componentes puramente destinados al procesamiento de las peticiones del cliente, son los que proporcionan la funcionalidad al sistema, serán los elementos programados en Python, y adicionalmente se usarán plugins específicos para este lenguaje para omitir las carencias del mismo. Más concretamente, se utilizará Pylint para enlazar el proyecto con Sonarqube para realizar los análisis de calidad del mismo, Flask, será utilizado para realizar las comunicaciones REST y con la base de datos MySQL, respectivamente, dependiendo del contexto del código.

4.2.3. MySQL

La base de datos principal del sistema donde se guardará la información del diccionario de datos, necesaria para la ejecución de los diferentes procesos alojados tanto en el backend como en el frontend. Se ha elegido MySQL porque ya era conocida por los integrantes del equipo y porque es la más familiar a los desarrollos anteriores realizados por los mismos.

En un principio se había pensado usar Phpmyadmin, una herramienta orientada a la visualización gráfica de las bases de datos para poder monitorizar de forma sencilla las propias instancias de MySQL que aloja nuestro sistema, pero no se ha llegado a utilizar debido a la sencillez de acceso a base de datos mediante consola.

4.2.4. Docker

Encapsulamiento de servicios basado en contenedores que se ejecutan sobre una máquina virtual linux. Docker es una tecnología reciente empleada para comunicar APIs de forma que no importe el sistema en el que finalmente se acaben ejecutando. Es decir, aislamos las APIs para que puedan interaccionar entre sí como si estuvieran comunicándose en el mismo host, pudiendo desplegarlas en otros. Esta tecnología permite el escalado de servicios

al desplegar estos contenedores en Kubernetes, creando así clusters de contenedores con nuestros microservicios en ellos.

Además, Docker nos permite levantar todos los servicios al mismo tiempo y respetando las dependencias entre ellos. En caso de necesitar ejecutar más de una instancia del mismo tipo de servicio, o bien de servicios diferentes en el mismo sistema, y darles un puerto para comunicarse con el exterior del propio contenedor.

4.2.5. Kubernetes

En cuanto al problema del escalado de servicios, Kubernetes soluciona todo este tema de forma relativamente fácil en comparación a la implementación manual de los mismos. Esto se debe a que Kubernetes posee un balanceador de carga interno que evita que los nodos en los que están desplegados nuestros microservicios se sobrecarguen, y gestiona de forma automática el proceso de orquestación de los mismos.

Además, Kkubernetes nos permite desplegar nuestros servicios en contenedores, como docker en este caso, y hacerlo compartiendo la misma dirección IP incluso estando desplegados en máquinas diferentes. Como punto extra, Kubernetes permite replicar los microservicios de forma individual, podemos elegir cuál replicar, y monitorizar el estado de nuestro cluster en todo momento.

Finalmente, Kubernetes permite gestionar el despliegue continuo de las nuevas versiones que vayamos desarrollando permitiendo la sustitución progresiva de las instancias desplegadas de la versión anterior por los de la nueva, al mismo tiempo, podremos revertir los despliegues de versiones nuevas a otras anteriores en caso de ser necesario.

Como contrapunto a todas estas características ventajosas, se debe indicar que éstas son algo complejas en conjunto para aprender a usarlas de forma eficiente, por lo que su aprendizaje es algo costoso y, además, requiere emplear recursos como máquinas diferentes para desplegar los servicios en distintos nodos.

4.2.6. REST

Para realizar los servicios de comunicación se estudió comparativamente GraphQL y REST.

Como se explica en [1], podríamos usar un modelo de subscripciones como implementa GraphQL, pero en ese caso deberíamos proveer de un mecanismo de subscripción y desubscripción y mantener el estado en el servidor. Al ser GraphQL muy nuevo y estar en evolución no elegimos esta aproximación para solucionar el problema, aunque se indica que técnicamente GraphQL es una mejor forma de implementar los servicios de comunicación que REST.

REST presenta inconvenientes sobretodo en torno a aplicaciones móviles, pero nuestro objetivo está enfocado en una aplicación para ser ejecutada de forma estática, es decir, en un ordenador con una conexión estable.

Por otro lado, se comenta que los endpoints pueden llegar a ser problemáticos en REST debido al aumento de información transmitida con las versiones posteriores o el aumento de usuarios, pero en nuestro caso, la carga de datos no debería ser excesivamente elevada en ningún momento por lo que no tendríamos inconvenientes en ese aspecto.

Pese a esto, REST es una mejor opción ya que nos provee de fiabilidad, escalabilidad y flexibilidad en el lado servidor, además de independencia tecnológica y separación con el cliente, cosa que aprovecharemos a la hora de usar JavaScript en el *frontend* y Python en el *backend*.

4.2.7. HTML5

En cuanto al frontend, en otras palabras, la parte visualizada por el usuario, usaremos HTML5 para darle forma a la estructura principal de la página web que mostraremos. Gracias a HTML podemos representar las peticiones que se realizarán al servidor de forma gráfica desde el punto de vista final que utilizará el usuario. En resumen, HTML5 es el esqueleto del cliente que se rellena con el uso complementario de JavaScript y Jquery, y se embellece gracias a la librería de estilo que se ha seleccionado de Bootstrap.

4.2.8. JavaScript y Jquery

Se carga en el cliente y se adhiere al servicio REST proporcionado por el servidor para crear la conexión entre el frontend y el backend. Representan los scripts de formateo de HTML5, son el código que crea de forma dinámica las interacciones del usuario web, crea las tablas, desactiva los botones y pestañas, carga la paginación y lo más importante, envía las peticiones mediante el uso de Ajax dentro de los scripts dedicados al formateo de las mismas.

4.2.9. CSS y Bootstrap

Una vez tenemos la estructura de nuestro frontend, peticiones a servidor, debemos hacer que sea agradable a la vista del usuario si es que queremos que sea usado de forma continua y eficaz. Además, una buena apariencia ayuda a la comprensión del usuario del propio servicio que se ofrece y a un aprendizaje más rápido.

En nuestro caso, mezclaremos el uso puntual de CSS puro con Bootstrap, que es una biblioteca de estilos compatible con el uso de CSS que nos proporciona modelos ya creados por la comunidad para no tener que emplear excesivo tiempo en ajustar posiciones o colores. De esta forma agregando clases a nuestro código HTML podremos aplicar nuestros modelos Bootstrap y a su vez hacer cambios manuales mediante el uso de CSS.

4.2.10. Nameko y RabbitMQ

Estas herramientas son la base fundamental del funcionamiento de los microservicios más allá de Docker y su despliegue o el lenguaje en el que estén programados.

Por su parte, Nameko proporciona el concepto de microservicio en sí mismo, permitiendo la declaración de clases que se definen como servicios y puntos de enlace de puertos web para llegar a ellos. Para ello necesitaremos hacer uso de otra de sus funcionalidades, las llamadas a procedimientos remotos, que son las que hacen uso de las funciones definidas dentro de cada "clase servicio".

Complementariamente, RabbitMQ es un servicio que proporciona Nameko para usarse como "broker" de mensajería, es decir, es un servicio encargado de hacer llegar las peticiones a los servicios y la respuesta al solicitante. Funciona mediante AMQP "Advanced Message Queuing Protocol", que traducido sería protocolo avanzado de colas de mensajes.

A lo largo de todo el proyecto se mencionará el uso de esta tecnología ya que es la base software sobre la que se apoya todo el sistema creado y su funcionamiento.

Capítulo 5

Desarrollo basado en microservicios

Tradicionalmente el desarrollo software se centraba en construir aplicaciones con un único servidor que prestaba todos los servicios requeridos por los clientes y que se podría describir como un bloque único. Se las clasifica como monolíticas, debido a su estructura indivisible que generalmente está compuesta por tres partes, servidor, base de datos y interfaz de usuario cliente [11].

Frente a la simpleza de una aplicación monolítica, siempre contando con las ventajas de dicha simpleza, se pueden distinguir varios problemas por los cuales se está optando por dejar atrás esta arquitectura. Entre ellos podemos destacar la escalabilidad y la complejidad que adquieren las reescrituras de código sobre sistemas mayores que dificultan su desarrollo futuro.

Es por estos problemas que principalmente se pueden centrar en la escalabilidad de los sistemas y desarrollo en definitiva con ojos en el futuro, por lo que se está apostando fuertemente dentro de la industria de desarrollo software por arquitecturas basadas en microservicios.

Dentro de las fortalezas de los microservicios encontramos la ya mencionada escalabilidad, la facilidad de comprensión del código debido a la separación de los servicios, y finalmente, la independencia de estos servicios entre sí. Si a todo esto le añadimos la flexibilidad tecnológica que esta separación supone y la agilidad añadida que surge del procesamiento dividido de la información, podemos concluir, en que el desarrollo basado en microservicios será y en cierto modo ya es un candidato crucial en el futuro de la industria.

Para finalizar, ha de comentarse que no todo son ventajas a la hora de elegir el desarrollo basado en microservicios, y que las desventajas que presenta son también importantes. Por un lado, la complejidad añadida de un sistema distribuido, el manejo de su estado es algo muy complejo y depende de conexiones que pueden llegar a fallar, esto añade múltiples factores de configuración de servicios y al mismo tiempo presenta una barrera importante a la hora de realizar las pruebas de calidad pertinentes.

5.1. Conflictos en el diseño de microservicios

El desarrollo de microservicios es algo relativamente nuevo y a menudo podemos encontrar conflictos al tratar de compararlos con los servicios monolíticos habituales. Por ello, debemos tomar ciertas decisiones a la hora de diseñarlos para poder mantener los modelos deseados, en términos de consistencia con el diseño del sistema, y poder mantener la claridad en cuanto a reparto de responsabilidades entre las diferentes clases que manejaremos.

En la misma línea, pero diferente campo conceptual, una vez el sistema se encuentra en ejecución, podemos encontrar ciertos problemas en torno al manejo de estos servicios de forma no bloqueante en caso de que su carga de trabajo sea demasiado grande como para manejarlo de forma síncrona con el cliente. En este caso, deberemos tomar de nuevo ciertas decisiones para mantener la consistencia en diseño del sistema sin perjudicar los intereses del cliente.

5.1.1. Exceso de trabajo manejado

Poniéndonos en la situación de que la ejecución de un microservicio requiera de un procesamiento demasiado extenso por parte del servidor tendremos que evitar bloquear al cliente. Por ello, la asincronía es el mejor modelo existente. Por un lado, nos permite liberar al cliente una vez ejecute la petición. Por otro lado, nos permite realizar el procesamiento del trabajo sin tener que elevar la eficiencia del mismo de forma excesiva, pudiendo así notificar al cliente una vez el trabajo haya sido completado y dispongamos de los resultados.

Como podemos ver en [1], los problemas principales de los servicios que dependen de sincronía es principalmente la existencia de un receptor que espere por la respuesta del propio servicio o que pregunte por su finalización sin timeouts o grandes retrasos entre consultas. Por otro lado, el problema principal de las conexiones paralelas es que requieren dos endpoints y a su vez la escalabilidad del sistema se hace más complicada. Crear un servicio que se mantenga a la espera en el cliente de que las operaciones del servidor se completen es, en parte, la mejor solución posible para evitar dejar colgado al cliente y, a su vez, no tener que realizar escalados complejos de endpoints.

Finalmente, la asincronía no ha sido necesaria debido a que las cargas probadas han sido menores, pero en un futuro desarrollo del sistema serían convenientes junto con un sistema que indique si ha habido cambios en la base de datos.

5.1.2. Pendiente de la respuesta de otro servicio

Un microservicio podría quedar a la espera de la respuesta de otro con tiempos de respuesta muy elevados. La comunicación con el cliente va a ser implementada de forma limpia, la coordinación de los microservicios va a ser llevada por la gateway del sistema. Al dejar al cliente libre, los microservicios dependiendo de los datos que necesiten para ejecutarse,

podrían ser puestos de forma paralela en ejecución, siempre y cuando se sincronicen al terminar sus tareas. En caso de depender la ejecución de un microservicio de los datos generados por otro, simplemente se quedará en espera de la respuesta del microservicio llamado. Podemos optar por esta solución gracias a la creación de métodos específicos para los servicios que sean ejecutados de forma rápida y no bloqueen excesivamente el uso de recursos.

5.2. Comunicación entre microservicios

Cuando nos centramos en los elementos que componen a nuestros microservicios, aparte de la funcionalidad independiente de cada uno de ellos presentada por la lógica de negocio, los métodos de comunicación implantados son una parte fundamental para describir su comportamiento y funcionalidad. Por ello, hemos decidido que la forma óptima de establecer sus comunicaciones esté separada en, por un lado, la comunicación con el cliente y, por otro lado, la comunicación interna de los microservicios.

Con la forma óptima de establecer estas comunicaciones nos referimos a la forma más clara, fiel al modelo y orientada a una correcta ejecución y comprensión del mismo por parte del cliente.

5.2.1. Comunicación con el cliente

El cliente será un cliente básico HTML5 con CSS y Boostrap, y será el que realice las peticiones al servidor mediante el uso del frontend. Para ello, emplearemos una comunicación basada en el modelo REST bajo HTTP. Todo esto será implementado mediante Flask, tanto las peticiones del cliente, como las respuestas del servidor. En este caso, especificaremos los métodos de las llamadas en base a CRUD, es decir “Create, Read, Update, Delete”.

En todo momento el cliente estará dirigiéndose a la gateway de nuestro servidor, que en otras palabras es el punto de entrada para todas las peticiones entrantes y la que responderá a las peticiones de los clientes. A su vez, la gateway es la encargada de redirigir el tráfico de entrada y traducirlo en llamadas a los diferentes microservicios desplegados en el sistema, que serán ejecutados de forma síncrona o asíncrona dependiendo de la carga que posean. Finalmente, los microservicios darán su respuesta a la gateway y esta retornará su resultado al cliente.

5.2.2. Comunicación interna

Los microservicios como tal estarán desplegados en el nodo servidor de forma independiente, por lo tanto es posible que necesiten datos de otros microservicios para llevar a cabo su tarea. Para ello, la gateway será la encargada de administrar esos datos entre microservicios. Por lo tanto, no estableceremos una comunicación directa punto a punto, si no una comunicación asistida por la gateway. De esta manera podemos lograr menos dependencias entre ellos y mejor escalado tanto en despliegues como en microservicios posteriores.

Por lo tanto, el modelo arquitectónico es, en este caso, mucho más importante para mantener la claridad de desarrollo y escalado de microservicios que la optimización pura de los mismos realizando llamadas dentro de los propios procesos. La única excepción que se llevará a cabo en cuanto a peticiones punto a punto se da en la comunicación puntual entre microservicios, aunque no se puede llegar a definir realmente como punto a punto, ya que se comunican usando el servicio de cola de mensajes que redirige la petición a cualquier nodo libre del microservicio o esperan a que se liberen.

5.2.3. Conclusión/Esquema general

Como conclusión, la comunicación entre los microservicios es uno de sus aspectos fundamentales ya que es la que mantiene la coherencia arquitectónica de estos. El pilar fundamental de la comunicación se encuentra en la gateway o puerta de enlace, y se ha empleado una cola de mensajes para realizar la comunicación entre contenedores**6.3.1**.

5.3. Construcción y composición

Como se había mencionado anteriormente, la aplicación consiste en un servidor RESTful que escucha las peticiones de los clientes, y las procesa mediante el uso de microservicios localizados en contenedores llamados "workers". Estos "workers.ª su vez son servicios invocados mediante llamadas sucesivas alojadas en una cola de mensajes en RabbitMQ, manejado a su vez por Nameko.

Todo esto no sería posible sin una conexión y un despliegue ordenado, proporcionados en nuestro caso por docker-compose. Esta herramienta enlaza los contenedores en los que desplegaremos los diferentes servicios que necesitamos mediante una red interna y, al mismo tiempo, levanta los propios contenedores y crea volúmenes en los que almacenar la información.

5.3.1. Docker-compose

Para la creación del mismo utilizaremos un YAML versión 3.4 que principalmente tendrá dos apartados en la misma línea: "services" y "volumes" que representan, por un lado, los servicios desplegados y, por otro lado, los puntos de montado para persistencia de datos de la base de datos utilizada.

A continuación se describen las etiquetas utilizadas y sus respectivos significados. Para encontrar los archivos mencionados, en el capítulo de implementación se discutirá la distribución de las carpetas **7.1.3** y en el manual de mantenimiento su contenido **A.2**.

Services

En nuestro caso tendremos servicios para la API, los workers, RabbitMQ y la base de datos MySQL .

Las etiquetas comunes utilizadas son las siguientes:

- image : etiqueta la imagen del contenedor.
- container_name : nombre del contenedor.
- restart: configurada con "always", para que siempre que se caiga algún servicio se reinicie.

Con la excepción del servicio Rabbit todos utilizan la etiqueta:

- build : dentro de esta nos encontramos con otras dos etiquetas:
  - context : ruta donde encontrar los archivos relativos a la imagen.
  - dockerfile : ruta donde encontrar el archivo de composición docker para el servicio, deriva de context como punto de partida.

De nuevo, excepto para los servicios de Rabbit y MySQL todos utilizan la etiqueta:

- command : comando de ejecución para lanzar el servicio.

La etiqueta exclusivamente utilizada por la API:

- links : enlaces de red interna entre contenedores de servicios.

Empleada por Rabbit, la API y la base de datos MySQL:

- environment : definición de variables de entorno, cada servicio tiene las suyas propias.
- ports : puertos sobre los que se despliega el servicio de forma concreta, mapeados con el formato (puerto externo:interno).

Como complemento a la anterior etiqueta, en la base de datos MySQL:

- expose : expone los puertos a la máquina host, de esta forma se pueden realizar conexiones a la base de datos desde el host.

Finalmente, tanto en la base de datos MySQL como en la API:

- volumes : explicados en la siguiente sección, mapean un volumen local con una carpeta en el interior del contenedor.

Como apunte adicional, la versión 8 de MySQL dio problemas cuando se empezó a probar la composición de los servicios. Por esa razón, finalmente, se está utilizando la versión 5.7 para la base de datos MySQL. En cuanto al resto de servicios, estamos utilizando su última versión por lo que no es necesario anotarlo de forma explícita en el archivo YAML, ya que docker descarga por defecto las últimas versiones del software que se le indica. En los requisitos, por contrario, necesitamos indicar las versiones de Flask, Nameko y el entorno virtual de Python, entre otros.

Volumes

El almacenamiento de datos persistente de la API debe ser soportado por la base de datos. Por ello tenemos, como método nativo en docker, los volúmenes.

En resumen, los volúmenes son puntos de montado que mapean el interior del contenedor con el host. Al conectar un servicio a un volumen, podemos obtener acceso a los datos del volumen que a su vez pueden estar en uso por otro servicio.

Esto es algo especialmente útil ya que en caso de caída del servicio de base de datos no tendríamos que volver a crearlo todo. Los datos no se borrarían, e incluso podemos hacer volcados (backups) de la base de datos para tenerlos de repuesto en caso de querer restaurar un estado previo. La localización de este archivo YAML se discute

Despliegue

El conjunto de servicios se levanta al mismo tiempo mediante el comando: “docker-compose up”.

En primera instancia se descargarán las imágenes y se instalarán los requisitos para que funcione todo correctamente, instalándose así en los drivers de los diferentes contenedores. Se iniciará la base de datos tomando como referencia nuestro esquema, veremos más adelante en la Sección **6.2** como está formado. Finalmente, la API se levantará cuando todo lo anterior esté operativo.

En posteriores despliegues, los volúmenes e imágenes ya estarán disponibles para ser desplegados de forma rápida. En caso de realizar algún cambio en el código del proyecto “docker-compose build” construirá de nuevo las imágenes de despliegue del sistema y ejecutando “docker-compose up” de nuevo volveríamos a tener el sistema operativo.

En caso de querer parar el sistema, “docker-compose down” terminará su ejecución, si deseamos borrar los volúmenes que se han generado con la opción “-v” se terminará la ejecución de los contenedores y se borrarán los volúmenes asociados.

Backups

Para hacer una copia de la base de datos en un momento concreto de la misma seguiremos los siguientes pasos (Notar que "password" es la contraseña de la base de datos de ejemplo):

Dump bases de datos a /db.sql, dentro del contenedor (RUTAS RELATIVAS A WINDOWS):

docker exec mysql sh -c 'exec mysqldump –all-databases -u root -p"password">/db.sql '

Copiar el archivo al host, asumimos que estamos ejecutando a partir de este comando desde la carpeta src:

docker cp mysql:/db.sql .\main\db-data\

Copiar el archivo de volcado de vuelta:

docker cp .\main\db-data\db.sql mysql:/

Restaurar db:

docker exec mysql sh -c 'exec mysql -u root -p"password"</db.sql'

Capítulo 6

Análisis y Diseño

En este Capítulo se mostrarán algunos modelos importantes fruto del análisis y el diseño del sistema. Se proporcionarán varios tipos de modelos diferentes, el modelo del dominio que representa la información que maneja el sistema, el modelo de diseño relacional de la base de datos que representa la información almacenada, los modelos arquitectónicos del sistema dados por el modelo lógico aplicando algunos patrones arquitectónicos, algunos modelos dinámicos de ejemplo que describen la interacción entre los elementos de la arquitectura lógica su funcionamiento, y el diseño del despliegue adoptado.

6.1. Modelo del dominio

El modelo del dominio representado en la Figura 6.1 define los diferentes objetos que maneja el sistema para llevar a cabo su cometido, se van a analizar cada uno de estos elementos en base a su significado dentro de la lógica de negocio propuesta y sus atributos asociados. Este modelo del dominio fue realizado con la tutora para el Deliverable 9.1 del proyecto LOCOMOTION [6].

User, un objeto usuario es el que encapsula los datos de un usuario del sistema, se manejará el nombre de usuario (username), nombre completo (full_name) y correo electrónico (email).

TypeOfUserRole, clasifica los diferentes roles que puede tener un usuario. Más adelante en el desarrollo del proyecto se decidió que los roles fueran los siguientes: líder de proyecto (Project Leader), supervisor general (General Supervisor), supervisor de módulo (Module Supervisor), programador de módulo (Module Programmer), equipo de calidad(QA Team) y finalmente el invitado(Guest).

Module, un objeto módulo es el que encapsula la información de identificación de los módulos, su nombre (name), sirve para agrupar los diferentes símbolos del sistema.

