UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Generador automático de servicios REST

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ramírez Carrilero, Diego

Tutor/a: Carsí Cubel, José Ángel

CURSO ACADÉMICO: 2022/2023

Agradecimientos

En primer lugar, me gustaría agradecer a mi madre allí donde este, ya que, sin ella no me habría convertido en la persona que soy hoy en día, y, a mi tío, Javier Sevilla Peris por su tiempo para aconsejarme y asesorarme durante esta etapa.

También me gustaría agradecer a mi tutor José Angel Carsí Cubel, por su tiempo, esfuerzo y entendimiento durante el desarrollo del proyecto, un gran profesor que he tenido la suerte de tener en dos asignaturas de la carrera.

Por último, me gustaría agradecer a mi familia, amigos y pareja por el apoyo y cariño recibido.

¡Muchas gracias!

Diego Ramírez Carrilero.

Resumen

Este Trabajo Final de Grado describe el desarrollo de una herramienta para la generación automática de una API REST o servicio REST para acceder al contenido de una base de datos SQL y/o a un grafo de conocimiento. Debido a que, en la actualidad, una gran parte de los proyectos soportan sus datos mediante una base de datos relacional, y cada vez con más frecuencia mediante un grafo de conocimiento. La interoperabilidad de una aplicación es un requisito cada vez más demandado, por lo que esta herramienta puede ayudar a reducir el tiempo de desarrollo en cada vez más proyectos.

La interacción con la misma se realiza a través de una página web, y mediante un archivo de configuración, en formato JSON. Como resultado produce un archivo comprimido con el código generado, para su posterior instalación en un servidor.

La principal finalidad de esta herramienta no es otra que reducir el coste de tiempo invertido en el desarrollo del *back-end*, en concreto la realización del código necesario que forma parte de la comunicación entre el *back-end* y el sistema de almacenamiento de datos.

Palabras clave: api rest, servicio rest, generador automático, grafo de conocimiento.

Abstract

This Final Degree Project describes the development of a tool for the automatic generation of a REST API or REST service to access the contents of a SQL database or a knowledge network. Due to the fact that nowadays a large part of the projects support their data through a relational database, and more and more frequently through a knowledge network. The interoperability of an application is an increasingly demanded requirement, so this tool can help to reduce development time in more and more projects.

Interaction with it is done through a web page, and through a configuration file, in JSON format. As a result, it produces a compressed file with the generated code, for subsequent installation on a server.

The main purpose of this tool is none other than to reduce the cost of time invested in the development of the back-end, specifically the creation of the necessary code that forms part of the communication between the back-end and the data storage system.

Keywords: api rest, rest service, automatic generator, knowledge graph.

Tabla de contenidos

Contenido

1. Introducción 9
   - 1.1 Marco del proyecto 9
   - 1.2 Motivación 9
   - 1.3 Objetivos 10
   - 1.4 Impacto esperado 10
   - 1.5 Metodología 11
   - 1.6 Organización de la memoria 13
2. Estado del arte 15
   - 2.1 Estado del arte 15
   - 2.1 Bases de datos 15
   - 2.2 Grafos de conocimiento 16
   - 2.3 Métodos de comunicación 18
   - 2.4 Herramientas similares 22
3. Análisis del problema 26
   - 3.1 Requisitos 26
   - 3.2 Casos de uso 28
   - 3.3 Costes y riesgos 32
     - 3.3.1 Costes temporales 32
     - 3.3.2 Costes económicos 34
     - 3.3.3 Riesgos 36
4. Solución propuesta 39
   - 4.1 Solución propuesta 39
   - 4.2 Identificación y análisis de otras soluciones posibles 40
5. Diseño de la solución 43
   - 5.1 Arquitectura del sistema 43
   - 5.2 Diseño detallado 45
     - 5.2.1 Diagrama de clases UML 45
     - 5.2.2 Estructura de directorios 46
     - 5.2.3 Mockups 48
   - 5.3 Tecnologías utilizadas 51
6. Desarrollo e implementación 55
   - 6.1 Desarrollo de las API Rest base 55