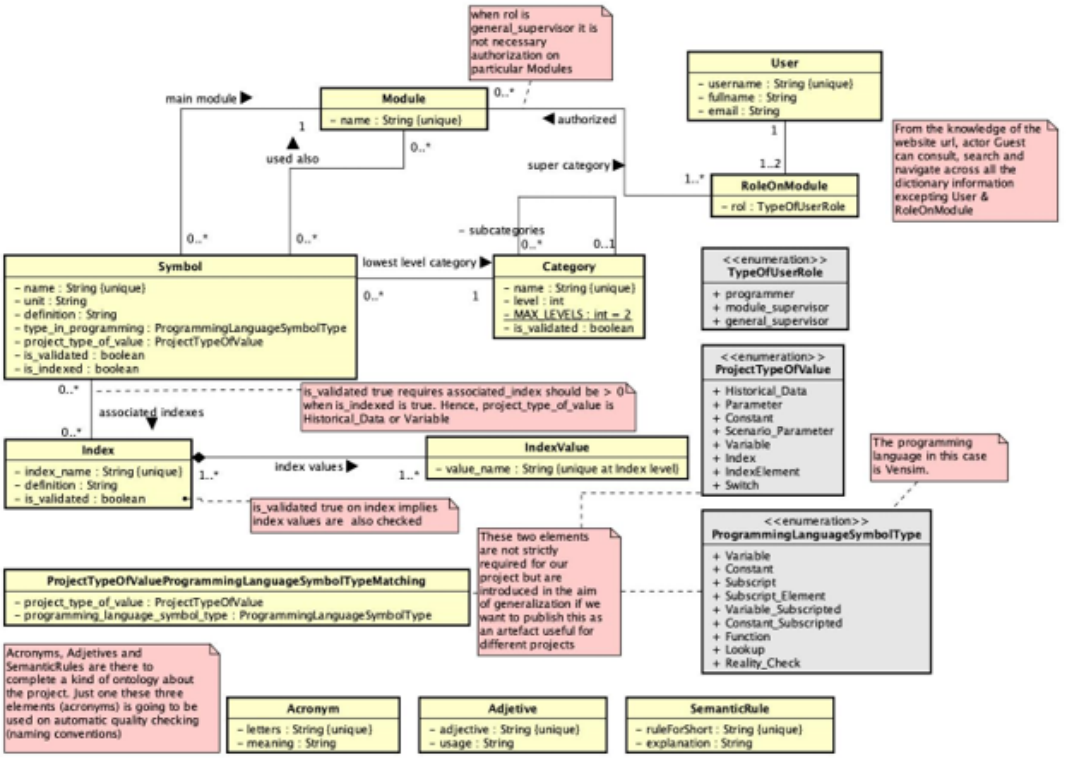

Figura 6.1: Modelo de Dominio.

RoleOnModule, la asociación que relaciona a los usuarios con un rol y un módulo, asigna el permiso que otorga el rol al que pertenece el usuario al módulo al que se le asocia.

Symbol, un objeto símbolo encapsula la información relativa al propio símbolo, está formado por un nombre (name), unidad (unit), definición (definition), y puede estar asociado a una categoría o subcategoría, a ninguno, uno o más índices, a un tipo de valor del proyecto (project_type_of_value) y un tipo de lenguaje de programación Vensim (type_in_programming). Se asocia a un módulo principal pero puede estar asociado a módulos secundarios adicionales. Adicionalmente, se comprueba su validez (is_validated) y su relación con los índices (is_indexed).

Category, un objeto categoría encapsula la información de una categría, su nombre (name), su nivel (level) que muestra si es una supercategoría o subcategoría, de ahí que el máximo de niveles de relación entre categorías sea de 2 (MAX_LEVELS), y su validez (is_validated). Una categoría puede tener asociados múltiples símbolos o ninguno.

Index, un objeto índice encapsula la información relativa a los propios índices a los que se asocian o no símbolos. Los índices están formados por sus valores. Se almacena el nombre del índice (index_name), su definición(definition), y si está validado(is_validated).

IndexValue, representa los objetos que son los valores de los índices. Estos pueden tener varios asociados al mismo tiempo y pueden existir valores repetidos pero sin estar vinculados al mismo índice. Se almacena el valor (value_name). Más adelante en el proyecto se han pasado a llamar "IndexElements", se ha sustituido el término valor por elemento.

ProjectTypeOfValue, clasifica los valores que representa el sistema a nivel de proyecto.

ProgrammingLanguageSymbolType, clasifica los valores de programación Vensim.

ProjectTypeOfValueProgrammingLanguageSymbolTypeMatching, relaciona los valores extraídos de Vensim y los asocia con los del proyecto.

Acronym, encapsula la información de un acrónimo, sus letras(letters), que lo representan y su significado(meaning) que lo describe.

Adjective, encapsula la información de un adjetivo(adjective), y se lo describe mediante su uso(usage).

SemanticRule, encapsula una regla semántica(ruleForShort) y su explicación(explanation).

Como conclusión, tenemos un sistema complejo que maneja bastante información, y por ello el diseño de la base de datos tiene que ser preciso y la separación entre las diferentes entidades al sistema tiene que estar bien definida. A continuación pasaremos al mencionado diseño de la base de datos.

6.2. Diseño de la base de datos

La Figura 6.2 muestra el diseño relacional de la base de datos que viene del análisis del modelo del dominio antes presentado. El diseño relacional que se muestra se basa en el diseño original realizado por María del Carmen Hernández para el Deliverable 9.1 del proyecto LOCOMOTION [6].

Como se puede ver a simple vista, hay mas tablas que clases en el modelo de dominio, que se utilizan para representar las relaciones anteriormente mencionadas en este.

Por un lado, las relaciones entre tablas son los usos de claves foráneas. En la misma línea, las tablas que tienen un "id" son las que han sido representadas anteriormente en el modelo de dominio y por lo tanto el resto son tablas de relación entre las anteriores.

En general, las relaciones son unilaterales en todo el modelo, pero la relación de símbolo con el rol del usuario(rayas discontinuas) en un módulo es algo especial ya que no se implican de forma directa, si no a través del permiso asociado a un módulo al que pertenece tal símbolo.

Cabe mencionar que la tabla "user_session_time" es algo especial ya que sirve para el almacenamiento de claves(tokens), asociarlas a un usuario y anotar su momento de creación(date), para calcular cuando expira dicha clave.

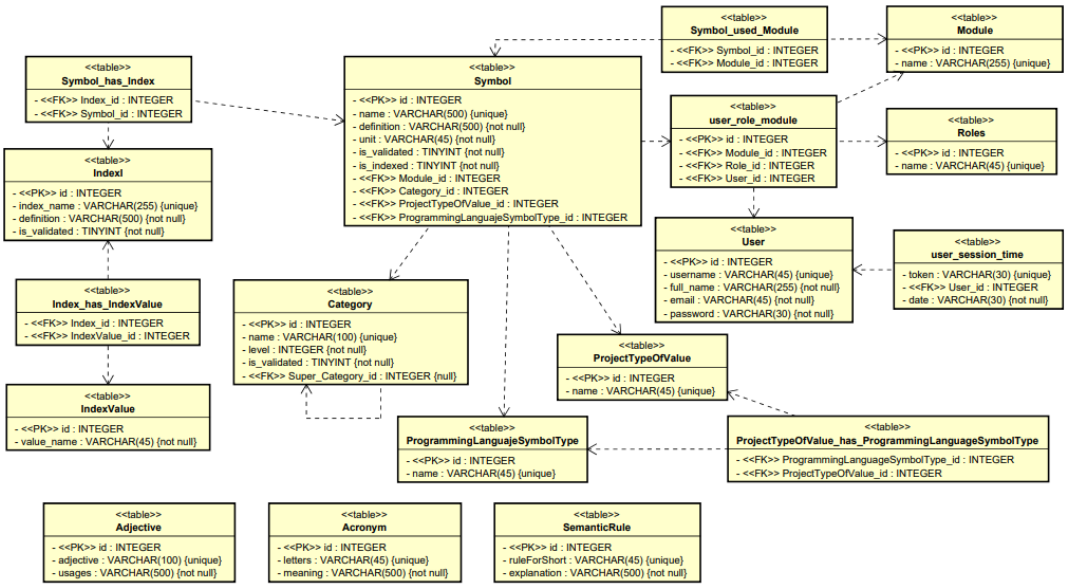

Figura 6.2: Diseño relacional de la Base de Datos.

El resto de datos sigue exactamente igual que en el modelo de dominio a excepción de los atributos extra que han sido añadidos en base a las previamente mencionadas relaciones entre tablas, y el nombrado de la tabla “Roles” que representa al enumerado “TypeOfUserRole”.

6.3. Arquitectura lógica de la aplicación

En esta sección se van a mostrar los diagramas de diseño lógico que forman el sistema. El diseño lógico general del mismo y los patrones seleccionados.

El modelo lógico del sistema está basado en la arquitectura cliente-servidor, alojando en el cliente la interfaz de usuario de la aplicación y en el servidor la capa de negocio y persistencia. Teniendo en cuenta que contamos con que la puerta de enlace y los microservicios representan la lógica de negocio y la capa de persistencia solo es accedida por los microservicios.

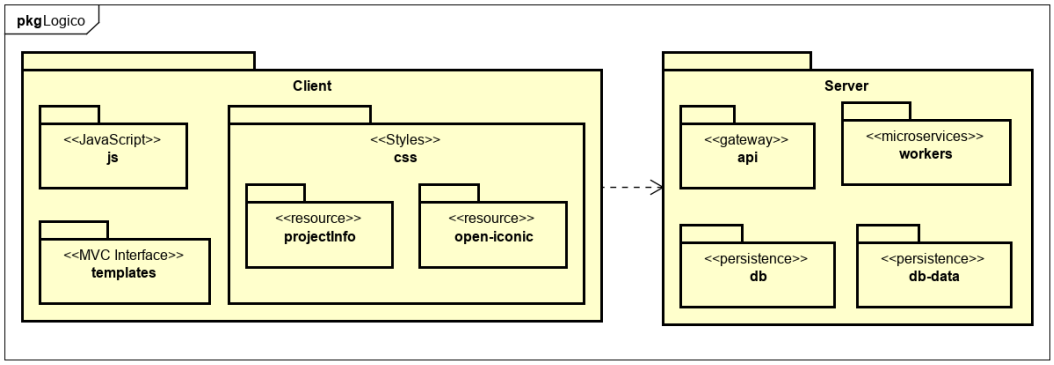

Figura 6.3: Diseño lógico cliente-servidor.

A continuación se van a comentar los patrones utilizados a lo largo del desarrollo del proyecto que han supuesto la organización del código que lo compone. Se mostrará un esquema en forma de diagrama y se discutirán las implicaciones de su aplicación.

Podemos destacar tres patrones aplicados a lo largo del proyecto:

- Gateway, estructura de microservicios
- DAO, objeto de acceso a datos
- MVC, modelo vista controlador

6.3.1. Gateway

El patrón elegido para el diseño del servidor es el patrón gateway, puerta de enlace en español, este patrón es similar al patrón tradicional fachada, tanto en uso como en diseño.

Su objetivo es aislar el backend del frontend que componen al sistema. Es decir, unifica el acceso de los clientes mediante el uso de, en este caso, el protocolo REST, pero podría ser cualquier otro tipo de protocolo. La ventaja que presenta este sistema es que los microservicios no necesitan conocer el protocolo empleado en la comunicación cliente-servidor.

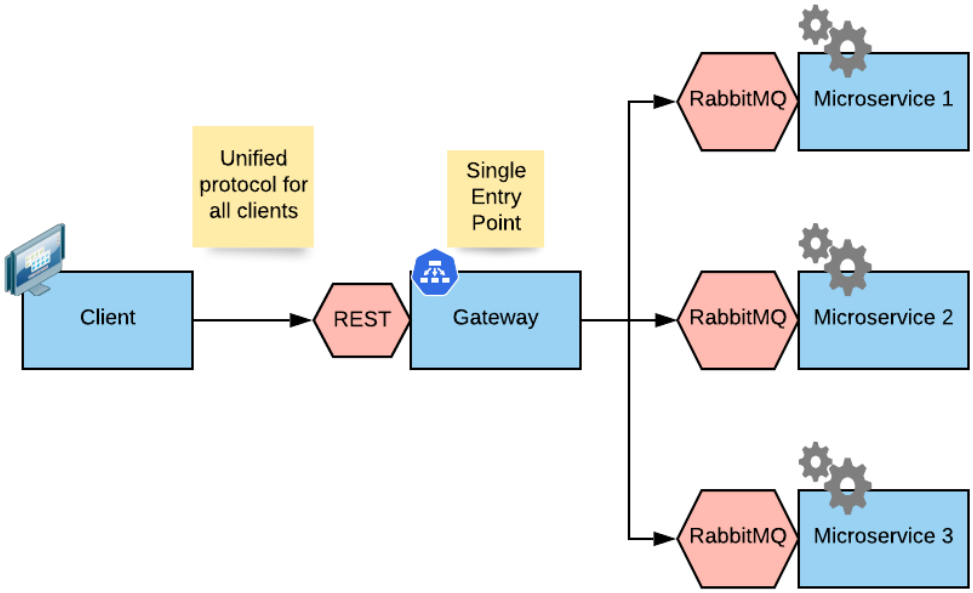

Figura 6.4: Patrón puerta de enlace. Creado con Lucidchart [16].

Dado que la única manera de acceder a la funcionalidad que presentan los "workers" es mediante la propia puerta de enlace, se consigue controlar el flujo de carga que representa cada petición sin perder tiempo de trabajo en los microservicios para procesar dicha petición, los simplifica. Por lo tanto, la funcionalidad principal que plantea la gateway es la de balancear la carga de los microservicios enviando las peticiones mediante una cola de mensajes. En este caso la cola es creada en RabbitMQ y, una vez obtenidos los resultados de la ejecución de los microservicios, formar una respuesta para el cliente [2].

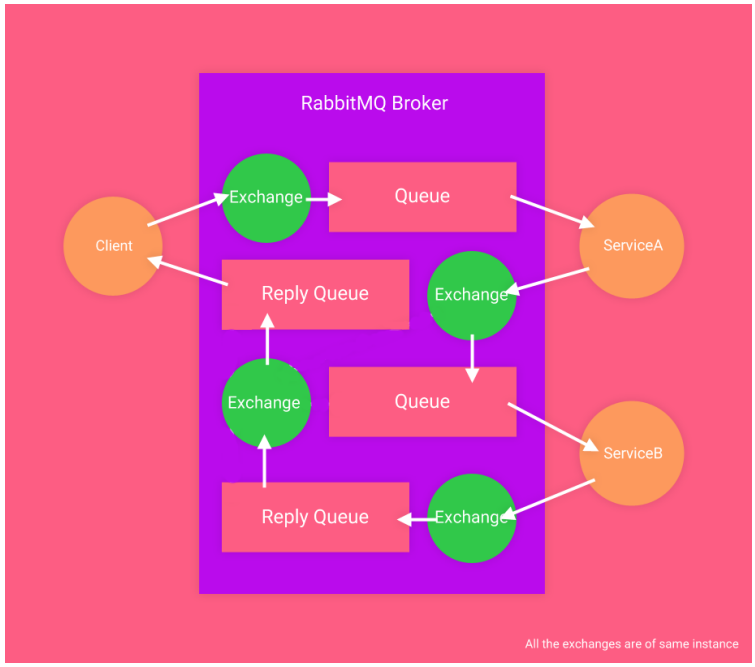

Figura 6.5: Patrón Cola de mensajes RabbitMQ. Imagen tomada de [2]

6.3.2. DAO, Objeto de Acceso a Datos

El objetivo principal del patrón objeto de acceso a datos (DAO), es el de aislar la lógica de negocio, en este caso localizada en los microservicios, y que esta capa de lógica de negocio sea la que use un objeto creado especialmente para la interacción con el almacenamiento persistente [3].

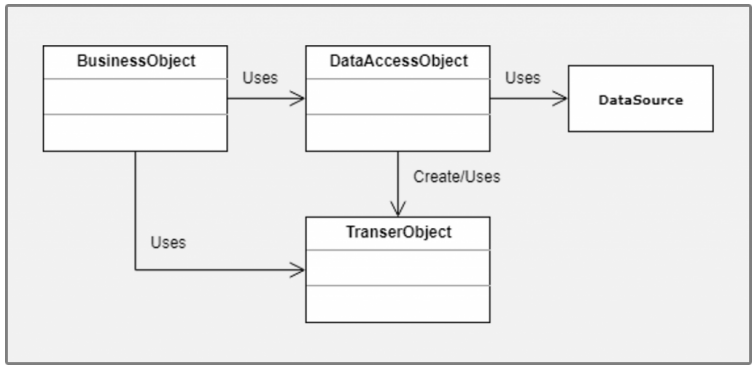

Figura 6.6: Patrón Objeto de Acceso a Datos. Imagen tomada de [3]

Como podemos ver en el diagrama de la Figura 6.6, el propio objeto de lógica de negocio es el que usa el objeto de acceso a datos, este crea una instancia que será la que actúe sobre la fuente de datos (ver Figura 6.7).

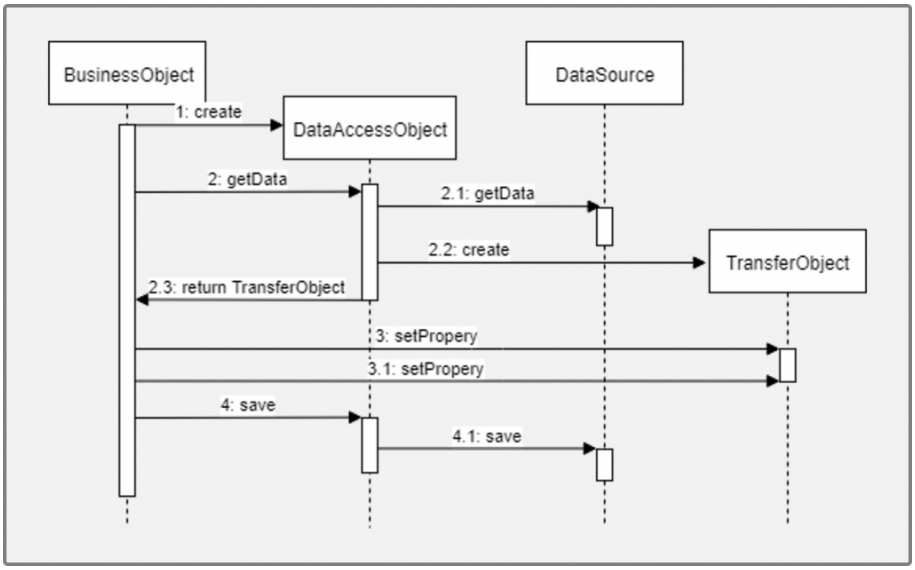

Figura 6.7: Diagrama de secuencia DAO. Imagen tomada de [3]

En nuestro caso particular, será la clase DbHelper la que nos proporcione la instancia de conexión con la base de datos MySQL, y cada microservicio tendrá asociado un DAO que será el encargado de crear las consultas SQL y solicitar la ejecución al DbHelper.

Los objetos de transferencia de datos serán cadenas json.

Este patrón es importante ya que ayuda a que la capa de persistencia del sistema sea mucho más legible y quede aislada de la capa de negocio, ya que esta no necesita saber el origen de los datos.

6.3.3. MVC, Modelo Vista Controlador

Como su propio nombre indica, hay tres conceptos clave en este patrón (ver Figura 6.8):

Modelo, representa la encapsulación de la información que se maneja en el sistema.

En el caso de este proyecto, los modelos manejados de objetos no están definidos en clases como suele ser común, están manejados directamente como objetos con el formato JSON.

Vista, representa los resultados que se le muestran al usuario tras procesar una petición. Es la parte visual de la interfaz de usuario que proporciona el sistema, formada por los archivos que usan HTML5, JavaScript, Jquery o Ajax. En resumidas cuentas es lo que el cliente ve, con lo que interactúa, lo que le muestra información.

Controlador, representa el manejo de la información de entrada y salida, la lógica del sistema.

En el caso de este proyecto está formado por los microservicios, con la gateway actuando de intermediario.

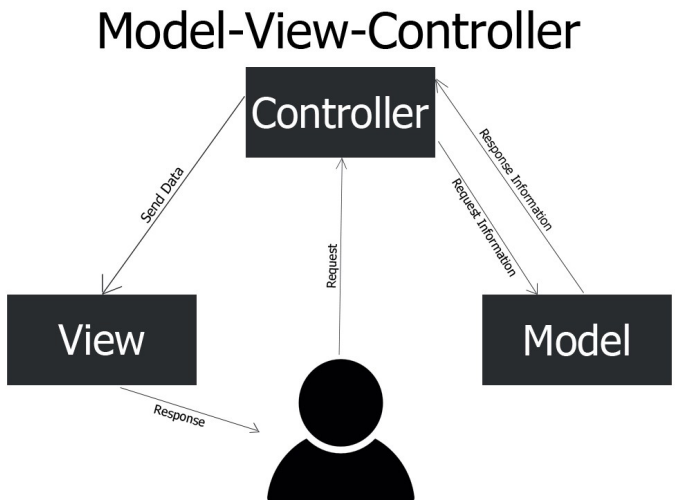

Figura 6.8: Patrón Modelo Vista Controlador. Figura tomada de [31].

Un aspecto negativo del patrón MVC es la gran cantidad de archivos que se manejan. Pero esto, por otra parte, es útil ya que ayuda a la comprensión del código. De esta forma el mantenimiento y desarrollo del sistema se agilizan enormemente.

6.4. Modelo dinámico

Para la representación del modelo dinámico que siguen las peticiones en el sistema respetando la arquitectura lógica propuesta, se seleccionarán ciertos casos específicos de historias de usuario que representan el conjunto de historias similares a ella.

El login es una de las historias más representativas del sistema, y el resto siguen un esquema similar. Por un lado, recibimos la petición mediante el cliente web, aunque puede ser realizado desde consola evitando la interfaz web.

A continuación, se obtiene la conexión con el microservicio de usuarios mediante Nameko-RabbitMQ. En otras palabras, Nameko proporciona el acceso a la cola de mensajes que se encuentra bajo el servicio RabbitMQ

Una vez en el microservicio se procesa la llamada mediante el uso de los objetos de acceso a datos y se crea el objeto que se necesita, en este caso la representación JSON del usuario. Dicho objeto usuario se encuentra vacío al principio y en caso de ser correcta la contraseña y nombre de usuario se asignan sus valores.

Este usuario es el que se devuelve a la gateway y en caso de tener información asociada se inicia la sesión que utilizarían los usuarios de consola. Por último, se inicializan los valores de las variables de sesión para los clientes web.

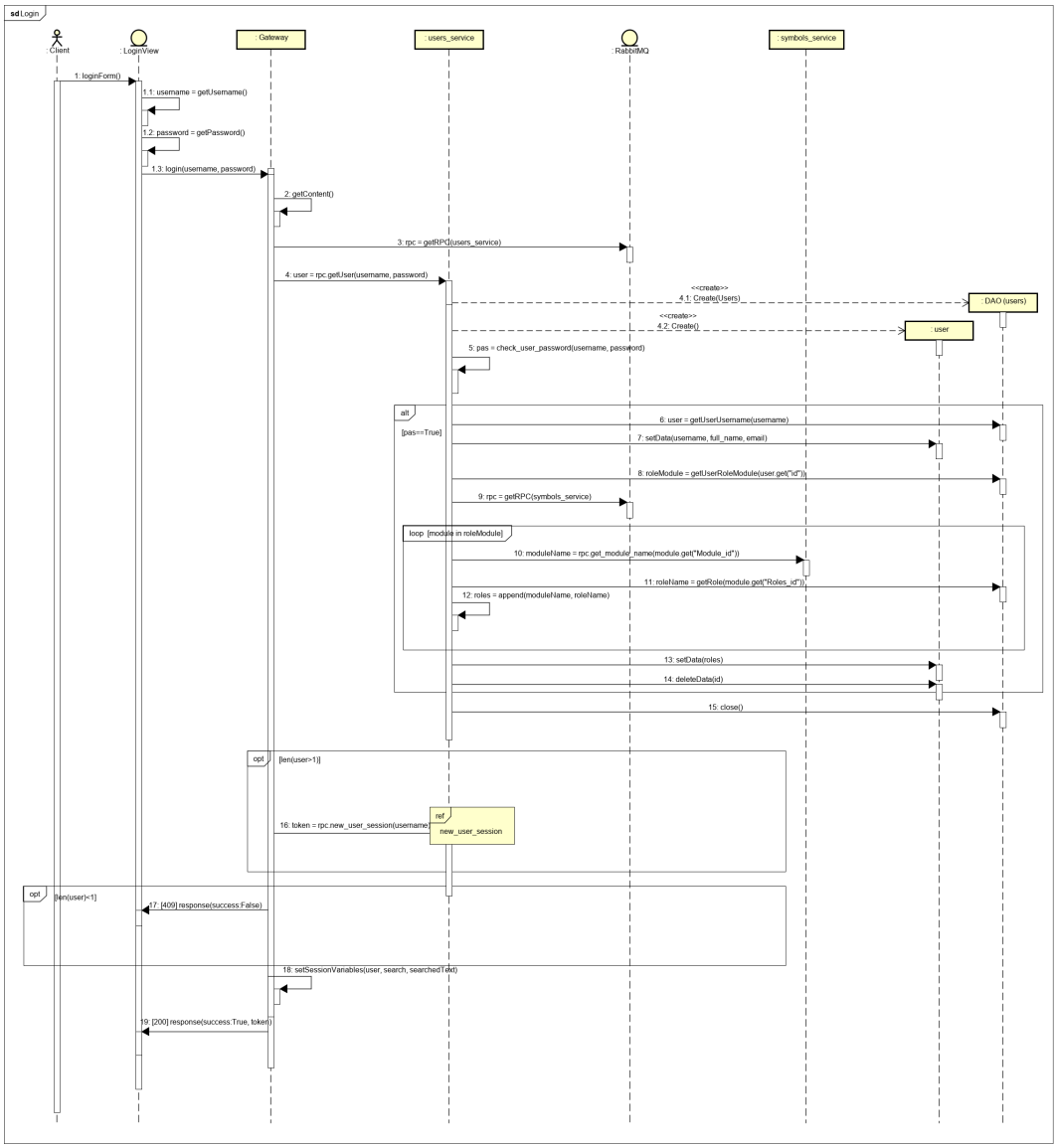

Figura 6.9: Diagrama de secuencia Login.

Una vez visto el funcionamiento del login, el resto de la estructura es similar, e incluso más sencilla ya que solo un par de funciones requieren del microservicio de símbolos, haciendo que se simplifique su uso.

A continuación, se van a mostrar otros dos ejemplos, que representarán el funcionamiento de la adición de índices y la modificación de categorías.

Se puede notar la diferencia entre estos diagramas con el inicial de login ya que en el resto hay más funciones de comprobación de errores o que deniegan permiso de acceso a las funciones que en el login.

Es preciso mencionar que las respuestas que se dirigen directamente al cliente son respuestas dirigidas únicamente a la comunicación mediante tokens y por lo tanto su objetivo es el cliente y no la interfaz web.

Por último, hay ciertas llamadas a funciones que no se han cubierto dentro del diagrama, pero que siguen el mismo esquema que el del método mostrado, no se muestran por falta de espacio y redundancia.

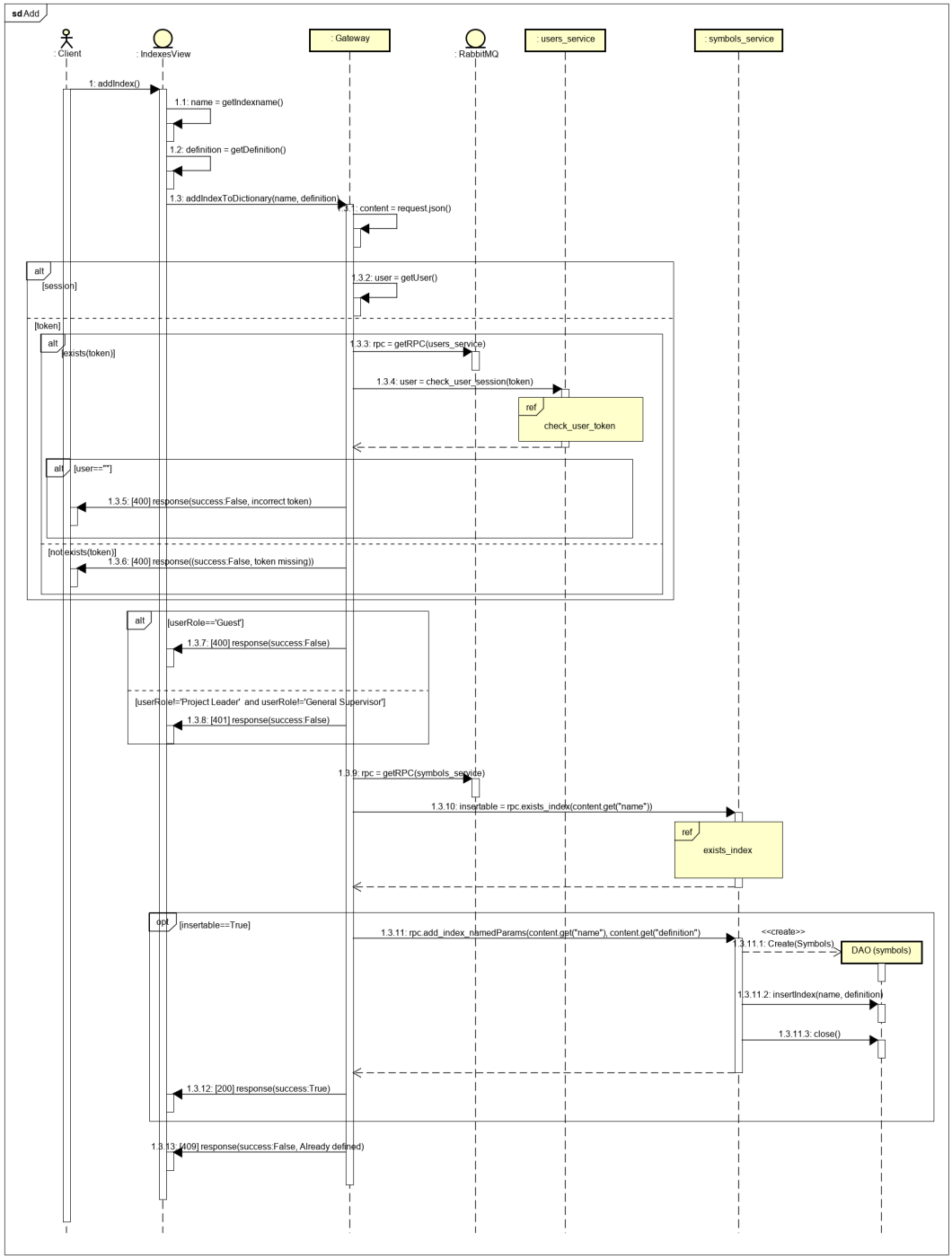

Figura 6.10: Diagrama de secuencia Añadir Índice.

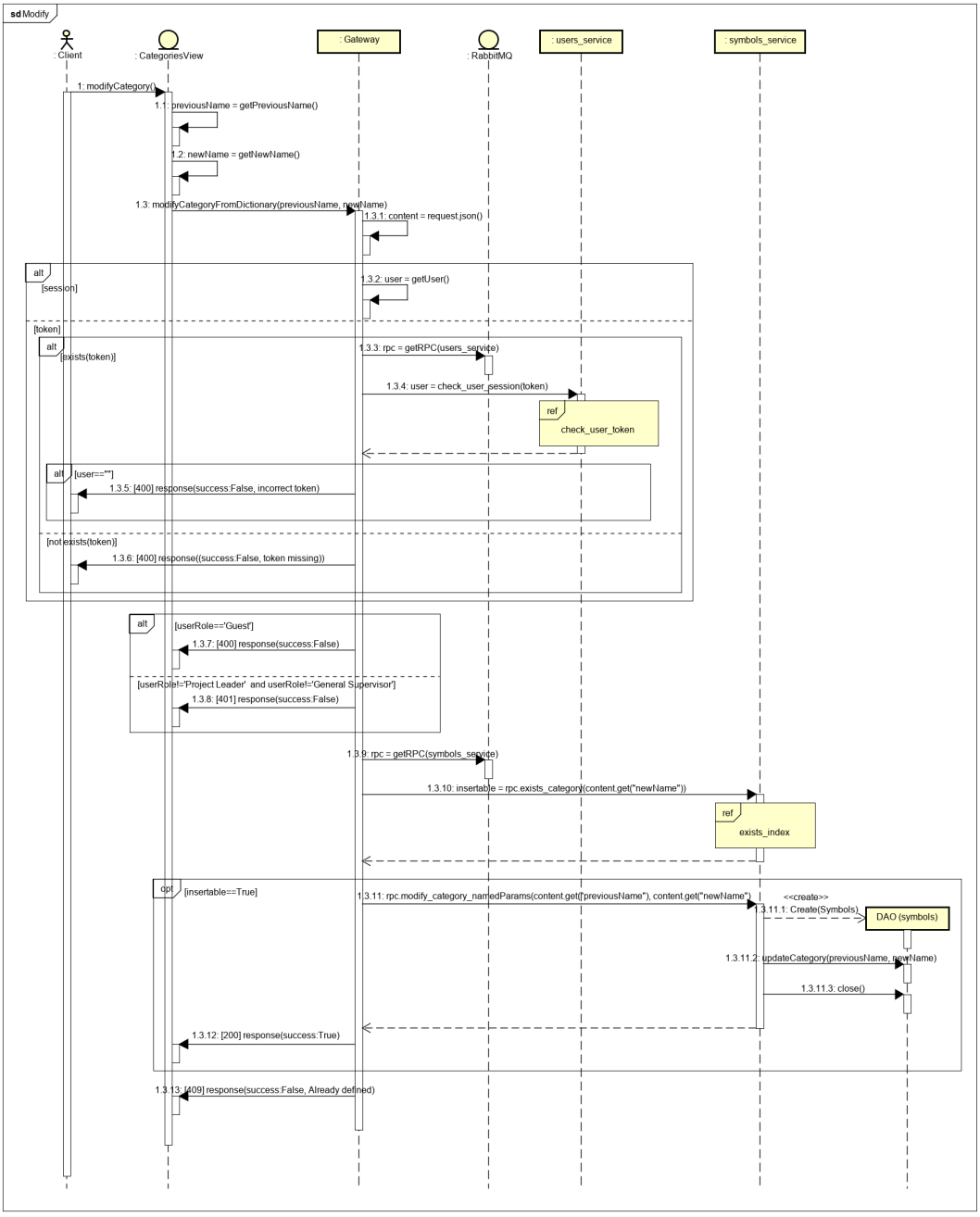

Figura 6.11: Diagrama de secuencia Modificar Categoría.

6.5. Diseño del despligue

El modelo de despliegue muestra la correspondencia de la arquitectura lógica sobre la arquitectura física.

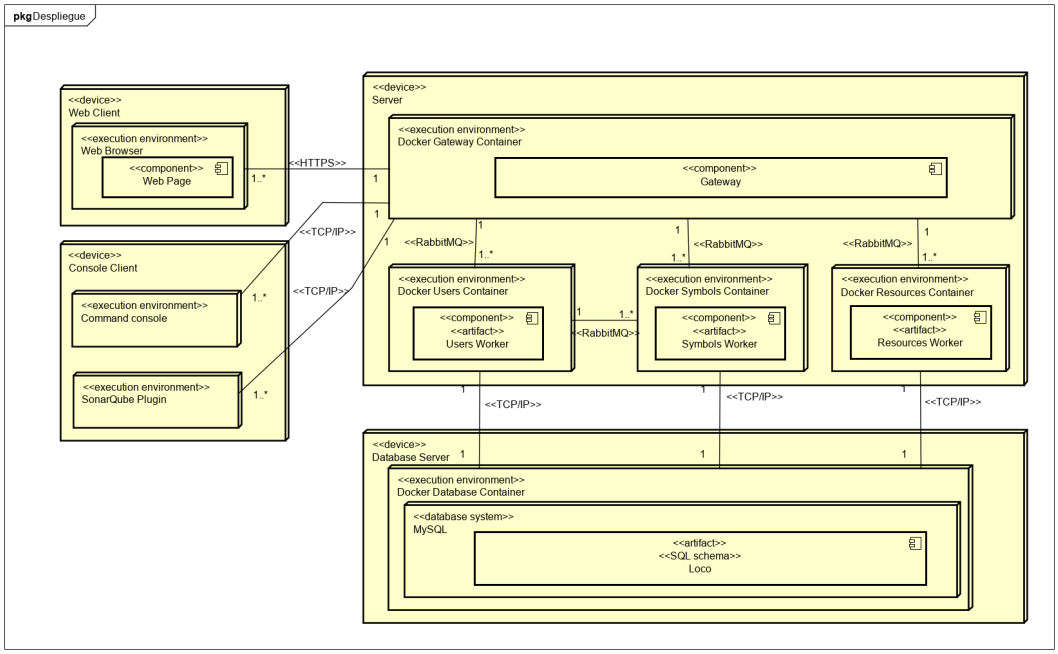

Figura 6.12: Modelo de Despliegue.

En este modelo que se representa en el diagrama de la Figura 6.12, podemos ver cómo se realiza la separación interna del servidor, el sistema puede tener la base de datos dentro del propio servidor o alojada en un dispositivo diferente como se muestra. Es importante indicar esto debido a que en el inicio del desarrollo se optará por el conjunto servidor-base de datos, y en un futuro se podría desvincular.

Por otro lado, se indica cómo la comunicación de los clientes pasa por la puerta de enlace directamente, y que es esta la que usa los microservicios. Se ha asociado a mayores el microservicio de usuarios con el de símbolos de forma que este pueda obtener información relativa a los módulos.

Las comunicaciones con los microservicios y base de datos pasan por las diferentes capas de estos hasta alcanzar su destino que es el componente en sí mismo.

Capítulo 7

Implementación y Pruebas

En este Capítulo se va a profundizar sobre la construcción del entorno de desarrollo utilizando Visual Code Studio y en la estructura de directorios subyacente que se ha utilizado. Por otro lado, se muestran las pruebas realizadas para verificar si el sistema software se comporta correctamente. Todo el proyecto se ha desarrollado desde un sistema Windows 10 y está diseñado para poder ser desplegado en cualquier sistema operativo que cumpla los requisitos de instalación.

7.1. Implementación

En esta sección se va a indicar el proceso de instalación, configuración y organización del sistema y se intentará proporcionar información sobre sus futuras posibilidades sin entrar en mucho detalle, ya que ese contenido se abordará al final de la memoria.

7.1.1. Instalación

El diseño es especialmente complejo pero gracias a esto la instalación es relativamente sencilla.

Se requiere de la instalación de los siguientes elementos dentro de la máquina que vaya a alojar el programa:

- python3, depende del sistema del que se disponga, pero la instalación suele ser sencilla. Para windows consultar [32], para Linux [30].
- docker-compose, en linux la instalación es muy sencilla, pero en windows se requiere "docker desktop" para su obtención[7].

- netcat, esta librería es necesaria instalarla ya que es la que usan los contenedores para comunicarse entre sí. "pip install netcat" es el comando necesario para obtener la librería. Para ello necesitamos Python.

Especialmente para Windows, debemos acceder a los ajustes de docker desktop, click derecho sobre su icono en la flecha de iconos ocultos al lado del reloj, y una vez allí compartir el disco en el que se vaya a alojar el sistema. De no realizar esto, los volúmenes de docker no se podrían guardar y el sistema no tendría uso alguno.

Una vez hecho esto, crear una consola en la carpeta del proyecto, desplazarnos hasta la carpeta "src", y ejecutar "docker-compose up".

7.1.2. Configuración

Una vez tenemos el sistema desplegado podemos optar por varios tipos de configuración que ayudarán al despliegue del mismo.

Se recomienda el uso de Nginx[25], para asignarle un dominio y conexión segura al sistema. Incluso se puede desplegar un servicio de Nginx desde docker.

En caso de tener conocimientos acerca de docker se puede modificar el archivo "docker-compose.yml", en el comando que ejecuta el servicio API. Se puede cambiar el puerto y dirección sobre los que se despliega el sistema. Por otro lado, se puede cambiar la contraseña y usuario de RabbitMQ desde el archivo .env que se encuentra a la altura del "docker-compose.yml".

Continuando con los archivos .env, si descendemos por "/src/main/api", encontramos un archivo al que le podemos cambiar la clave de sesión que usa Flask(SECRET_KEY).

Por último, si se desciende hasta "/src/main/api/static/css/projectInfo", se puede cambiar el texto del nombre, cabecera y descripción del proyecto en el que se utilizará el diccionario de datos. Para ello es necesario editar los archivos: name.txt, header.txt y description.txt respectivamente. En esta misma carpeta se encuentra el logo del proyecto, que se puede sustituir por otra imagen con el mismo nombre y extensión: "Logo.PNG"

Si se ha realizado cualquier tipo de cambio, desde la carpeta src se debería ejecutar "docker-compose build", y con el siguiente "docker-compose up" se habrían aplicado los cambios.

7.1.3. Organización del código

La forma más sencilla de mostrar la organización del código es mediante un diagrama (ver Figura 7.1) que muestre la jerarquía de carpetas y a su vez explicar el contenido de cada carpeta y su función. Por ello, a parte de la jerarquía principal de carpetas, se va a describir

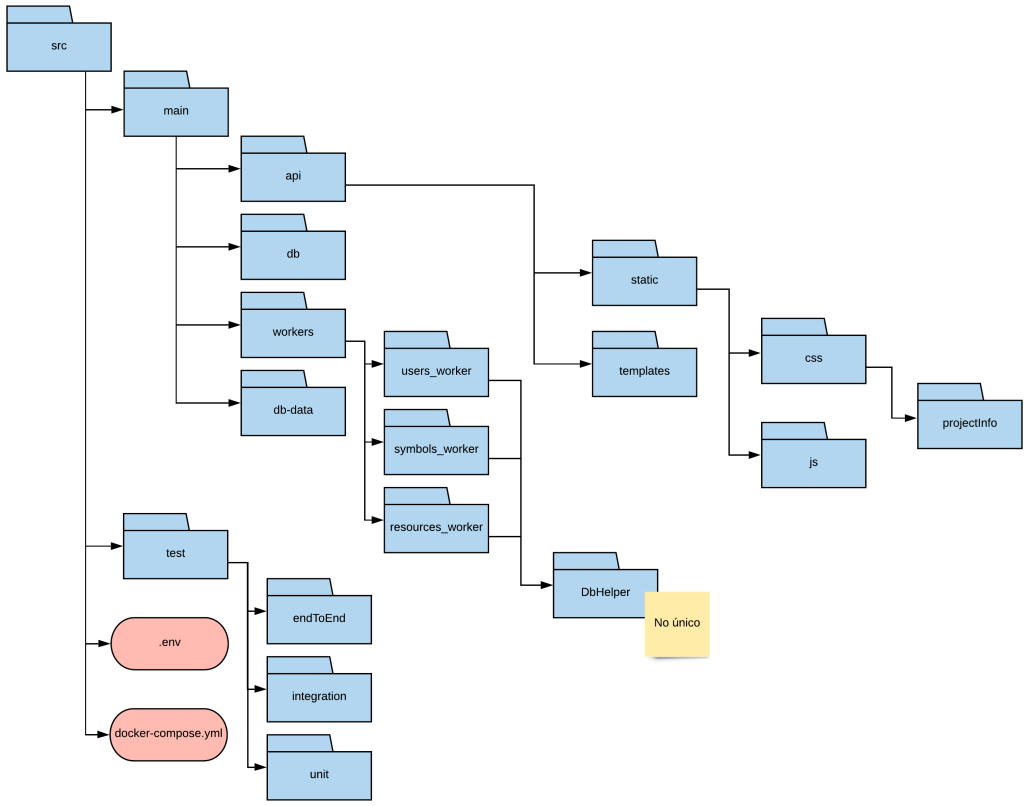

Figura 7.1: Diagrama de organización en carpetas.

la jerarquía de archivos que pertenece a uno de los microservicios "workers", al servicio "api" de puerta de enlace, y a la carpeta de estilos "css".

La carpeta **main** es la que contiene el código fuente del proyecto. Dentro de la misma se pueden observar 4 subcarpetas.

Las carpetas **db** y **db-data** son las que contienen información de la base de datos, **db** es la que contiene el esquema con el que se inicializa la base de datos y **db-data** en un principio debería estar vacía pero al ejecutar el servicio de arranque se debería volcar la información de la base de datos en esta carpeta.

Por otro lado, tenemos las carpetas de **api** y **workers**, en estas carpetas se encuentra el código de la puerta de enlace, gateway, y de los microservicios, workers, respectivamente.

Empezamos por la carpeta **workers** ya que es la que menos niveles subyacentes tiene. Esta carpeta contiene los 3 microservicios que se han desarrollado: **resources_worker**, **symbols_worker** y **users_worker**, dado que su estructura es similar, con describir uno de ellos

será suficiente para entender los tres.

Si describimos el contenido de **resources_worker**, podemos ver 9 archivos diferentes:

- **__init__.py**, este archivo es repetitivo para muchas carpetas y solo indica que se usa python en esa carpeta, por lo que no se mencionará más.
- **.env** es un archivo de configuración para varibles de entorno, se ve en el esquema de jerarquía principal que también lo usa **docker-compose.yml**.
- **config.yaml** es un archivo de configuración para Nameko y RabbitMQ.
- **requirements.txt** es un archivo en el que se listan las tecnologías que necesita instalar el contenedor docker.
- **resources_worker.py** es el archivo que contiene la lógica del microservicio, en este caso de recursos.
- **ResourcesDAO.py** es el archivo que contiene el acceso a datos, las peticiones a base de datos en concreto.
- **run.sh** este archivo lo usa docker para iniciar el servicio y esperar a que la cola de mensajes de RabbitMQ esté disponible.
- **DbHelper** es la carpeta que contiene un archivo de su mismo nombre **DbHelper.py**, que es el encargado de establecer la conexión con la base de datos y crear el DAO y destruirlo.
- **.docker** esta carpeta es un tanto especial, contiene un archivo "Dockerfile" con los comandos que ejecuta docker antes de levantar el servicio. Se encuentra presente en todas las carpetas que tengan un servicio que levantar. No se mencionará más ya que es igual en el resto de casos pero cambiando el nombre y los comandos pertinentes por cada servicio.