6.2 Desarrollo del sistema 56
6.3 Desarrollo de la interfaz gráfica o *front-end* 57
6.4 Desarrollo de la comunicación front-end y back-end 58
7. Pruebas 60
7.1 Herramienta 60
7.2 Pruebas unitarias 61
7.3 Pruebas del sistema o pruebas de integración 62
7.4 Verificación 63
7.5 Validación 67
8. Conclusiones y trabajos futuros 69
8.1 Problemas a lo largo del proyecto 69
8.2 Conclusiones y revisión de objetivos 69
8.3 Trabajos futuros 70
Referencias 71
Bibliografía 72

Índice de figuras

Figura 1: Metodología en V 12
Figura 2: Las diez bases de datos más populares. Fuente: DB-Engines 2023 16
Figura 3: Aplicaciones de los grafos de conocimiento. Fuente: (Sevilla J, 2022) 18
Figura 4: Estructura de los mensajes SOAP 20
Figura 5: Porcentaje de cuestiones en Stack Overflow REST vs SOAP 21
Figura 6: Interfaz gráfica herramienta NodeApi Generator 23
Figura 7: Diagrama de casos de uso del sistema 28
Figura 8: Ilustración del sistema 39
Figura 9: Patrón cliente-servidor 43
Figura 10: Diagrama de clases UML del sistema 45
Figura 11: Estructura de directorios del sistema 46
Figura 12: Estructura de directorios de la API Rest 47
Figura 13: Pantalla inicial 48
Figura 14: Pantalla inicial - estado 1 - archivo adjunto 49
Figura 15: Pantalla de error 49
Figura 16: Pantalla de espera 50
Figura 17: Pantalla de descarga 50
Figura 18: Plantilla - Método de acceso a la base de datos 56
Figura 19: Plantilla - Método de acceso al grafo de conocimiento 56
Figura 20: Test de la clase CheckJSON.js 61
Figura 21: Salida del terminal del servidor 62
Figura 22: Salida del terminal del cliente 62

Índice de tablas

Tabla 1: Número de instancias, Hechos y tipos de Entidades y Relaciones de los grafos de conocimiento más populares. (Paulheim 2017) 17
Tabla 2: Deficiencias de la herramienta Postman mock server 24
Tabla 3: Deficiencias de la herramienta Generador de API REST NodeJS de MySQL + Postman Json + JWT Auth - Windows 24
Tabla 4: Requisitos funcionales 26
Tabla 5: Requisitos no funcionales 27
Tabla 6: Caso de uso: Descargar plantilla JSON y manual de instrucciones 29
Tabla 7: Caso de uso: Subir archivo de configuración 29
Tabla 8: Caso de uso: Sustituir archivo de configuración 30
Tabla 9: Caso de uso: Generar API Rest 30
Tabla 10: Caso de uso: Descargar API Rest 31
Tabla 11: Caso de uso: Rellenar archivo de configuración 31
Tabla 12: Costes del hardware 34
Tabla 13: Coste amortizado del hardware 34
Tabla 14: Coste del software 34
Tabla 15: Tabla de riesgos del proyecto 36
Tabla 16: Medidas de mitigación 37
Tabla 17: Medidas de contingencia 37
Tabla 18: Verificación de Requisitos Funcionales 63
Tabla 19: Verificación de Requisitos no Funcionales 66

1. Introducción

1.1 Marco del proyecto

En la actualidad, una gran parte de los proyectos soportan sus datos mediante una base de datos relacional, y cada vez con más frecuencia mediante un grafo de conocimiento. En el campo de las bases de datos, las del tipo relacional son las más utilizadas, ya que utilizan el lenguaje SQL uno de los más populares y demandados. Durante los últimos diez años, los grafos de conocimiento han emergido como una solución ampliamente adoptada en este campo, tanto en la industria como en el ámbito académico, y se han convertido en una tendencia de investigación en constante crecimiento.