Por otro lado, tenemos la carpeta **api**, que contiene la gateway. Se pueden distinguir 6 archivos en su interior:

- **.docker**, ya mencionado su contenido en los workers.
- **static**, una carpeta en la que se profundizará a continuación. Contiene recursos para dar forma a las plantillas web.
- **templates**, una carpeta que contiene los archivos plantilla de HTML5 que sirven para dar forma a la interfaz de usuario.
- **.env**, es el archivo de variables de entorno de la puerta de enlace.
- **app.py**, es el archivo que contiene la lógica de la puerta de enlace (API gateway).
- **requirements.txt**, ya mencionado su contenido en los workers.

Como se ha indicado, la carpeta **static** contiene además dos carpetas extra en concreto: **css** y **js**.

Sobre **js**, se puede decir que es una carpeta con archivos JavaScript, que son los encargados de la dinámica de las acciones web.

Por su parte, **css**, contiene los archivos de estilo que hacen que la página luzca de forma más ordenada y agradable a la vista. Dentro de esta podemos destacar dos cosas, el archivo llamado **bootstrap.min.css**, un archivo comprimido que contiene el Bootstrap aplicado al formato de las páginas web, y **projectInfo**, una carpeta con los archivos de relleno para la página como el logo, nombre, etc. La otra carpeta solo vale para obtener dibujos para usar como botones, como las flechas de los popovers de la paginación. Los dos archivos restantes son unas líneas de estilo personalizadas y un fondo para la página.

7.2. Pruebas

Las pruebas realizadas en el sistema se han llevado a cabo en su mayor parte de forma continua, se ha intentado mantener todo el sistema actualizado y se han aislado los fragmentos que presentaban errores hasta llegar a solucionarlos. Este proceso se ha realizado de forma manual desde el comienzo del desarrollo del proyecto y cada vez que se añadía funcionalidad nueva al mismo.

Para ver estrategias de pruebas consultar: [28]. En concreto Nameko proporciona unos ejemplos de tests de unidad e integración: [19].

7.2.1. Test unitarios

Los tests unitarios son los que se enfocan en probar cada parte de funcionalidad aislada del sistema. En este caso, las funciones de los microservicios son el objetivo más importante que se puede fijar para probar la lógica de negocio y que todo funcione como se espera. Gracias a este enfoque se pueden localizar errores en métodos concretos y evitar la propagación de dichos errores a otros puntos del sistema.

Debido a la falta de tiempo necesario para llevar a cabo el desgranamiento que necesita una automatización de los test unitarios, se va a prescindir de su realización.

No obstante, se ha buscado la forma de llevarlos a cabo, consultar la documentación de nameko [19], para saber más.

7.2.2. Test integración

El objetivo principal de los tests de integración es el de comprobar que la comunicación se realiza correctamente entre diferentes módulos que componen al sistema, en nuestro caso entre la puerta de enlace y los microservicios.

Por ello, los test de integración se van a llevar a cabo con el despliegue completo del sistema, pero sin usar el sistema web del mismo **A.3.2**, es decir se podrán a prueba siguiendo el llamado "Big Bang approach" [23] todos los componentes al mismo tiempo, la puerta de enlace y los microservicios desde peticiones generadas a la puerta de enlace directamente desde consola.

Existen otros enfoques diferentes, que implican simulaciones de los objetos que deberían comunicarse con los probados, pero su desarrollo requeriría mucho más tiempo del que se dispone en esta etapa del proyecto.

7.2.3. Test end-to-end

El objetivo de los tests end-to-end es probar el despliegue completo del sistema de forma que desde un extremo, cliente web, se realice todo el proceso de gestión de la petición y transformación de la misma en una respuesta que sea la adecuada o, en otras palabras, la esperada. En caso de recibir una respuesta errónea se tendría que revisar el recorrido que ha realizado dicha petición para localizar el error.

Por la misma razón que en los test unitarios, estos tests no se han podido implementar de forma automatizada. En un principio se iban a llevar a cabo con Selenium [18]. No obstante, se han realizado de forma manual a medida que el desarrollo del proyecto avanzaba de forma sistemática para probar toda la funcionalidad web añadida.

Para compensar su ausencia pero al mismo tiempo mostrar cómo se han llevado a cabo las pruebas manuales, se detallará cada caso de uso en el manual de usuario **A.3.1**, de forma que se verán reflejados los diferentes escenarios representativos del sistema.

7.2.4. Problemas encontrados

La gran mayoría de los problemas encontrados han surgido en términos de desarrollo, pero también se han encontrado diversos problemas a la hora de desplegar las versiones tempranas del proyecto.

Para empezar, la ejecución de los servicios requería cierta estructuración en las carpetas. Además, los microservicios tenían dependencias tecnológicas y resolverlas llevó bastante tiempo (alrededor de una semana dedicada a ello) en el sprint inicial de preparación del proyecto.

Por otro lado, la persistencia de la base de datos y sus cambios fueron un gran problema a tratar ya que requerían del paso especial mencionado en **7.1.1**, donde se comparte el disco con Docker para poder utilizar su espacio de almacenamiento. La documentación aportada en Docker [8] no fue de especial ayuda y se perdió mucho tiempo consultando foros para poder llegar a la versión final.

Por último, se han encontrado los siguientes errores en las pruebas realizadas:

- Los algoritmos de símbolos no están correctamente formados. En ellos se pide un módulo, y se comprueban los permisos del usuario sobre el módulo. Es decir, en caso de introducir un símbolo y un módulo no asociados por consola, se interpreta el permiso del usuario asociado al módulo en vez de obtener el módulo del símbolo y el permiso del usuario derivarlo de este. La solución encontrada es extraer el módulo directamente del símbolo.
- No se comprueba la longitud de los datos, por lo que se pueden desencadenar errores en las peticiones a base de datos.

Ninguno de estos errores se ha podido corregir debido a la falta de tiempo y momento en el que se han encontrado. Esto se anotará en trabajo futuro (ver Sección **9.2**).

Ambos errores son derivados del desarrollo enfocado en web y de repartir las comprobaciones de seguridad entre el frontend y backend para no sobrecargar la puerta de enlace con demasiadas peticiones erróneas.

Capítulo 8

Seguimiento del proyecto

En este Capítulo se pretende mostrar con unidades de trabajo medidas en horas la realización del proyecto, de forma que se puedan ver y analizar las subdivisiones de trabajo realizadas, la carga de trabajo asignada, las estimaciones de tiempo previstas y las empleadas reales y la categorización empleada en dichas unidades de trabajo.

En resumidas cuentas, se llevará a cabo un trabajo de seguimiento del proyecto, caracterizado por actividades con el siguiente formato:

Historia	**Tipo**	**Descripción Actividad**	**Tiempo Estimado**	**Tiempo Empleado**	**Estado**
US-000	Core	Nombre de la historia de usuario	5h	3h 30m	Completa

Tabla 8.1: Tabla de ejemplo del product backlog.

Cada campo es auto descriptivo, como podemos ver, de izquierda a derecha:

- Historia: RF: "Requisito funcional", NRF: "Requisito no funcional" historia de usuario en Español, 000: número de historia de usuario.
- Tipo: la categoría de la actividad, puede haber categorías como Bugs o Core, destinadas a abordar errores del sistema o por otro lado actividades esenciales para el desarrollo del proyecto.
- Descripción Actividad: la descripción de la actividad es el nombre dado a la misma.
- Tiempo Estimado: el tiempo estimado en la actividad es el tiempo que se prevé que dure.
- Tiempo Empleado: el tiempo empleado el tiempo real invertido en la actividad.

- Estado: el campo estado indica si la actividad ha sido completada o no, y si esta ha sido iniciada.

Y al pie de todo esto estaría el resultado del sprint:

Total	**5h**	**3h 30m**	**1:1**

Tabla 8.2: Ejemplo de pie del *sprint backlog*.

De nuevo, de izquierda a derecha:

- Total: únicamente representa una etiqueta para indicar que nos encontramos en el pie de tabla.
- Segundo campo: las horas estimadas totales en el sprint.
- Tercer campo: las horas empleadas totales en el sprint.
- Cuarto campo: la relación de actividades previstas y comprobadas (el número de la izquierda representa las completadas y el de la derecha las previstas).

8.1. Sprint 0

8.1.1. Sprint planning

Como inicio del proyecto, se necesita recabar información y tener un ejemplo simple para poder partir de él hacia el desarrollo real del proyecto. Por ello, el trabajo que se dedicará a este sprint de preparación es en mayor parte formativo e informativo, aprender tanto la teoría estructural que aplicaremos para el desarrollo de los microservicios como aprender a usar la tecnología que necesitaremos para su implantación.

Durante este sprint se buscarán las tecnologías y se probará su compatibilidad, al mismo tiempo, se desplegará un ejemplo sencillo que representará el futuro sistema.

8.1.2. Sprint backlog

En este sprint no se planificó el tiempo ya que se empleó de diversas formas, estudiar herramientas y patrones o implementar y probar ejemplos para el comienzo del desarrollo.

8.1.3. Sprint review

Desde el punto de vista teórico: se han analizado los microservicios y se han seleccionado las tecnologías iniciales que se utilizarán en el desarrollo del proyecto, y revisado su documentación. Podemos ver esta clasificación en la sección anterior de herramientas utilizadas. Por otro lado, se han estudiado los patrones a utilizar para plantear el ejemplo inicial y se han leído artículos relacionados con la escalabilidad, ventajas e inconvenientes de los microservicios y DevOps.

Desde el punto de vista práctico: se ha creado un ejemplo inicial con "docker-compose" y un "worker" inicial que sirve de ejemplo para comprobar la comunicación entre contenedores. Se ha creado un contenedor de base de datos de ejemplo y se ha accedido a él. Se han creado endpoints en el lado del servidor a los cuales les podemos mandar peticiones desde el frontend, ambos son sencillos solo para comprobar su funcionamiento con Flask.

8.2. Sprint 1

8.2.1. Sprint planning

Para empezar el desarrollo real de la aplicación necesitamos establecer las bases iniciales, partiendo del ejemplo que se planteó en el sprint 0.

Lo primero que se necesita es crear una interfaz de usuario para implementar el código JavaScript que realice las peticiones a servidor mediante JSON. Con el formato JSON enviarle los datos a un "worker", y devolverlos con el formato deseado a la gateway para volver a enviar la respuesta al cliente.

Adicionalmente, el manejo básico de los usuarios para poder establecer un registro inicial, y la distinción de los roles de los mismos obtenidos de la base de datos. Se crearán sesiones para que los usuarios que se hayan introducido en el sistema no tengan que poner sus credenciales cada vez que hagan una petición a servidor.

Aparte de todo esto, se realizará limpieza de código inutilizado del ejemplo y se enlazarán las referencias web entre sí.

8.2.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación sprints	5h	4h	Completa
-	Core-01	Limpieza de workers de ejemplo y YAML del docker-compose	2h	2h	Completa
FR-1	Core-02	Creación de una clase abstracta para usuarios	5h	5h 30m	Completa
FR-1	Core-03	Manejo de usuario mediante JSON serializable	2h	2h	Completa
-	Core-04	Asignación de keys de sesión	5h	4h	Completa
-	Core-05	Enviar y recibir JSON	8h	7h	Completa
FR-1	Core-06	Distinción de roles de usuario	8h	9h	Completa
-	Core-07	Adaptar base de datos para pruebas con usuarios	3h	2h	Incompleta
FR-22	Core-08	Búsqueda dependiendo de página y tabulación	8h	3h	Incompleta
-	Core-09	Enlazar página	1h	1h	Completa
FR-16	Core-10	Opciones y tabulaciones de settings usuario	8h	6h 30m	Completa
-	Core-11	Interfaz de usuario inicial	8h	8h	Completa
-	Bug-01	Base de datos eliminó los permisos de root	1h	4h	Completa
-	Bug-02	Docker-compose no crea la imagen correctamente	2h	3h	Completa
-	Bug-03	Archivos Javascript no cargan en el cliente web	4h	3h	Completa
Total			**70h**	**64h**	**13:15**

Tabla 8.3: Sprint 1 backlog.

8.2.3. Sprint review

He visto que las sesiones de Flask son seguras generalmente, por ello he asumido que la conexión segura a través de HTTPS, clave secreta pasada a través de una variable de entorno para la sesión y el encriptado de la contraseña en base de datos serán suficientes medidas de seguridad para evitar acciones mal intencionadas, de todas formas se puede usar sesiones de Flask-Sessions, que nos proporciona algunos métodos extras de seguridad aunque al no

guardar datos relevantes en los objetos de sesión, no necesitaremos llegar a tales niveles de complejidad.

Se ha planteado un modelo inicial de interfaz y de peticiones de búsqueda, y se ha creado una barra de navegación en la parte superior de la página web.

Se ha realizado una revisión a la interfaz inicial planteada, y se han corregido algunos aspectos para mostrar los recursos correctamente, pestañas de tabulación.

Las opciones de usuario serán desarrolladas a medida que se requiera su funcionalidad.

Se ha anotado la falta del campo de definición de los índices en el modelo inicial de base de datos y astah.

Queda pendiente el desarrollo de las búsquedas en el lado del servidor y la adaptación de la base de datos de pruebas para manejar a los usuarios. Pasan al siguiente sprint las tareas: Core-09 y Core-08.

8.3. Sprint 2

8.3.1. Sprint planning

Se quiere plantear el uso de Mariadb como base de datos, y hay que terminar de adaptar la base de datos inicial al esquema propuesto.

Se ha reservado un poco de tiempo a la búsqueda de información para completar el lado cliente de las búsquedas y la adaptación de la base de datos a la nueva.

Se tiene que hacer una revisión al proyecto GitLab, traducir historias de usuario e introducirlas antes de plan de proyecto.

Finalmente, empezar la implementación de los símbolos del diccionario de datos, comenzando por las peticiones dirigidas a la API que se acoplarán con el plugin SonarQube que se está desarrollando de forma paralela en otro Trabajo de Fin de Grado. Representados por las historias de usuario: NFR-06 y NFR-07.

8.3.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	4h	3h	Completa
-	Core-07	Adaptar base de datos para pruebas con usuarios	5h	8h	Completa
FR-22	Core-08	Búsqueda dependiendo de página y tabulación	8h	10h	Completa
-	Core-12	Capítulo de requisitos e historias de usuario	6h	6h	Completa
-	Core-13	Búsqueda de información para el sprint	4h	4h	Completa
-	Bug-04	Mariadb no funciona con un punto de montado local en docker-compose	4h	4h	Completa
NFR-07	NFR-07	Responder a una petición si los símbolos recibidos están validados y su información relativa	10h	-h	No iniciado
NFR-06	NFR-06	Añadir símbolos e información vía peticiones a la API RESTful	10h	2h	Incompleta
Total			**51h**	**37h**	**6:8**

Tabla 8.4: Sprint 2 backlog.

8.3.3. Sprint review

Se han nombrado las historias de usuario de forma que ahora se identificarán por FR para las funcionales, NFR para las no funcionales e IR para las de información. Al mismo tiempo se han actualizado los tableros de GitLab con las historias nuevas y su nombrado, y se ha establecido el modelo "T-Shirt" de asignación de carga por historia en GitLab, similar al tiempo estimado de duración de las actividades planteado en estos sprint backlogs.

Se ha establecido un fondo de pantalla para la página.

En cuanto a la base de datos, Mariadb ha dado problemas en su despliegue con docker, como solución se seguirá con MySQL ya que ambos utilizan un conjunto de consultas similar. Por otro lado, en la generación de las tablas, se han realizado ciertos cambios relacionados con las claves únicas que no estaban presentes en el esquema de creación inicial, y se ha cambiado el nombre de la tabla "Index" por "IndexI", ya que es una palabra reservada. Lo mismo ocurre con el atributo "usage" de los adjetivos, por lo que lo he cambiado a "usages".

Casi a mitad de sprint se ha declarado el estado de alarma en el territorio nacional por la pandemia derivada del CoVid-19 y por lo tanto se pasará a teletrabajar desde casa. Las reuniones serán realizadas mediante Skype y se mantendrán en el mismo horario.

Se ha llegado a iniciar el requisito: NFR-07, pero solo se ha realizado un planteamiento inicial. Queda pendiente su desarrollo junto con el del requisito NFR-06.

8.4. Sprint 3

8.4.1. Sprint planning

Se pretende crear el microservicio que de soporte a las peticiones relacionadas con símbolos, esto será parte del desarrollo centrado en los requisitos NFR-06 y NFR-07, y en parte servirá para probar la lógica de acceso a base de datos para los símbolos. Y por otro lado, se probarán las peticiones mediante consola en el lado del servidor.

Se plantea el uso de un sistema de tokens para acceder a la funcionalidad sin necesidad del uso de una sesión Flask que en consola no funciona.

Se plantea abordar el requisito FR-02 en caso de que todo lo anterior funcione correctamente y que sobre tiempo.

8.4.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	3h	1h 30m	Completa
-	Core-14	Búsqueda de información para el sprint	4h	3h	Completa
NFR-07	Core-15	Scripts SQL de prueba para las tareas del sprint	3h	3h	Completa
NFR-07	Core-16	Scripts CURL y cadenas JSON de prueba para las tareas del sprint	3h	3h	Completa
NFR-07	Core-17	Plantear esquema con los datos actuales, entrada/salida	1h	1h	Completa
NFR-07	Core-18	Crear nuevo worker y actualizar docker-compose	1h 30m	1h 30m	Completa
NFR-07	Core-19	Crear lógica para extraer los datos de la base de datos	7h	8h	Completa
NFR-07	Core-20	Debug métodos de los workers para extraer y formatear los datos de la Base de Datos	6h	6h	Completa
NFR-07/ NFR-06	Core-21	Comprobar clave de acceso/token y crear usuario dedicado para estos trámites	4h	4h	Completa
NFR-06	NFR-06	Añadir símbolos e información vía peticiones a la API RESTful	10h	2h	Incompleta
FR-02	FR-02	Programador de módulo introduce metadatos pendientes de validación en el diccionario	10h	-h	No iniciado
Total			**52h 30m**	**32h**	**10:12**

Tabla 8.5: Sprint 3 backlog.

8.4.3. Sprint review

Se ha realizado una subdivisión del NFR-07 en tareas menores para poder ver su desarrollo.

Se han preparado scripts de prueba para comprobar la introducción y extracción en la base de datos.

Se ha buscado información acerca de la recepción de peticiones JSON en el servidor y de la formulación de peticiones al servicio de base de datos.

En cuanto a la obtención de los símbolos, se plantea su formato y su petición a base de datos usando siempre identificadores únicos para no caer en duplicados o errores accidentales. Para ello se han añadido identificadores que se incrementan de forma automática. Por otro lado, la tabla de Categorías ha sido cambiada para dejar que el campo que representa a su super-categoría sea nulo en caso de no existir.

Se ha añadido el rol de asegurado de calidad al sistema para llevar a cabo los requisitos NFR-06 y NFR-07.

Finalmente se ha planteado cambiar las conexiones a base de datos y adoptar un patrón de acceso a datos para el siguiente sprint.

8.5. Sprint 4

8.5.1. Sprint planning

Debido a que en el sprint anterior se retrasó bastante la planificación del proyecto, se pretende dar por terminados los requisitos de conexión con el plugin de SonarQube y abordar finalmente los requisitos de implementación de metadatos, empezando por los de los símbolos.

Prioritariamente, se van a cambiar las conexiones a base de datos lo antes posible para que no supongan un problema en el futuro, y se intentará seguir el modelo “DAO+DTO” para su reescritura.

8.5.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	3h	1h 30m	Completa
-	Core-22	Búsqueda de información para el sprint DAO	3h	3h	Completa
-	Core-23	Actualizar backlog gitlab	2h	2h	Completa
NFR-07	Core-24	Refactor conexión base de datos	3h	9h	Completa
NFR-06	Core-25	Crear métodos de inserción en el DAO	2h	1h 30m	Completa
NFR-06	Core-26	Crear lógica del worker para la inserción y comprobación de las entradas	4h	5h	Completa
NFR-06	Core-27	Crear punto de entrada en la gateway	1h	1h 30m	Completa
NFR-06	Core-28	Debug y pruebas	4h	4h	Completa
FR-22	Core-29	Mostrar símbolos en el diccionario Web	8h	12h	Completa
FR-02	FR-02	Programador de módulo introduce símbolos pendientes de validación en el diccionario Web	10h	2h	Incompleta
FR-03	FR-03	Programador de módulo modifica símbolos y pasan a pendientes de validación en el diccionario Web	10h	-h	No iniciado
Total			**50h**	**41h 30m**	**10:12**

Tabla 8.6: Sprint 4 backlog.

8.5.3. Sprint review

Se ha creado el modelo de conexión a la base de datos planteado (DAO+DTO), aunque ha consumido mucho más tiempo del estimado. Se han vuelto a plantear todas las conexiones con el nuevo modelo, y se han actualizado los métodos que se habían visto afectados.

Se han actualizado las tablas de Gitlab para mostrar las historias de forma que muestren las tareas realizadas.

Se ha creado la interfaz de los símbolos, tarea que ha sido más lenta de lo previsto y por lo tanto ha lastrado el avance del proyecto de nuevo. Se ha creado un formularios inicial para

añadir símbolos.

Se han realizado pruebas y debug en los métodos de acceso a base de datos y nuevas funcionalidades.

Quedan pendientes la introducción y modificación de los símbolos.

8.6. Sprint 5

8.6.1. Sprint planning

Para este sprint se tienen que introducir los datos de los símbolos en el sistema, su modificación y borrado.

Hay que empezar a distinguir entre las acciones que realiza cada usuario y desactivar los botones que tengan asociadas las acciones no permitidas, en caso de los usuarios invitados, no se mostrarán dichos botones ni las pestañas de pendiente.

8.6.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	2h	1h	Completa
FR-02	FR-02	Programador de módulo introduce metadatos pendientes de validación en el diccionario Web	10h	12h	Completa
FR-03	FR-03	Programador de módulo modifica metadatos pendientes de validación en el diccionario Web	5h	5h	Completa
FR-03.1	FR-03.1	Supervisor de módulo borra metadatos del diccionario Web	5h	5h	Completa
FR-08, FR-12	FR-08, FR-12	Supervisores realizan las tareas de los roles inferiores	5h	2h	Incompleta
FR-03 y FR-03.1	Core-30	Botones de acción en la interfaz	1h	1h	Completa
-	Core-31	Mejorar diseño de interfaz de usuario	2h	2h	Completa
-	Core-32	Formato JSON Python-JavaScript, para trabajar con cada uno en su dominio	6h	7h	Completa
-	Core-33	Alertas para notificar sobre los resultados de las acciones	4h	4h	Completa
-	Bug-	Bug interfaz de usuario layout principal no carga (JavaScript)	2h	2h	Completa
Total			**42h**	**41h**	**9:11**

Tabla 8.7: Sprint 5 backlog.

8.6.3. Sprint review

Se ha planteado una aproximación diferente para el código del frontend, en vez de tener JavaScript y HTML todo junto se ha separado el código JavaScript en una carpeta diferente. De esta forma el código será mucho más legible.

Los botones de acción no se muestran en caso de ser el usuario invitado.

Se han detectado requisitos que faltan, borrar símbolos como supervisor de módulo (FR-3.1) , borrado y promoción de módulos principales asociados al símbolo como supervisor general.

Los requisitos FR-12 y FR-8 se han planteado en una función JavaScript y dentro de la gateway, se irán completando a medida que se creen las diferente páginas.

Se ha mejorado la distribución de la interfaz de usuario para que sea más compacta y agradable a la vista.

Se ha abordado la adición de módulos secundarios desde el desplegable de símbolos, para el programador de módulo o superior, se ha hecho un botón que permite acceder a su contenido y su modificación.

Se han creado alertas que notifican de los resultados de las peticiones al servidor web, y de su error de formulación. Existe un problema, que al cargar la página no se cargan las alertas porque se vuelve al código base. Se dejará esto para el final del proyecto ya que no es prioritario.

Se han completado las actividades previstas inicialmente para el sprint, pero dado que sobró un poco de tiempo se ha planteado la creación de unos filtros en los símbolos y evitar que un símbolo se borre en caso de tener módulos secundarios.

8.7. Sprint 6

8.7.1. Sprint planning

Como hemos indicado en el sprint anterior, las tareas relacionadas con FR-8 y FR-12 se repiten a lo largo de los diferentes sprints por lo que junto con el FR-03, vamos a juntar todas las tareas de permisos en un código JavaScript para el frontend y por otro lado, en el backend mediante la cancelación de las peticiones. Además, al ser recurrentes, los futuros sprints tendrán de forma implícita estos requisitos incluidos.

Se crearán filtros para los símbolos, uno para sus módulos secundarios y otro para las categorías.

Se necesita añadir un nuevo campo a los símbolos para distinguir los que estén insertados vía web o consola: "Is_indexed", que permitirá establecer la indexación de un símbolo en caso de que este esté asociado a estos.

Se quiere marcar los campos incorrectos en un formulario para evitar que peticiones sin rellenar sean enviadas al servidor y lo colapsen.

Se quiere evitar el exceso de símbolos en la misma página, por ello se creará un sistema de paginación.

8.7.2. Sprint backlog

Tabla 8.8: Sprint 6 backlog.

Historia	**Tipo**	**Descripción Actividad**	**Tiempo Estimado**	**Tiempo Empleado**	**Estado**
-	Weekly	Actualizar documentación del sprint	2h	1h	Completa
FR-08	Weekly	Supervisor de módulo realiza las tareas del programador en su módulo	-h	-h	Recurrente
FR-12	Weekly	Supervisor general realiza todas las tareas del supervisor de módulo	-h	-h	Recurrente
FR-03 FR-12 FR-08	Core-34	Usuarios con permiso modifican módulos secundarios	4h	6h	Completa
FR-03 FR-12 FR-08	Core-35	Usuarios con permiso modifican índices	4h	3h	Incompleta

Continúa en el siguiente página

Viene de la página anterior

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Core-36	Barra principal de navegación cambia dependiendo de la opción	1h	1h	Completa
FR-22	Core-37	Filtros al seleccionar módulo principal o categoría en símbolos	4h	4h	Completa
FR-05 FR-06	Core-38	Restringir botones de acción dependiendo del rol y módulo al que se tenga acceso	4h	4h	Completa
FR-04	Core-39	Supervisor de módulo valida símbolo y crear comprobaciones de corrección	4h	2h	Incompleta
FR-04	Core-40	Añadir campo "Is_indexed" a los símbolos para distinguir los insertados vía web y comando	3h	3h	Completa
-	Core-41	Marcar los campos incorrectos en caso de añadir y modificar símbolos	4h	4h	Completa
FR-15	Core-42	El supervisor general puede eliminar el módulo principal de un símbolo y promover uno secundario a principal	4h	4h	Completa
-	Core-43	Paginación de símbolos	8h	3h	Incompleta
Bug	Bug-	JavaScript no admite que los nombres de las id tengan espacios o no reconoce los desplegables	2h	2h	Completo
Total			**44h**	**37h**	**11:14**

Fin de la Tabla Sprint 6 backlog

8.7.3. Sprint review

Se ha actualizado el código de la barra de navegación para cambiar a la pestaña activa.

Se han creado los filtros de símbolos de manera que pueden ser aplicados al mismo tiempo.

Se ha encontrado un error en el que si un id tiene espacios no se reconocía y funcionaban mal los desplegables. Se impide la creación de nombres de dato con espacios en blanco. Las convenciones de nombre se dejan a gusto del usuario.

Se ha trabajado especialmente en la gestión del código de permisos y en la forma de mostrar los campos deshabilitados.

Se han añadido alertas de señalización para la información mal formada en la modificación, validación, borrado y a la hora de añadir nuevos símbolos. Adicionalmente se permite cambiar el valor del campo "Is_indexed" en la creación de estos, y se permite la gestión de los índices. Se permite modificar y añadir/borrar valores desde la pestaña de índices.

A parte se han solucionado pequeños errores como la creación de datos duplicados y se han creado indicadores de alerta en caso de enviar la petición y presentar dicho error.

Se ha planteado la gestión de índices de modo que los duplicados no afecten al comportamiento del sistema.

Se ha comenzado a implementar la paginación de símbolos.

8.8. Sprint 7

8.8.1. Sprint planning

En este sprint se pretende acabar con los símbolos y reproducir su contenido en los índices para agilizar el desarrollo.

Se pretende abordar todo el contenido de los índices ya que ya disponemos del modelo previo de los símbolos y terminar la paginación de símbolos que quedó pendiente en el sprint anterior, añadiendo la creación de popovers que ayuden a desplazarse por las páginas.

Se reservará cierto tiempo para adaptar la interfaz a los índices.

8.8.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	2h	1h	Completa
FR-03 FR-12 FR-08	Core-44	Usuarios con permiso modifican índices asociados a un símbolo	2h	-h	No iniciada
FR-3.2	Core-45	Añadir índice y asignarle valores	4h	3h	Completado
-	Core-46	Búsqueda de información popovers	2h	2h	Completado
-	Core-47	Paginación de símbolos	8h	12h	Completa
FR-03	Core-48	Reproducir la funcionalidad de los símbolos en el apartado de índices	14h	12h	Incompleta
-	Core-49	Adaptación de la interfaz a la información de los índices	2h	2h	Completa
-	Core-50	Paginación de índices	2h	2h	Completa
Total			**36h**	**34h**	**6:8**

Tabla 8.9: Sprint 7 backlog.

8.8.3. Sprint review

Se finaliza la creación del script de paginación y popovers de símbolos y se generaliza para poder ser rápidamente adaptado al resto de tablas futuras.

Se permite la asociación de múltiples símbolos con un índice, pero este no puede tener valores repetidos.

Solo se permite que los supervisores generales modifiquen los índices debido a que los usuarios pueden asignarlos a los símbolos y se estarían dando permisos a sí mismos sobre su edición, cosa que se ha querido evitar.

Se evita la creación de índices con el mismo nombre. Solamente se permite eliminar un índice si no esta asociado a ningún símbolo. Al borrar el índice también se borran sus elementos.

Queda pendiente la modificación de permisos de interfaz en los índices.

8.9. Sprint 8

8.9.1. Sprint planning

En este sprint se pretende finalizar las tareas previas de gestión de índices y se pretende comenzar y finalizar los módulos y categorías con sus opciones de adición, modificación y borrado, además de las de paginación y permisos de usuario propias.

8.9.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	2h	1h	Completa
FR-03 FR-12 FR-08	Core-44	Usuarios con permiso modifican índices asociados a un símbolo	2h	2h	Completa
FR-03 FR-12 FR-08	Core-51	Restringir acciones sobre índices dependiendo de los símbolo asociados	4h	4h	Completa
FR-3.3	Core-52	Modificar índice con símbolo asociado, símbolo pasa a pendiente	4h	4h	Completa
FR-03	Core-53	Marcar índices no validados al validar símbolo	3h	3h	Completa
FR-13 FR-14 FR-15	Core-54	Módulos	6h	6h	Completa
FR-13.1 FR-14.1 FR-15.2	Core-55	Categorías	8h	8h	Incompleta
-	Bug-	Errores de funcionamiento encontrados	3h	3h	Completa
Total			**32h**	**31h**	**7:8**

Tabla 8.10: Sprint 8 backlog.

8.9.3. Sprint review

A la hora de modificar un índice con símbolos asociados, estos símbolos pasan a pendiente.

Se cambia el término valor de índice por elemento de índice.

Se permite poner los valores validados de los índices en los símbolos y comparar que el símbolo tenga todos asociados validados, si no, no se permitirá su validación.

Al borrar el módulo se comprueba que no tenga símbolos asociados, se ha planteado la opción de redirigir los símbolos a otro módulo, pero no se ha llegado a implementar.

Se ha creado una tabla de jerarquías de categorías. No se puede borrar una supercategoría a no ser que no tenga ninguna subcategoría asociada. Se añade un conteo de símbolos y categorías asociado a cada módulo.

Se han solucionado errores a lo hora de añadir selectores de índices en los símbolos. Y se ha solucionado un error que no se actualizaban las referencias a los módulos secundarios de los símbolos. Además, los selectores de modificación de módulos no cambiaban su valor ni en los módulos secundarios.

Se ha hecho limpieza de código y se han eliminado datos mal formados previos, en la creación de la base de datos.

Se ha creado un método para añadir subcategorías dentro de una superior.

Se ha extendido la duración del sprint por temas de coincidencia de una reunión con el sprint planning, este será realizado el viernes.

8.10. Sprint 9

8.10.1. Sprint planning

Dado que este será el último sprint de desarrollo puro se va a priorizar el contenido más urgente, como finalizar algunos ajustes restantes en las categorías.

Por un lado, las búsquedas quedaron bien definidas anteriormente, pero faltaba la parte del servidor. Por otro lado, se tiene que acabar la página de recursos y se asignará contenido a la página de inicio del proyecto, la página general.

Se pretende revisar el uso de las tokens para poder ser usado en cualquier operación y no solo en las relativas a SonarQube.

La gestión de usuarios de los ajustes es un aspecto importante que también tiene que ser abordado.

Finalmente, se quiere modificar el contenido de los índices asociados a los símbolos para no sobrecargar el contenido visual de los mismos.

8.10.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	2h	2h	Completa
FR-13.1 FR-14.1 FR-15.2	Core-55	Categorías	2h	2h	Completa
FR-09 FR-10	Core-56	Recursos	10h	10h	Completa
NFR-09	Core-57	Búsquedas dependiendo de tabulación	12h	14h	Completa
FR-16 FR-17 FR-18 FR-19 FR-20	Core-58	Ajustes de usuarios	12h	14h	Completa
FR-22	Core-59	Página extra para símbolos asociados a índice	2h	2h	Completa
-	Core-60	Página general (Logo y descripción Locomotion)	2h	3h	Completa
NFR-08	Core-61	Cambios autenticación, tokens de sesión	4h	4h	Completa
-	Core-62	Refactorización de código	4h	4h	Completa
-	Bug-	MySQL LIKE no permite la distinción del campo validación en índices	2h	2h	Completa
Total			**52h**	**57h**	**10:10**

Tabla 8.11: Sprint 9 backlog.

8.10.3. Sprint review

Surge la idea de crear una promoción de categorías para las subcategorías. Y se restringe el borrado de categorías a las que no tienen símbolos asociados ni otras subcategorías.

Se crea una tarjeta para mostrar los índices asociados a los símbolos.

Se crean los métodos de búsqueda basados en los métodos de obtención de recursos antiguos. Estas búsquedas están basadas en MySQL y requieren de un formato especial. Se indica que se ha realizado una búsqueda en cada página. Y se cambia en la interfaz el botón de búsqueda flexible por búsqueda exacta para realizar de forma normal la flexible ya que es la más común.

Se crea el formato de la página de inicio, logo, descripción, cabecera y nombre del proyecto, se encapsula en una carpeta y se saca de esta.

Se corrige un error de borrado de índices.

Se crea el contenido de los ajustes, se permite que los usuarios líder de proyecto y supervisor general los administren. Se permite el cambio de contraseña y se crea un sistema de gestión de roles para que se puedan reasignar y crear al añadir usuarios. Se permite el borrado de usuarios. Respecto al resto de usuarios, se permite su modificación de datos de perfil. Ningún usuario puede cambiarse el rol a sí mismo.

Se crea una tabla nueva en la base de datos para añadir la token, su momento de creación y el usuario al que está asociada.

Al realizar acciones se guarda la tabulación para cambiar a ella, por motivos de usabilidad.

Se cambian las reuniones próximas de los sprints a los viernes.

8.11. Sprint 10

8.11.1. Sprint planning

Para finalizar, en este sprint se planea terminar la documentación del proyecto, al igual que la realización de las pruebas pertinentes sobre el sistema para asegurar el funcionamiento del contenido existente. Por lo tanto, no se añadirá código nuevo a menos que sea necesario para la corrección del contenido revisado.

Es necesario realizar una revisión de la documentación previa, redactar los diferentes conceptos anotados y darle forma a la información restante.

8.11.2. Sprint backlog

Historia	Tipo	Descripción Actividad	Tiempo Estimado	Tiempo Empleado	Estado
-	Weekly	Actualizar documentación del sprint	2h	2h	Completa
-	Core-63	Actualizar métodos del plugin SonarQube	3h	3h	Completa
-	Core-64	Actualizar documentación de herramientas y requisitos	6h	6h	Completa
-	Core65-	Actualizar documentación de implementación y pruebas	12h	14h	Completa
-	Core-66	Actualizar documentación de seguimiento y resumen	8h	8h	Completa
-	Core-67	Actualizar documentación de análisis y diseño	8h	14h	Completa
-	Core-68	Revisar y corregir documentación de plan de proyecto	2h	2h	Completa
-	Core-69	Reescribir introducción	3h	3h	Completa
-	Core-70	Referencias y Bibliografía	4h	4h	Completa
-	Core-71	Escribir manuales	6h	12h	Completa
-	Core-72	Corregir errores de código	6h	8h	Completa
-	Core-	Formato de tablas y figuras	3h	3h	Completa
Total			**63**	**79**	**12:12**

Tabla 8.12: Sprint 10 backlog.

8.11.3. Sprint review

En este sprint final se han empleado las horas necesarias para terminar la documentación del proyecto e intentar poner en orden las prioridades futuras del mismo.

Al mismo tiempo se han realizado pruebas sobre el código y se han detectado varios errores arrastrados del desarrollo inicial.

Se detecta que Associated symbols en index se cuenta mal.

Se encuentra un error en índices de borrado y se comprueba todo lo relativo a estos.

Se ha encontrado el error de comprobación de longitud de las peticiones y si existe el contenido enviado. Ninguna de las dos comprobaciones se realizan en este momento. Quedarán

anotadas en el Capítulo de pruebas (ver Sección 7.2.4), en errores encontrados.

Se determina hacer prueba con base de datos vacía de como sería el sistema en un principio al arrancar.

Se detecta que para los usuarios programador de módulo no se carga bien su perfil en settings.

8.12. Resumen y costes

En este apartado se realizará un análisis post desarrollo de la calendarización real y horas totales del proyecto así como de los riesgos que se han presentado durante el transcurso del mismo.

8.12.1. Incidencias

En general las incidencias han sido anotadas a lo largo de los sprints de seguimiento con la etiqueta "Bug". Dado que todos ellos han sido ya comentados, estos representan solo los errores que han supuesto una inversión de tiempo mayor, pero todos ellos han sido finalmente resueltos.

Se debería destacar uno no mencionado, que representa a las alertas del sistema relacionadas con el éxito de las peticiones o los errores. Estas no se muestran tras cargar las vistas y no se ha llegado a solucionar dado que probablemente habría que registrar estas alertas dentro del código de la puerta de enlace y enviarlas para ser mostradas tras la carga de la nueva página.

8.12.2. Resumen

En cuanto a la preparación del proyecto, en principio iba a estar dirigido a backend, por lo que hubo que buscar software para desarrollar el frontend sobre la marcha cerca del final del sprint 0.

Los primeros tres sprints de desarrollo dejan claro que el ejemplo inicial no fue lo suficientemente completo como para solventar muchas dudas al inicio, y por lo tanto se generaron pérdidas de tiempo.

En cuanto al desarrollo del proyecto, la mayoría del mismo ha sucedido en periodo de cuarentena debido al CoVid-19, como se comentó anteriormente, y esto retrasó el trabajo de dos sprints debido a los cambios en el entorno de trabajo y la situación social vivida.

La falta de experiencia del equipo de desarrollo se ha notado bastante, ya que se han tenido que realizar varias vueltas a código antiguo para reescribirlo en un formato más correcto. Por la misma razón, se podía haber enfocado mejor el tiempo en cuanto a contenidos prioritarios.

Finalmente, el equipo de desarrollo se pudo adaptar a los cambios del entorno y se aumentó el ritmo de producción, necesitando realizar muy pocas búsquedas y pudiendo centrarse en las partes restantes del proyecto.

Debido a los retrasos acumulados en los sprints no se ha podido completar el 100 % de los requisitos anotados, pero sí la mayor parte de ellos.

Los no abordados son los siguientes:

- NFR-10, el proyecto no se ha concluido por lo que no será desplegado en un entorno real, solo en local.
- FR-21, los valores de "ProjectTypeOfValue" y "ProgrammingLanguageSymbolType", junto con la modificación y comprobación de sus asociaciones no han sido finalmente abordados aunque se encuentran introducidos en la base de datos de forma estática, y pueden verse en los ajustes.
- FR-15.1 y FR-15.2.1, la reasignación de símbolos al borrar un módulo o categoría.
- FR-11, este requisito no era prioritario en ningún caso y se planteó para el futuro del sistema, la creación de un archivo Word legible es más bien compleja en el aspecto de creación del propio archivo ya que la información ya se encuentra organizada en el diccionario de datos del propio sistema.
- FR-07, se llegó a buscar un framework de notificaciones pero no se encontró uno que encajara con el sistema actual, por lo que se dejó para su implementación manual. Por falta de tiempo esto nunca llegó a suceder.

8.12.3. Costes reales y costes simulados

Coste simulado final

En cuanto al coste de recursos simulado que se realizó inicialmente, se preveían 360 horas de desarrollo y se han realizado finalmente 374 horas y 30 minutos, incrementando el coste en 14 horas y media: aumentando el coste total en 135€. Por otro lado el sprint final de documentación ha ocupado 79 horas frente a las 40 asignadas en un principio en la previsión y por lo tanto un coste añadido de 39 horas que traducidas a Euros son: 363€.

Al no estar completo, el proyecto tendría un coste añadido adicional, esto quiere decir que en unas 60 horas, tiempo que se estima para su finalización, se añadiría a su vez un coste estimado de 558€.

Finalmente, el coste total añadido sería de: 135€, derivados del desarrollo, + 363€, derivados de la documentación, + 558€, derivados de la predicción del tiempo necesario para acabar la funcionalidad del proyecto.

Haciendo esto un total de 1.056€, de coste añadido para la finalización del proyecto. En otras palabras: 4.650€+ 1.056€= 5.706€, de coste global simulado.

Coste real final

Efectivamente el coste material y de software se ha mantenido en 0€, como se había previsto en la sección **3.3**, ya que se han utilizado materiales propios del equipo de desarrollo y software libre durante todo el transcurso del proyecto.

El período a lo largo del cual se pagó la beca, teniendo en cuenta que se admitió la prórroga de la misma, llega a un total de 9 mensualidades ascendiendo el total a 2.700€. Siendo este el único coste real.

Por otro lado, si contamos con las horas restantes previstas para la finalización del proyecto (60 horas), y dada la asignación de horas por sprint(40 horas), que se hizo en la planificación. Se añadiría un coste adicional de 2 mensualidades a la finalización del proyecto, sumando así 600 €, al total.

Finalmente se tendría el coste añadido de 600€, ya que se predijo julio como fecha de finalización inicial. En otras palabras, 2.700€+ 600€= 3.300€, de coste global real.

Capítulo 9

Conclusiones y Trabajo futuro

Los objetivos generales de este TFG se han cumplido satisfactoriamente, quedando solamente algunos requisitos sin realizar. Algunos de estos requisitos estaba previsto no tratarlos en este TFG. Otros se han descartado por el gran volumen de trabajo y tiempo dedicado al TFG. Pero este proyecto sirve de base sólida para la continuación del original.

A continuación, se van a presentar las conclusiones personales sobre este TFG y, posteriormente, se describe el trabajo futuro que he podido llegar a ver como prioritario. Este trabajo futuro se podrá llevar a cabo tras la lectura de esta memoria, pero también necesitará un esfuerzo de comprensión de código y una revisión a algunas de las referencias bibliográficas más relevantes no mencionadas: sobre puertas de enlace: [26], sobre endpoints en Flask [21].

9.1. Conclusiones

Si tuviera que enumerar las cosas que he aprendido a lo largo del proyecto destacaría unas cuantas que me han hecho ver la realidad de un desarrollo puro basado en la investigación y partiendo de ningún conocimiento previo.

En primer lugar, el diseño y la elección de una arquitectura a la hora de pensar en la implementación es uno de los aspectos que más he notado ejercer su peso durante el transcurso del desarrollo. En otras palabras, antes de comenzar con la implementación del código tener en mente patrones para gestionar cada parte del sistema. Lo menciono en especial por el patrón de acceso a datos que cambió radicalmente la limpieza del código.

En segundo lugar, he aprendido a cerca de la metodología ágil que no la había usado en proyectos reales hasta el momento y he de decir que el sistema de gestión del trabajo se hace mucho más llevadero cuando se divide y se reparte de forma adecuada. La contra-parte de esta metodología se encuentra en la planificación que es bastante compleja de manejar correctamente, resultando en parte en la no finalización completa de los objetivos del proyecto.

En tercer lugar, he aprendido a cerca del funcionamiento de diversas tecnologías que desconocía, en especial sobre microservicios y despliegues de sistemas que en general ya es algo muy bueno de por sí. Si sumamos el uso de colas de mensajes, de estados de sesión y lo aprendido de cara a envíos de peticiones y JSON, me quedo más que satisfecho.

En cuarto lugar, el hecho de teletrabajar durante la gran mayoría del trabajo me ha creado una sensación de incertidumbre sobre el mismo y ha supuesto un reto del que puedo llegar a sacar mucho beneficio en el futuro.

Para finalizar, cabe mencionar que la motivación extra que supone desarrollar un sistema que en un futuro puede ser utilizado por mucha gente, y sobre todo con un buen propósito como es el de cuidar el medio ambiente y alertar sobre el estado de nuestra civilización, me parece algo increíble y realmente significa mucho a la hora de trabajar y plantear nuevas ideas o intercambiar opiniones con los miembros del proyecto.

9.2. Trabajo futuro

Dado que el proyecto no ha quedado completamente finalizado se pueden destacar varios puntos de trabajo que en un futuro serían prioritarios para el correcto desarrollo del sistema.

Fundamentalmente, la conclusión de los requisitos pendientes es uno de los aspectos primordiales a tratar en el futuro. Estos se han mencionado anteriormente en las conclusiones del seguimiento (ver apartado **8.12.2**), especialmente habría que emplear trabajo en el despliegue del sistema de forma real mediante Kubernetes en nodos distribuidos o en caso de solo desplegarlo en un nodo mediante Nginx tal y como se mencionó en el apartado de configuración (ver apartado **7.1.2**).

Si se va a desplegar el sistema bajo un dominio específico se deberían configurar los archivos JavaScript y crear una variable que represente al dominio para que este se pueda modificar fácilmente en las llamadas a los métodos REST, lo más funcional sería que la compartan variables de entorno.

Por parte del resto de requisitos, no habrá demasiado problema a la hora de implementarlos, quizás el requisito de notificaciones a clientes puede ser problemático, pero se puede optar por su implementación manual o mediante un framework especializado. Al mismo tiempo, se puede extender el sistema de notificaciones para usar los correos electrónicos, aunque habría que comprobar su corrección y existencia ya que ahora mismo no se realizan.

En cuanto a las variables de programación del proyecto y de Vensim, se debería habilitar una forma de relacionar ambas y comprobar que su relación se cumpla a la hora de asignarlas a los símbolos. Por supuesto, se deberían de poder modificar borrar y añadir, cosa que no se permite en este momento.

Pasada la implementación de requisitos, se deberían de realizar reorganizaciones de código y correcciones de errores mencionados en **7.2.4**, para comprimir código que sea compartido entre varias funciones y se deberían establecer módulos separados para las distintas funciones

del sistema. Con esto se hace referencia a, por ejemplo, en el microservicio de símbolos se han dejado muchas funciones y aunque tienen separadores que las delimitan se pueden modularizar para facilitar su lectura.

Para conseguir que el sistema esté finalmente bien modularizado este proceso se puede llevar a cabo en la puerta de enlace de forma que el punto de acceso quede intacto pero se establezca una jerarquía mucho más organizada.

En adición a lo previamente mencionado, un balanceo de los microservicios, reasignar la amplitud de cada uno especialmente del de símbolos, y una división de la base de datos por microservicio sería necesaria en caso de querer optimizar el sistema.

En cuanto al carácter estético, se pueden llegar a mejorar muchos aspectos relacionados con la interfaz y la colocación de los elementos visuales, pero eso dependerá del plan que haya para su reescritura. En lo que concierne a errores o más bien a cambios en su funcionamiento, en el apartado **8.12.1**, queda descrito el mal funcionamiento de algunas alertas.

Por último, la automatización y pruebas del sistema debería ser hecha siguiendo el modelo DevOps, que se intentó aplicar al comienzo del proyecto, pero que finalmente se abandonó. Se hace referencia al proceso de subida a GitLab, sus pruebas automáticas unitarias, de integración y end-to-end, y finalmente su despliegue. Para esto se deberían usar las pipelines de GitLab y un archivo ".gitlab-ci.yml" que sea el que organice las etapas mencionadas.

Apéndice A

Manuales

En este anexo, se trataran los diferentes pasos a seguir y archivos a tener en cuenta para un correcto uso del sistema, desde su despliegue pasando por su mantenimiento, y el uso por los usuarios finales.

Es preciso mencionar que durante el transcurso de la memoria entera se han abarcado muchos aspectos que hacen del conocimiento aportado en esta sección completo, y que se harán referencia en caso de ser considerado necesario.

A.1. Manual de despliegue

Durante toda la memoria se ha mencionado el procedimiento de despliegue basado en Docker, e incluso su instalación. Por lo tanto, no queda nada que añadir en este apartado. En el apartado **7.1.1** se aborda la instalación y despliegue del sistema probado en una máquina virtual nueva.

En resumen, el comando "docker-compose up" es el que despliega el sistema sobre la máquina. Para ello necesitaremos tener instalado en el sistema Docker, Python3 y netcat.

A.2. Manual de mantenimiento

Para realizar el mantenimiento del sistema, hay varios aspectos a tratar, donde el primero de todos, sería la lectura de la memoria, para comprender las decisiones tomadas a lo largo de la planificación y desarrollo del proyecto.

Dentro de esta memoria, se han descrito ya numerosos mecanismos para mejorar el código, recientemente tratados en la Sección **9.2**. Este apartado debería dar suficientes pistas sobre

posibles modificaciones y mejoras funcionales pendientes a realizar, pero en caso de querer actualizar el sistema queda pendiente determinar el mecanismo por el cuál obtenemos las versiones de las diferentes herramientas que se han utilizado.

En cuanto al versionado de componentes o su sustitución:

- .env
- run.sh
- requirements.txt
- config.yaml
- docker-compose.yml
- Dockerfiles

Vamos a verlos uno por uno acompañados de imágenes de ejemplo. Cabe mencionar que todos estos archivos han sido previamente descritos en **7.1.3**.

Para empezar, se pueden distinguir dos tipos de contenido en estos archivos, pero todos son variables de entorno que se utilizan en el nivel en el que se encuentran.

Se debe destacar que el **.env** de la carpeta **src** es el que se usa para asignar los valores de RabbitMQ, y que en el resto se establecen las conexiones y en caso de la gateway la clave secreta de sesión.

```
AMQP_URI='amqp://user:pass@rabbit/'
SECRET_KEY='super secret key'
```

Figura A.1: Variables de entorno Gateway.

```
RABBITMQ_DEFAULT_USER=user
RABBITMQ_DEFAULT_PASS=pass
```

Figura A.2: Variables de entorno Docker-Compose.

Continuamos con **run.sh**, este archivo ejecuta un bucle de espera para el servicio RabbitMQ y acto seguido, tras esperar 5 segundos para la inicialización de la base de datos, aunque en general tarda más cuando se genera por primera vez, finalmente se ejecuta el proceso bajo el servicio nameko y la configuración elegida. Veremos el ejemplo del microservicio de recursos.

En cuanto a versiones utilizadas, **requirements.txt** contiene cada librería necesaria para el contenedor docker y que el sistema funcione, si queremos cambiar las versiones no debería

```
until nc -z rabbit 5672; do
    echo "$(date) - waiting for rabbitmq..."
    sleep 1
done

sleep 5
nameko run --config ./config.yaml resources_worker
```

Figura A.3: Composición run.sh

haber problemas de incompatibilidad, dicho esto, las versiones son las indicadas tras el doble igual, y las que no lo tienen representan a la última versión.

```
nameko==2.11.0
psycopg2-binary==2.7.6.1
python-dotenv==0.9.1
PyMySQL
PyMySQL[rsa]
```

Figura A.4: Composición requirements.txt

Por su parte, el archivo **config.yaml** contiene la configuración que utiliza nameko para levantar los microservicios, en general el archivo es bastante auto-descriptivo y para saber más consultar REFERENCIA A NAMEKO CONFIG, en resumen, establece el límite de trabajadores y configura la dirección del servidor para recibir peticiones de este.

```yaml
AMQP_URI: 'pyamqp://user:pass@rabbit'
WEB_SERVER_ADDRESS: '0.0.0.0:8000'
rpc_exchange: 'nameko-rpc'
max_workers: 10
parent_calls_tracked: 10

LOGGING:
  version: 1
  handlers:
    console:
      class: logging.StreamHandler
  root:
    level: DEBUG
    handlers: [console]
```

Figura A.5: Composición config.yaml

Sobre el archivo **docker-compose.yml**, ya se ha hablado bastante en la memoria, por lo que con lo tratado en el apartado **5.3.1**, es suficiente para describirlo.

Por último, los **Dockerfiles** son archivos vinculados con los comandos que sigue el "docker-compose up" a la hora de levantar el servicio de forma previa, en ellos, se define la instalación de requisitos y actualización de archivos necesarios para los contenedores. Así como el archivo de ejecución para levantar el servicio dentro del contenedor.

```
FROM python:3.7

RUN apt-get update && apt-get -y install netcat && apt-get clean

WORKDIR /app

COPY . .

RUN chmod +x ./run*.sh

RUN pip install --no-cache-dir -r ./requirements.txt
```

Figura A.6: Composición Dockerfiles

A.3. Manual de usuario

Esta sección está dedicada tanto al uso de la aplicación web como por consola, se mostrarán varios ejemplos prácticos y se mostrarán las vistas creadas.

Para comenzar, se mostrarán las vistas y su uso, y se terminará listando los comandos para acceder a la funcionalidad mediante consola.

A.3.1. Uso de las vistas

Bajo la ruta /, encontramos el login, que sirve para acceder al sistema identificándose como un usuario registrado, o como invitado con funciones reducidas, únicamente como espectador.

Se deberá introducir el nombre de usuario en el campo superior y la contraseña en el inferior y pulsar sobre "Identify me!", en caso de estar registrado. En caso contrario, con hacer click sobre "Log in as guest!", nos identificaremos como usuario invitado.

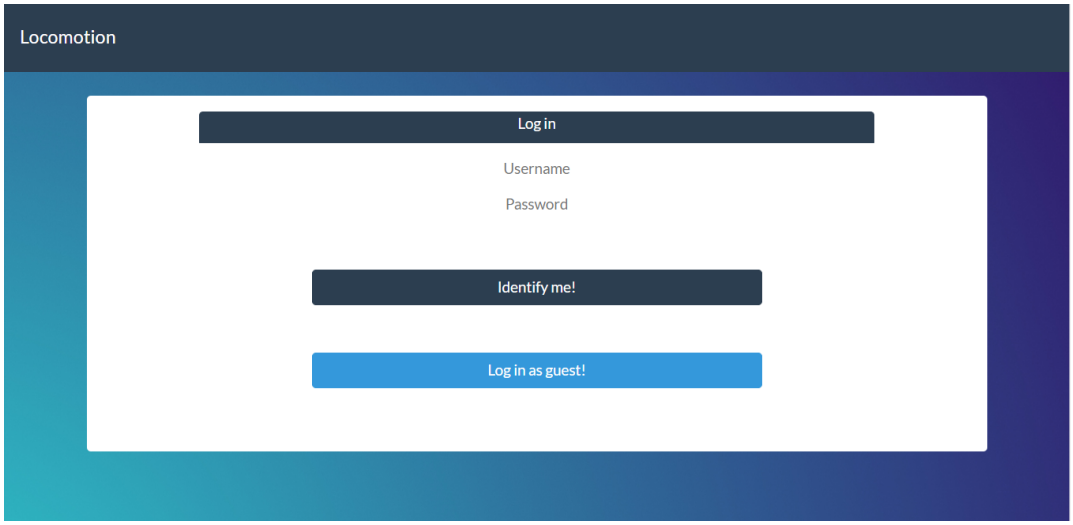

Figura A.7: Vista Login

Tras acceder al sistema, lo primero que se verá será la página general bajo **/general**, en esta página se muestran tanto el logo como el título del proyecto, su encabezado y su descripción justo a continuación.

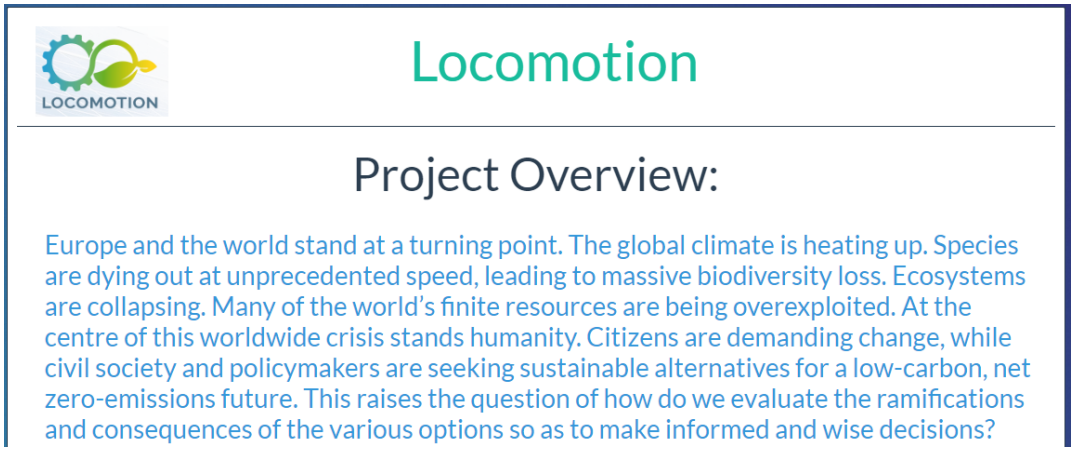

Figura A.8: Vista General

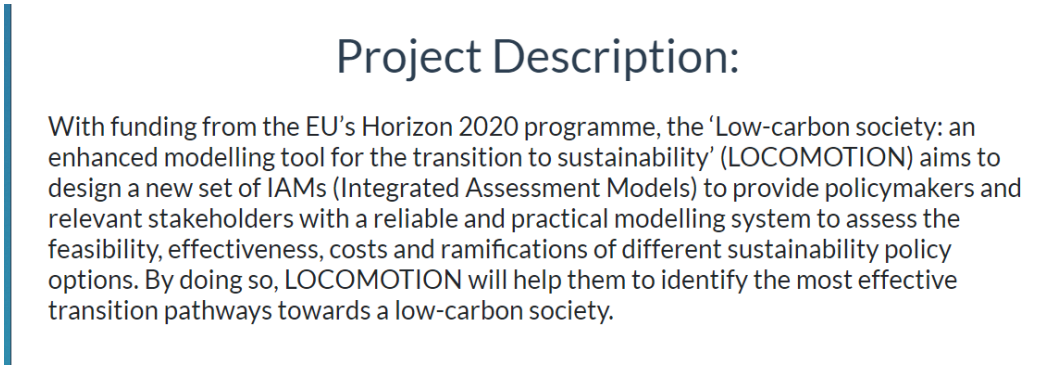

Figura A.9: Vista General Descripción

Al mismo tiempo que accedemos al sistema una barra de navegación aparecerá durante todo el tiempo en la parte superior de la interfaz, esta barra de navegación nos permite acceder a los diferentes recursos del sistema, realizar búsquedas flexibles o exactas marcando el switch, y finalmente al hacer click sobre el usuario, en este caso Admin, se puede acceder al menú de ajustes, notificaciones(no funcional), y a salir de la sesión.

Si recorremos de izquierda a derecha cada una de las opciones, tendríamos lo siguiente:

- "Locomotion" conectado a **/general**, lleva a la página de inicio.
- "Resources" conectado a **/resources**, lleva a los recursos del diccionario.
- "Modules" conectado a **/modules**, lleva a los módulos del diccionario.
- "Symbols" conectado a **/symbols**, lleva a los símbolos del diccionario.

- "Categories" conectado a **/categories**, lleva a las categorías del diccionario.
- "Indexes" conectado a **/indexes**, lleva a los índices del diccionario.
- "Exact Switch" botón que indica si se hace búsqueda exacta, desactivado por defecto.
- "Search box" campo de texto para introducir la cadena a buscar.
- "Search button" botón que desencadena la búsqueda dependiendo de dónde nos encontremos dentro del sistema.
- "Menu" botón que despliega las opciones del menú de usuario.

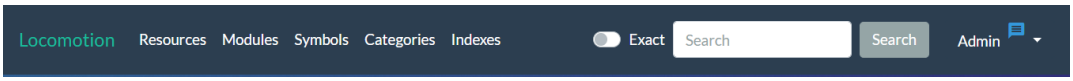

Figura A.10: Vista Navbar

Este sería el ejemplo del menú de usuario desplegado, al hacer click sobre "Settings" iríamos a los ajustes, y sobre "Log out" saldríamos del sistema, "Notifications" no tiene una vista asociada ni enlace. En caso de ser usuarios invitados solo veríamos la opción para salir del sistema.

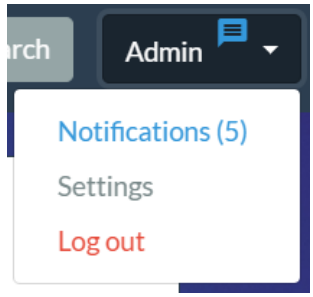

Figura A.11: Vista Menú de ajustes

Si accedemos a **/resources**, aparecerá una página similar a la mostrada a continuación. Podemos distinguir varios aspectos:

- 3 tabulaciones que separan el contenido de los acrónimos, adjetivos y reglas semánticas.
- añadir recurso, que dependiendo de la tabulación cambia, acrónimo en este caso, "Add acronym".
- Un grupo de botones que representan la paginación de la vista actual. Profundizaremos en ellos un poco más adelante REFERENCIA.
- Una tabla con el conjunto de recursos consultados.

- Botones de acción a la derecha de cada fila de la tabla.

Los botones de acción y añadir se desactivan para los usuarios que no tengan permisos, en este caso solo el líder de proyecto y supervisores generales tienen permiso para usarlos. En el caso de los usuarios invitados, estos no ven los botones de acción, ni el de añadir.

A la hora de añadir, modificar y borrar recursos, todos los formularios serán llevados en cartas como las que veremos un poco más adelante, pero al no tener nada de especial no merece la pena mostrarlas como tal.

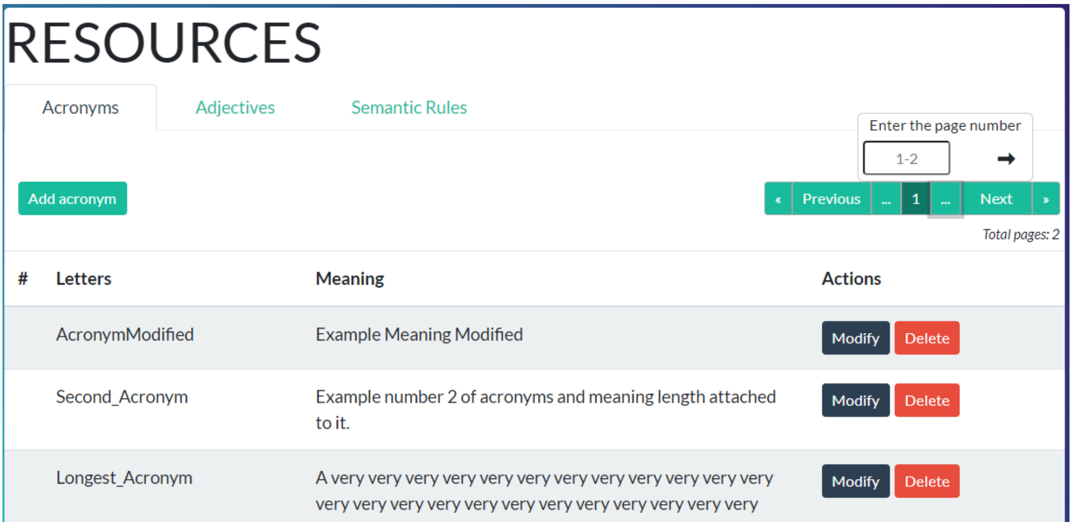

Figura A.12: Vista Recursos

En caso de acceder a **/modules**, aparecerá una página similar a la siguiente.

En ella podemos distinguir los mismos elementos que en la de recursos anteriormente mencionada. La columna "Symbols" de la tabla hace referencia al número de símbolos asociados a cada módulo, solo se podrá borrar un módulo en caso de no tener ninguno asociado.

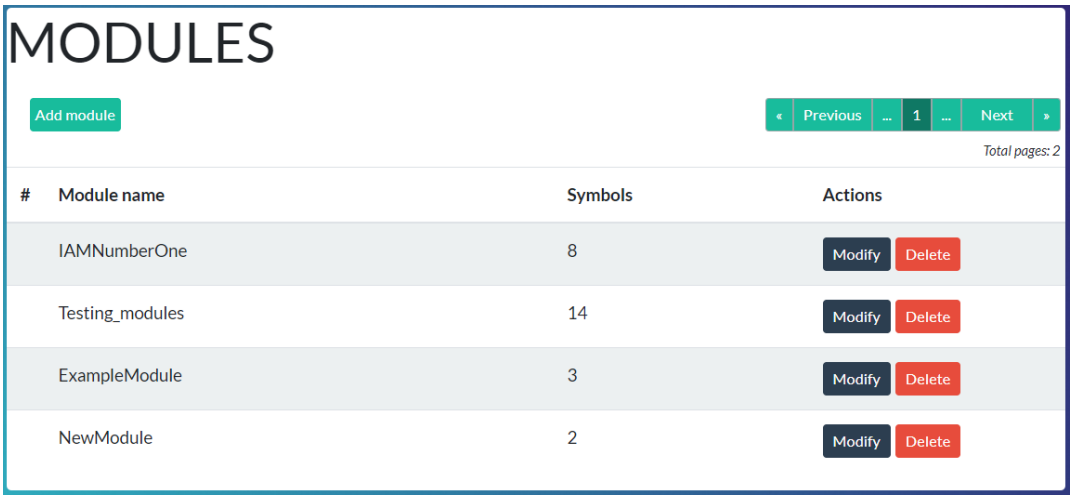

Figura A.13: Vista Módulos

La página de categorías **/categories**, es en la que se muestra la jerarquía categoría-subcategoría. Dicha jerarquía se representa mediante un desplegable y en cada fila ya sea de categoría o subcategoría se muestran los símbolos asociados a la categoría objetivo.

Las categorías de nivel menor pueden ser promocionadas en su modificación, y las superiores cuentan con un contador de subcategorías. Para añadir una subcategoría deberemos acceder a una categoría principal y en el desplegable hacer click sobre el elemento azul "SubCategories".

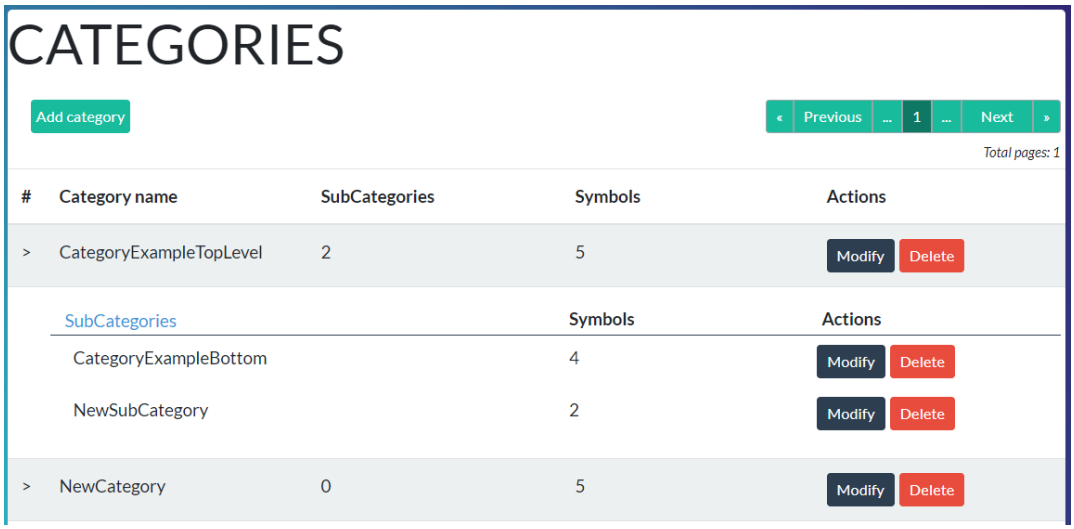

Figura A.14: Vista Categorías

Hacer click en el enlace de color azul de "SubCategories" abriría una carta como la siguiente, en la que se muestran las ya añadidas, el botón "+" sirve para añadir campos de texto. Se ignorarán los que sean iguales y los que ya estén definidos en el diccionario de datos. En caso de querer dejar un campo en blanco no habrá problema ya que será ignorado.

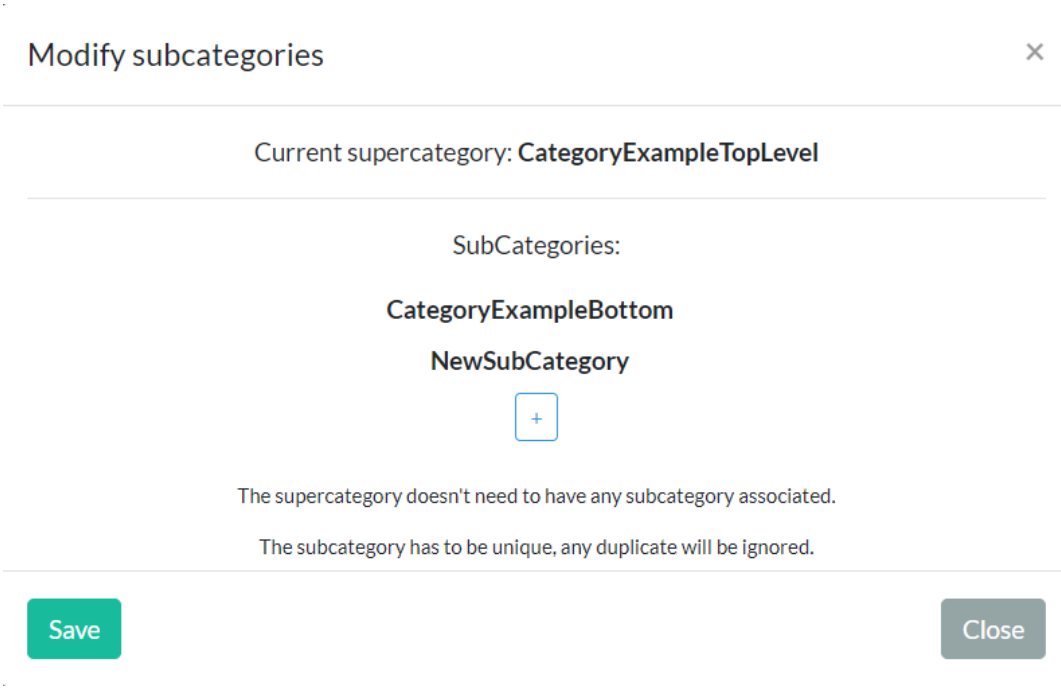

Figura A.15: Vista añadir Subcategoría

En cuanto a la modificación de categorías será realizada en una carta, el botón "Promote" es el que promocionará las categorías secundarias a principales. Y el botón "Modify" enviará la petición.

Añadir una categoría principal tiene el mismo formato que la modificación sin promoción.

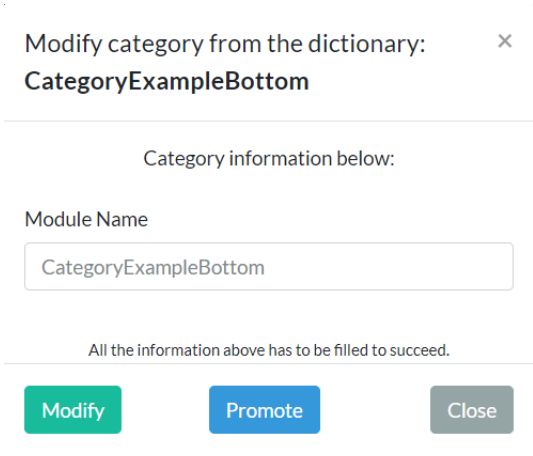

Figura A.16: Vista modificar Subcategoría

Por último, el borrado de categorías tiene un formato similar al de la carta que se mostrará a continuación. Cabe destacar lo anteriormente mencionado de que una categoría no puede tener subcategorías ni símbolos asociados. El botón "Delete" enviará la petición.

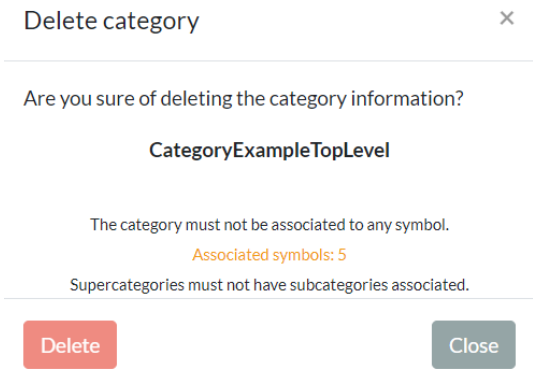

Figura A.17: Vista borrar Subcategoría

Por parte de los índices **/indexes**, tienen una función similar a la de las categorías en cuanto a modificación y añadir índices nuevos. Ya que se accede mediante los botones que se han descrito anteriormente, pero ahora tendrían referencia a los índices, no se hará hincapié en ellos.

En cuanto al borrado de índices es igual que el de categorías pero se borrarán los elementos asociados y no tendrá que tener ningún símbolo para poder ser enviada. Los símbolos se tienen que desvincular desde la propia pestaña de símbolos en el apartado que se mostrará un poco más adelante.

Por último, cabe destacar las tabulaciones de validados y pendientes que se encuentran tanto en los índices como en los símbolos.

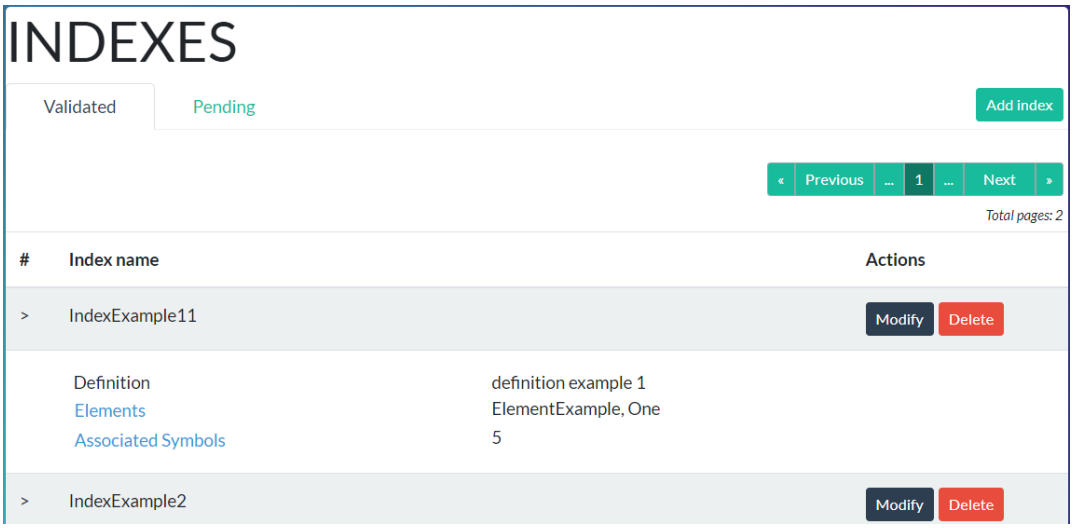

Figura A.18: Vista Indexes

Los símbolos asociados a cada índice se mostrarán de forma numérica en la fila "Associated Symbols", al hacer click sobre la etiqueta se abrirá una carta con la lista de nombres de los símbolos para identificarlos rápidamente. A continuación la carta de símbolos asociados.

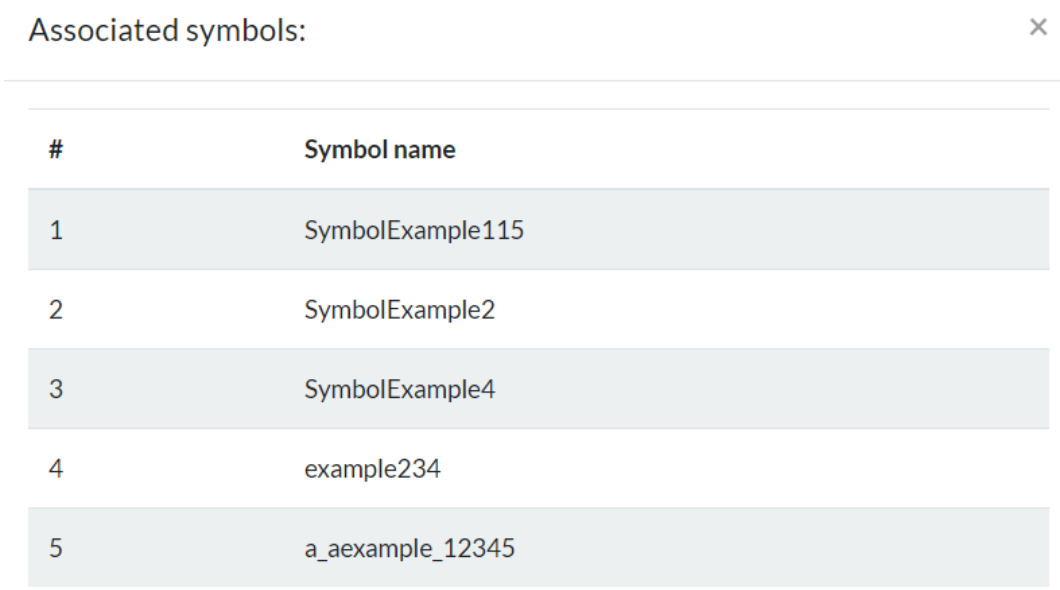

Figura A.19: Vista Símbolos asociados

Los elementos asociados a un índice funcionan de forma similar a las subcategorías anteriormente mencionadas, el único detalle a tener en cuenta de forma adicional es el de los botones rojos que eliminan los elementos actuales.

En caso de no querer eliminar un elemento que se ha quitado al presionar el botón, con salir de la pestaña con un click en la "X" superior derecha, en "Close" o en cualquier lugar fuera de la carta, no se guardarán las modificaciones realizadas.

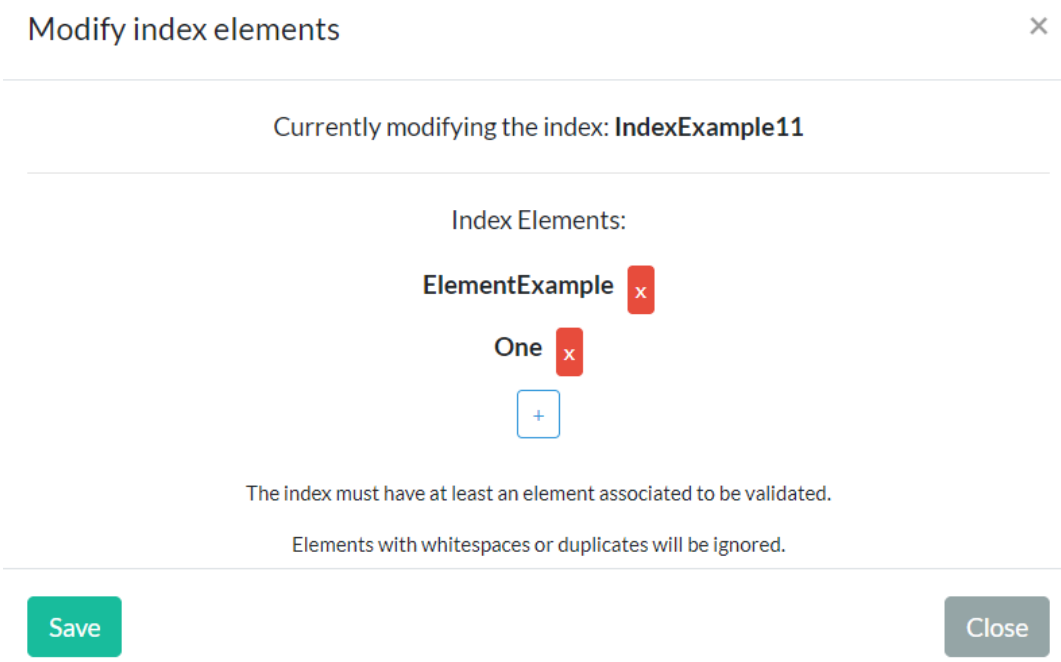

Figura A.20: Vista Elementos del índice

Finalmente, los índices pendientes pueden ser validados y para ello necesitar tener un elemento o más asociados.

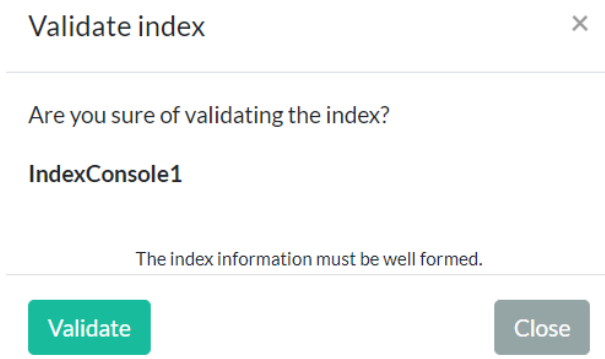

Figura A.21: Vista Validación índice

Por último, la parte más compleja de recursos del diccionario es la de símbolos.

Su estructura es la misma que la de los índices en cuanto a la vista, pero en el desplegable se añaden varias funcionalidades extra.

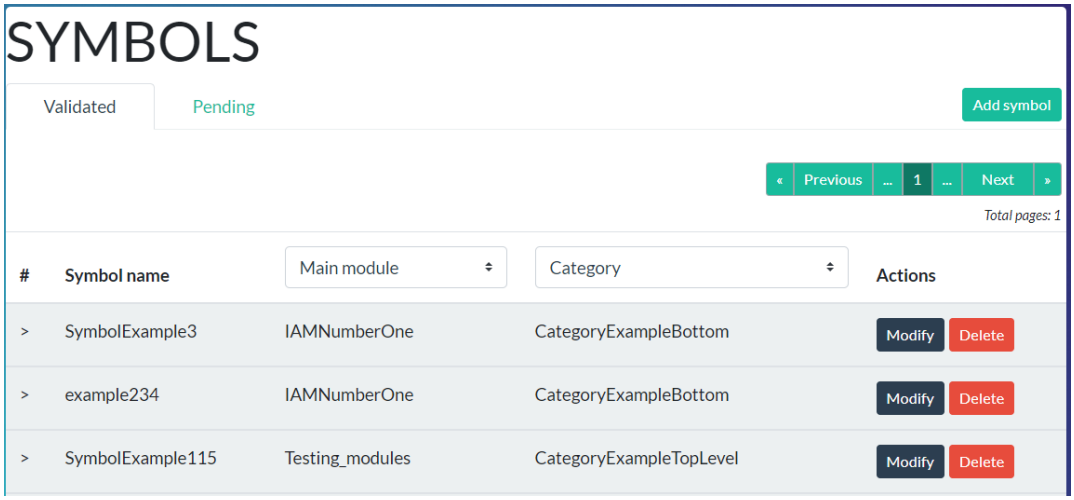

Figura A.22: Vista Símbolos

La única diferencia destacable con el resto de vistas de recursos, es la existencia de unos filtros en la cabecera de las tablas, también existente en los pendientes.

Esto ocultará los símbolos cuyo valor no coincida con el seleccionado, y se pueden usar

ambos al mismo tiempo. Representan el módulo principal y la categoría, que son los elementos filtrados respectivamente.

A continuación se mostrará un ejemplo de filtrado por módulo principal, y la lista de selección disponible en las categorías.

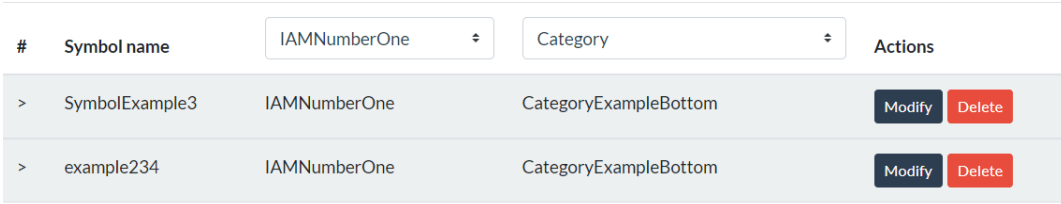

Figura A.23: Vista Filtro símbolos

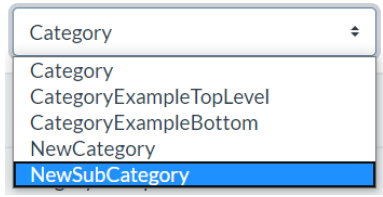

Figura A.24: Vista Filtro símbolos lista

Por otro lado, como se ha mencionado anteriormente, el desplegable de los símbolos es mucho mayor que el del resto en contenido.

Tiene dos cartas que se muestran al hacer click sobre los enlaces azules, uno de índices asociados, y otro de módulos secundarios.

Por su parte, los símbolos pendientes tienen el botón de validación al igual que los índices, y el único escenario en el que no se podrían validar sería en el que el símbolo se encuentre con el valor “Is indexed” en verdadero y sin ningún índice asociado.

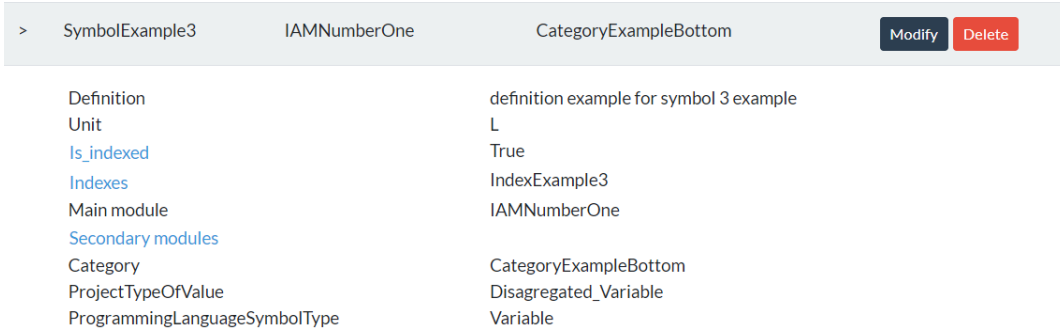

Figura A.25: Vista desplegable Símbolo

Si se hace click sobre "Is indexed" o "Indexes", se abrirá la carta de índices asociados al símbolo. En caso de "Is indexed" estar marcado como falso, el cambio de "Is indexed" solo puede realizarla un supervisor general o el líder del proyecto, no se podrán modificar los índices asociados, en concreto no habrá ninguno.

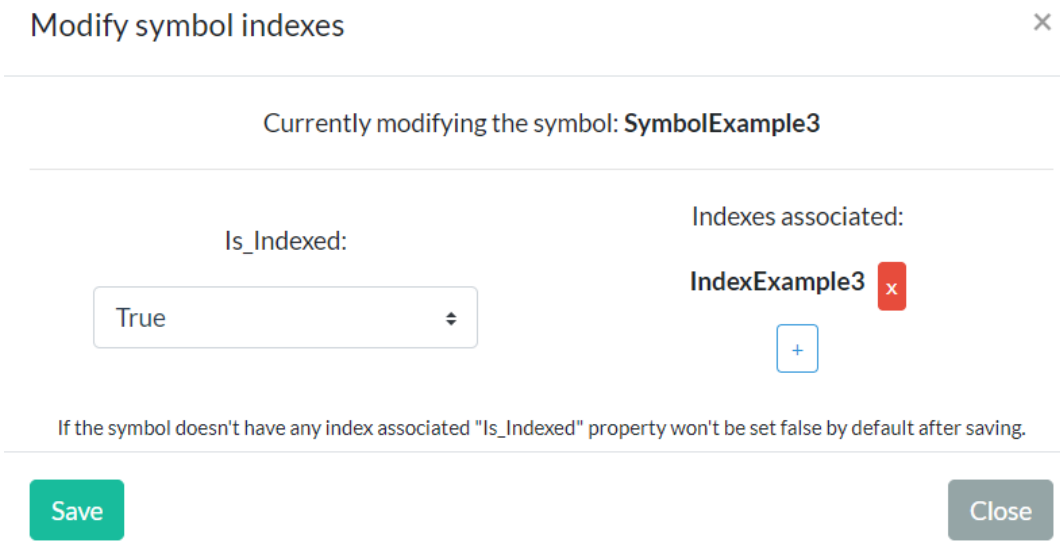

Figura A.26: Vista Indices asociados al símbolo

Los símbolos secundarios funcionan de forma similar a los índices asociados. El módulo principal se puede modificar desde el botón de modificar asociado a la fila del símbolo pero no desde la carta, al menos no por usuarios distintos a supervisores generales o líder de proyecto.

Los módulos secundarios siguen el mismo formato que en los ejemplos anteriores.

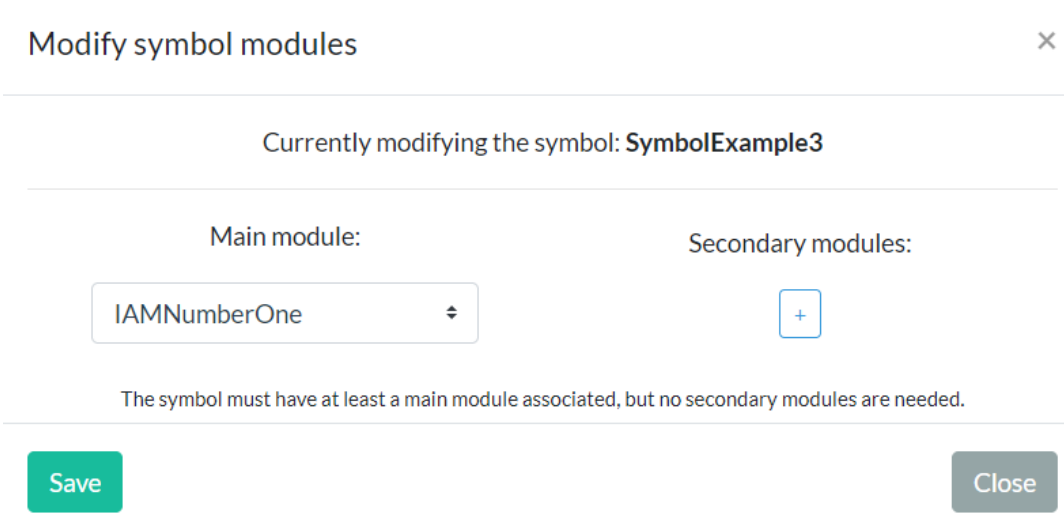

Figura A.27: Vista Módulos secundarios asociados al símbolo

El formato para añadir cualquiera de los valores anteriormente presentado en las últimas dos cartas, es similar al que se va a mostrar a continuación.

En este ejemplo, se quedará el “IndexExample3” y se añadirá “IndexExample2” al símbolo, el último “IndexExample3” sirve de ejemplo para mostrar que se puede volver a seleccionar el mismo, pero los duplicados serán ignorados.

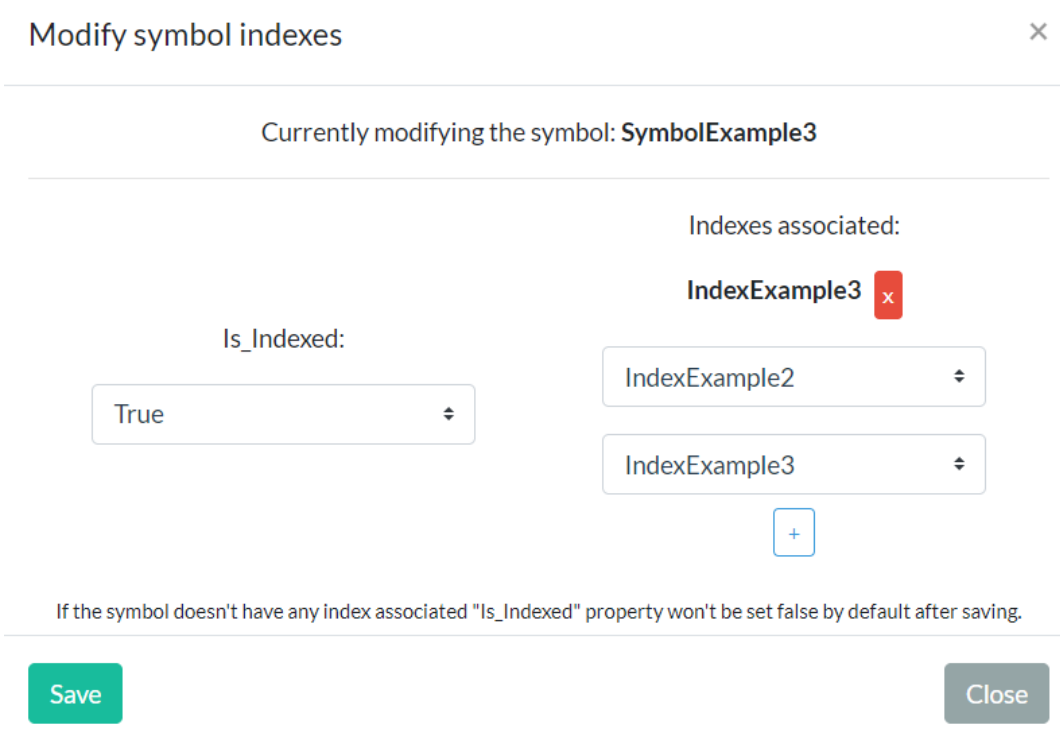

Figura A.28: Vista ejemplo modificación de índices asociados

A.3.2. Uso de consola

Todas las peticiones anteriores y tienen una estructura para ser enviadas al servidor. Este es el tema que vamos a tratar en este apartado. Se debe anotar que el servidor solo ha sido probado en local, y por lo tanto su dominio será el asignado por defecto: **localhost:8080**.

Para empezar, todas las peticiones están realizadas mediante el comando "curl", y añadiendo contenido JSON a las mismas. Además, cada petición se realiza bajo **CRUD**, las operaciones:

- Create, representado por la cabecera en el mensaje POST
- Read, representado por la cabecera en el mensaje GET
- Update, representado por la cabecera en el mensaje PUT
- Delete, representado por la cabecera en el mensaje DELETE

Por lo tanto, como primer ejemplo de comando se tendría algo similar a:

```
curl -H "Content-Type: application/json" -r POST \
-d ‘{"username":"usernameHere","password":"passwordHere"}’ \
‘http://localhost:8080/login’
```

Este caso es el de **/login**, paso previo necesario para acreditarnos en el sistema en caso de no estarlo o de que la token que hayamos obtenido previamente haya expirado. Obtendríamos una respuesta en caso de introducir las credenciales correctas con la token asignada a nuestra sesión de consola.

En este caso utilizaremos una token producida por el sistema: “5UAZaIpuZfqZFDdr2EZj”

El resto de las peticiones que listaremos a continuación necesitan la token para identificar al usuario. Se añadirá como si fuera uno de los campos previamente mencionados de nombre de usuario o contraseña.

En las siguientes listas anotaremos las rutas y los datos que necesita cada una para llevarse a cabo. Formato: ruta — datos

Las rutas de adición de recursos bajo POST:

- /add/symbol — name, definition, unit, module, category, ProjectTypeOfValue, ProgrammingLanguageSymbolType
- /add/index — name, definition
- /add/module — name
- /add/category — name
- /add/subcategory — supercategory, categories
- /add/acronym — letters, meaning
- /add/adjective — adjective, usages
- /add/semantic_rule — ruleforShort, explanation
- /add/user — username, fullname, email, password, roles

El tipo de datos de cada uno de los nombrados es el siguiente, los definidos bajo “Otros”, representan las listas formadas de forma especial y por ello se definirán entre paréntesis a continuación de los mismos:

- Cadena de texto(String): name, definition, unit, module, category, ProjectTypeOfValue, ProgrammingLanguageSymbolType, supercategory, letters, meaning, adjective, usages, ruleforShort, explanation, username, fullname, email, password.
- Otros(ArrayList): categories(String), roles(Dictionary(role(String), module(String)) **2.1**).

La respuesta a cualquier petición con el código **200**, seguido de un diccionario con la clave " 'success' : true", significará que se ha llevado a cabo la acción correctamente.

Por lo tanto, como ejemplo, vamos a realizar la petición a **/add/symbols** añadiendo sus diferentes datos:

```
curl -X POST 'http://localhost:8080/add/symbol' /
-H "content-type: application/json" -d '{"token": "5UAZaIpuZfqZFDdr2EZj", /
"name":"SymbolConsoleTest", "definition": "Console add test", /
"unit": "Kg", "module":"IAMNumberOne", "category":"NewCategory", /
"ProjectTypeOfValue":"Constant", /
"ProgrammingLanguageSymbolType":"Constant_Subscripted"}'
```

Para añadir un usuario al sistema:

```
curl -X POST 'http://localhost:8080/add/user' /
-H "content-type: application/json" -d '{"token": "5UAZaIpuZfqZFDdr2EZj", /
 "username":"consoleTest", "fullname":"Shell User", "email":"example@email", /
 "password":"consolepass", "roles":[{"role":"Module Programmer", /
"module":"NewModule"},{"role":"Module Supervisor", "module":"ExampleModule"}]}'
```

Esto vale para aclarar el tipo de datos solicitado en la clave "roles".

Las rutas de modificación de recursos bajo PUT:

- /modify/symbol — previousName, newName, definition, unit, moduleName, categoryName, PTOVName, PLSTName
- /modify/symbol_modules — moduleName(módulo principal al que pertenece el símbolo), name(nombre del símbolo), module(nuevo módulo principal), secondary(módulos secundarios)
- /modify/symbol_indexes — moduleName(módulo principal al que pertenece el símbolo), name(nombre del símbolo), is_indexed, indexes
- /modify/index — previousName, newName, definition
- /modify/index_elements — name(nombre del índice), elements
- /modify/module — previousName, newName
- /modify/category — previousName, newName
- /modify/category/promote — name
- /modify/acronym — previousLetters, newLetters, meaning
- /modify/adjective — previousAdjective, newAdjective, usages

- /modify/semantic_rule — previousRule, newRule, explanation
- /modify/userProfile — previousUsername, newUsername, fullname, email, roles
- /modify/userPassword — username, previousPassword, newPassword

El tipo de datos de cada uno de los nombrados es el siguiente, los definidos bajo "Otros", representan las listas formadas de forma especial y por ello se definirán entre paréntesis a continuación de los mismos, el resto son cadenas de texto "'String" por lo que no se mencionarán:

- Otros(ArrayList): secondary(String), indexes(String), roles(igual que los roles de añadir usuario)

A continuación un par de ejemplos para mostrar el funcionamiento de las rutas de modificación:

En caso de querer modificar los índices asociados a un símbolo: (el campo "moduleName" no debería estar ahí ya que no es funcional y deriva en un error, ver 7.2.4)

```
curl -X PUT 'http://localhost:8080/modify/symbol_indexes' /
-H "content-type: application/json" -d '{"token": "5UAZaIpuZfqZFDdr2EZj", /
"moduleName": "IAMNumberOne", "name":"SymbolConsoleTest", /
"is_indexed": "True", "in dexes": ["IndexExample11", "IndexExample2", /
"IndexExample3"]}'
```

```
curl -X PUT 'http://localhost:8080/modify/acronym' /
-H "content-type: application/json" -d '{"token": "5UAZaIpuZfqZFDdr2EZj", /
"previousLetters":"Second_Acronym", "newLetters":"S.A.", /
"meaning":"Second_Acronym new meaning"}'
```

Las rutas de borrado de recursos bajo DELETE:

- /delete/symbol — name, module(Al que pertenece el símbolo)
- /delete/index — name
- /delete/module — name
- /delete/category — name
- /delete/acronym — letters
- /delete/adjective — adjective
- /delete/semantic_rule — ruleforShort

- /delete/user — username

Todos los tipos de datos presentados son cadenas de texto.

Se mostrará un ejemplo a continuación de borrado de índice:

```
curl -X DELETE ‘http://localhost:8080/delete/index’ /
-H "content-type: application/json" /
-d ‘{"token": "5UAZaIpuZfqZFDdr2EZj", "name": "NewExample"}’
```

Las rutas de validación de recursos bajo PUT:

- /validate/symbol — name, module(módulo del símbolo)
- /validate/index — name

Todos los tipos de datos presentados son cadenas de texto.

No se van a exponer ejemplos ya que se ha utilizado “PUT” anteriormente y los datos solo son cadenas de texto.

Las rutas de obtención de recursos bajo GET y POST, el uso de POST es debido a la formulación de peticiones en Ajax para el cliente web pero no se harán modificaciones en el diccionario de datos:

- /search/resources — text, exact
- /search/modules — text, exact
- /search/indexes — text, exact
- /search/symbols — text, exact
- /search/categories — text, exact

El dato “text” es una cadena de texto que puede estar vacía, y “exact” es un booleano(verdadero o falso).

Se va a mostrar un ejemplo de búsqueda con texto vacío para recibir todos los datos, búsqueda flexible y búsqueda exacta, todas ellas sobre las categorías para mostrar las salidas.

Texto vacío:

```
curl -X GET ‘http://localhost:8080/search/categories’ /
-H "content-type: application/json" /
-d ‘{"token": "5UAZaIpuZfqZFDdr2EZj", "text": "", "exact":false}’
```

Su respuesta sería similar a:

```
{"success": true, "response": [{"category\_name": "CategoryExampleTopLevel", /
"subcategories\_count": 2, "symbols\_count": 5, "subcategories": /
[{"category\_name": "CategoryExampleBottom", "symbols\_count": 4},/
{"category\_name": "NewSubCategory", /
"symbols\_count": 2}]}, {"category\_name": "NewCategory", /
"subcategories\_count": 0, /
"symbols\_count": 6, "subcategories": []}]}
```

Búsqueda flexible:

```
curl -X GET ‘http://localhost:8080/search/categories’ /
-H "content-type: application/json" /
-d ‘{"token": "5UAZaIpuZfqZFDdr2EZj", "text": "Example", "exact":false}’
```

Su respuesta sería similar a:

```
{"success": true, "response": [{"category\_name": "CategoryExampleTopLevel", /
"subcategories\_count": 2, "symbols\_count": 5, "subcategories": /
[{"category\_name": "CategoryExampleBottom", "symbols\_count": 4}, /
{"category\_name": "NewSubCategory", "symbols\_count": 2}]}]}
```

Búsqueda exacta:

```
curl -X GET ‘http://localhost:8080/search/categories’ /
-H "content-type: application/json" -d ‘{"token": "5UAZaIpuZfqZFDdr2EZj", /
"text": "CategoryExampleTopLevel", "exact":true}’
```

Su respuesta sería similar a:

```
{"success": true, "response": [{"category_name": "CategoryExampleTopLevel", /
"subcategories\_count": 2, "symbols\_count": 5, "subcategories": /
[{"category\_name": "CategoryExampleBottom", "symbols\_count": 4}, /
{"category\_name": "NewSubCategory", "symbols\_count": 2}]}]}
```

A parte de las mencionadas, las rutas básicas que hemos usado en las vistas: **/resources**, **/modules**, **/indexes**, **/symbols**, **/categories**, **/general** y **/settings**, son rutas reservadas para la renderización de las vistas y no se podran acceder mediante consola. En caso de querer obtener sus recursos, se pueden buscar en las rutas bajo **/search**. Si queremos obtener todos con dejar "text" como una cadena sin longitud valdrá. No hay un método de búsqueda para la página general ni los ajustes ya que estos redirigen a los símbolos.

Existen rutas adicionales como / o **/logout**, pero la primera es para acceder a la vista del login y la segunda para terminar la sesión web, en caso de querer usar las credenciales de otro usuario solo tendríamos que hacer **/login** de nuevo y obtendríamos una token para este.

Por otro lado, las rutas de **/qaGetSymbolsDefinition** y **/qaAddSymbolsDefinition** son algo especiales ya que la coordinación con el servicio de SonarQube requiere un formato diferente, cambiando la token por una contraseña estática del usuario de calidad.

La contraseña estática es en un principio "PGUcZgkN_y1DpWuodzzz".

Este sería un ejemplo de petición a **/qaGetSymbolsDefinition**:

```
curl -X GET 'http://localhost:8080/qaGetSymbolsDefinition' \
-H "content-type: application/json" \
-H "Authorization: Bearer PGUcZgkN_y1DpWuodzzz" \
-d '{"symbols": ["SymbolExample3", "example234"]}'
```

Respuesta:

```
{"symbols": [{"name": "SymbolExample3", "definition": /
"definition example for symbol 3 example", "unit": "L", "is_indexed": 1, /
"indexes": ["IndexExample3"], "modules": {"main": "IAMNumberOne", /
"secondary": []}, "category": "CategoryExampleBottom", /
"projectTypeOfValue": "Disagregated_Variable", /
"programmingLanguageSymbolType": "Variable"}, {"name": "example234", /
"definition": "a", "unit": "kg", "is_indexed": 1, "indexes": /
["IndexExample11", "IndexExample3", "IndexExample4"], /
"modules": {"main": "IAMNumberOne", "secondary": ["Testing_modules"]}, /
"category": "CategoryExampleBottom", "projectTypeOfValue": "Constant", /
"programmingLanguageSymbolType": "Function"}], "modules": /
["IAMNumberOne", "Testing_modules"], "indexes": [{"name": "IndexExample3", /
"definition": "definition example 3", "elements": ["ValueExample4"]}, /
{"name": "IndexExample11", "definition": "definition example 1", /
"elements": ["ElementExample", "One"]}, {"name": "IndexExample4", /
"definition": "definition example 4", "elements": ["ValueExample4", /
"ValueExample5"]}], "categories": [{"name": "CategoryExampleTopLevel", /
"level": 1, "super_category": "null"}]}
```

Este sería un ejemplo de petición a **/qaAddSymbolsDefinition**:

```
curl -X POST 'http://localhost:8080/qaAddSymbolsDefinition' \
-H "content-type: application/json" \
-H "Authorization: Bearer PGUcZgkN_y1DpWuodzzz" \
-d '{"symbols": [{"name":"newSymbolConsole1", \
"definition":"Example definition for console add", \
"unit": "N", "ProgrammingLanguageSymbolType": "Constant", \
"Category": "NewCategory"}, \
{"name":"newSymbolConsole2", \
"definition":"Another example definition from console", \
"unit": "Kg", "ProgrammingLanguageSymbolType": "Variable", \
"Category": "NewCategory"}], "module": "NewModule", \
"indexes": [{"indexName":"IndexConsole1", \
"definition":"Index definition console", \
"indexElements":["IndexElementExample1", "IndexElement2"]}, \
{"indexName":"IndexConsole2", \
"definition":"Index definition number two from console", \
"indexElements":["IndexElementExample2", "IndexElementExample3"]}]}'
```

Respuesta:

```
"Success"
```

Apéndice B

Resumen de enlaces adicionales

Dado que este año no se entregará CD, el código se puede obtener accediendo al repositorio GitLab del proyecto: `https://gitlab.inf.uva.es/davgome/tfg-microservicios` y descargándolo como archivo comprimido, o clonando el repositorio con git mediante:

```
git clone https://gitlab.inf.uva.es/davgome/tfg-microservicios.git.
```

Bibliografía

[1] F. Arconada-Orostegi. Comunicación http de microservicios. Master's thesis, Universidad de La Rioja, 2016. Ultima vez consultado a fecha de 11/12/2019. Páginas 31-54.

[2] A. Bisht. Chained services. https://daydreamer3d.github.io/tutorials/nameko-rabbitmq/chained_services.html. Ultima vez consultado a fecha de 16/07/2020.

[3] O. Blancarte. Data access object (dao) pattern. https://www.oscarblancarteblog.com/2018/12/10/data-access-object-dao-pattern/. Ultima vez consultado a fecha de 16/07/2020.

[4] C. Blanco. Gestión de proyectos software. https://ocw.unican.es/pluginfile.php/274/course/section/196/Lista%20de%20Riesgos%20en%20proyectos%20SW.pdf. Ultima vez consultado a fecha de 16/07/2020.

[5] Consortium for International Earth Science Information Network (CIESIN). Thematic guide to integrated assessment modeling. integrated assessment modeling 10 things to know. https://sedac.ciesin.columbia.edu/mva/iamcc.tg/mva-questions.html, 1997. Ultima vez consultado a fecha de 16/07/2020.

[6] Y. Crespo, I. de Blas, J. Vegas, L. J. M. González, I. Capellán-Pérez, D. Álvarez, M. Baumann, M. Mediavilla, R. Samsó, C. Llamas, C. Hernández, L. F. Lobejón, I. Arto, I. Cazcarro, and G. Parrado. Report of the common modeling framework. https://cordis.europa.eu/project/id/821105. version 1: May 2020, version 2: July 2020, web publication: forthcoming.

[7] Docker. Docker desktop. https://www.docker.com/products/docker-desktop. Ultima vez consultado a fecha de 16/07/2020.

[8] Docker. Docker volumes. https://docs.docker.com/storage/volumes/. Ultima vez consultado a fecha de 16/07/2020.

[9] Dpto. Ciencia de la Computación e Inteligencia Artificial. Universidad de Alicante. Orquestación de servicios: BPEL. http://www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion03-apuntes.html, 2014. Ultima vez consultado a fecha de 5/12/2019.

[10] GitLab. Getting started with gitlab ci/cd. https://gitlab.inf.uva.es/help/ci/quick_start/README. Ultima vez consultado a fecha de 12/11/2019.

[11] R. Gnatyk. Microservices vs monolith: which architecture is the best choice for your business? https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/, 2018. Ultima vez consultado a fecha de 16/07/2020.

[12] IEEE. IEEE standard for software project management plans. *IEEE Std 1058-1998*, pages 1–28, December 1998.

[13] IEEE. IEEE 1540-2001 - IEEE standard for software life cycle processes - risk management. https://standards.ieee.org/standard/1540-2001.html, 2001.

[14] D. Knuth. Computers and typesetting. https://www-cs-faculty.stanford.edu/~knuth/abcde.html.

[15] Locomotion-h2020. https://www.locomotion-h2020.eu. Ultima vez consultado a fecha de 16/07/2020.

[16] Lucid Software Inc. Creación de diagrama de puerta de enlace. https://www.lucidchart.com/pages/es. Ultima vez consultado a fecha de 16/07/2020.

[17] Microsoft. Precios de máquinas virtuales windows. https://azure.microsoft.com/es-es/pricing/details/virtual-machines/windows/. Ultima vez consultado a fecha de 16/07/2020.

[18] MrAprenderPython. Selenium testing con test suite y unittest. https://unipython.com/selenium-testing-test-suite-unittest/, April 2018. Ultima vez consultado a fecha de 12/11/2019.

[19] Nameko. Testing services. https://nameko.readthedocs.io/en/stable/testing.html. Ultima vez consultado a fecha de 16/07/2020.

[20] Overleaf. Bibliography management with bibtex. https://www.overleaf.com/learn/latex/Bibliography_management_with_bibtex. Ultima vez consultado a fecha de 12/11/2019.

[21] Pallets. Flask web development, one drop at a time. https://flask.palletsprojects.com/en/1.1.x/. Ultima vez consultado a fecha de 16/07/2020.

[22] R. S. Pressman. *Ingenieria del Software - Un Enfoque Practico 5b: Edicion*. McGraw-Hill Companies, 2002.

[23] QATestLab. Big-bang testing specifics. https://qatestlab.com/resources/knowledge-center/big-bang-testing/#:~:text=Big%20Bang%20Integration%20Testing%20is,until%20all%20components%20are%20completed. Ultima vez consultado a fecha de 16/07/2020.

[24] A. Rabadan. Gestión de riesgos de proyectos software. https://es.wikiversity.org/wiki/Gesti%C3%B3n_de_riesgos_de_proyectos_software, 2014. Ultima vez consultado a fecha de 16/07/2020.

[25] C. Richardson. Choosing a microservices deployment strategy. https://www.nginx.com/blog/deploying-microservices/. Ultima vez consultado a fecha de 16/07/2020.

[26] C. Richardson. Pattern: Api gateway / backends for frontends. `https://microservices.io/patterns/apigateway.html`. Ultima vez consultado a fecha de 16/07/2020.

[27] E. B. School. Cómo elaborar una matriz de riesgos. `https://www.ealde.es/como-elaborar-matriz-de-riesgos/`. Ultima vez consultado a fecha de 16/07/2020.

[28] Stakater. Microservices-testing. `https://github.com/stakater/microservices-testing`. Ultima vez consultado a fecha de 16/07/2020.

[29] P. Sánchez. Gestión de riesgos de proyectos software. `https://ocw.unican.es/pluginfile.php/1408/course/section/1803/tema7-gestionRiesgos.pdf`. Ultima vez consultado a fecha de 16/07/2020. Páginas 20, 28-30.

[30] L. Tagliaferri. Cómo instalar python 3 y configurar un entorno de programación en ubuntu 18.04. `https://www.digitalocean.com/community/tutorials/como-instalar-python-3-y-configurar-un-entorno-de-programacion-en-ubuntu-18-04-guia-de-inicio-rapido-es`. Ultima vez consultado a fecha de 16/07/2020.

[31] Thereviewstories. Model-view-controller (mvc). `https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66`. Ultima vez consultado a fecha de 16/07/2020.

[32] R. Velasco. Ahora puedes instalar más fácil que nunca python en windows 10. `https://www.redeszone.net/2019/01/04/instalar-python-37-windows-10/`. Ultima vez consultado a fecha de 16/07/2020.

[33] C. Vellage. Bibliography in latex with bibtex/biblatex. `https://www.latex-tutorial.com/tutorials/bibtex/`, October 2017. Ultima vez consultado a fecha de 12/11/2019.

[34] Vensim software. `http://vensim.com/vensim-software/`. Ultima vez consultado a fecha de 16/07/2020.