Por otro lado, el hecho de la compartición de datos entre aplicaciones es cada vez más demanda por la industria y para ello, generalmente, se realiza mediante API's Rest, la tecnología más popular hoy en día. La parte mecánica de este trabajo puede ser automatizada a través de un sistema informático, lo cual permite que el desarrollador se enfoque en la generación de consultas para la base de datos o el grafo de conocimiento, en lugar de preocuparse por los procesos de acceso y generación del API Rest.

En el trabajo presentado en este documento se pretende configurar un sistema el cual, a partir de una serie de datos previamente solicitados, sea capaz de generar toda la infraestructura de un servicio capaz de poder comunicarse con una base de datos y/o grafo de conocimiento sin necesidad de codificar una sola línea de código.

1.2 Motivación

Esta herramienta surge como una propuesta de idea durante a una charla con un familiar cercano, en la cual se habló sobre qué temas me interesaban más, o qué aspectos de la ingeniería del software no había trabajado tanto. Después de varias sugerencias, propuso este tema, ya que en relación con su trabajo habría sido interesante disponer de una herramienta gratuita y sencilla que pudiera generar automáticamente un servicio Rest u API Rest. Mencionó que no había encontrado una herramienta que pudiera hacerlo, o en caso de haber una al menos él no sabía de su existencia.

Por otro lado, siempre me había dedicado más al trabajo relacionado con el desarrollo *front-end*, porque me resulta mucho más atractivo el diseño y codificación de interfaces tanto web como aplicaciones de escritorio. Debido a esto, la idea me pareció muy interesante para mejorar mis conocimientos relacionados con de desarrollo *back-end* y las bases de datos. Y, aunque más adelante me plantee rechazar la idea porque no sabía si era lo más adecuado, y quizás realizar un trabajo de fin de grado relacionado con aspectos de la informática que no son mi fuerte podría acabar en desastre, pensé que siempre habría algún tema o campo en el que no tuviera previa experiencia ni muchos conocimientos.

1.3 Objetivos

El objetivo principal de este trabajo es presentar una herramienta que minimice los tiempos de desarrollo de los proyectos software. Bajo esta premisa, se han especificado los objetivos asociados al principal, que se podrían resumir en los siguientes puntos:

1. Minimizar el número de errores cometidos. En el desarrollo de API o servicio Rest, una gran parte del código es reproducir partes codificadas previamente con ligeros cambios. Esta tarea repetitiva lleva en muchos casos a la generación de errores.

2. Disminuir la carga de trabajo. El desarrollo de un servicio Rest no es complicado, pero puede ser bastante repetitivo y largo en función de la magnitud de operaciones que se desea realizar.

3. Agilizar el desarrollo. El lapso que ocupa el desarrollo de esta parte del *back-end* se podría invertir en la implementación de nuevas funcionalidades o mejoras de las actuales. Además, se contaría desde un inicio la comunicación con el almacén de datos ya realizada y en funcionamiento.

4. Reducir los conocimientos necesarios del equipo de trabajo. El desarrollo de este tipo de servicios requiere unos conocimientos sobre *frameworks* web y especificación de API Rest que pueden no poseer todos los miembros del equipo.

5. Avanzar un paso en la automatización de servicios. La automatización de código es cada día más frecuente, sin embargo, esta práctica no se ha desarrollado tan en profundidad en este campo.

1.4 Impacto esperado

Esta herramienta supone una serie de ventajas para muchos programadores tanto *back-end*, *front-end* o *fullstack* en del desarrollo de un sistema. Las más destacables son la disminución de la carga de trabajo y la reducción del tiempo de desarrollo.

Para un desarrollador *back-end* este sistema supondría un ahorro de tiempo considerable en su trabajo, ya que toda la parte de comunicación con el centro de almacenamiento de datos ya sea una base de datos o un grafo de conocimiento, ya estaría codificado y en funcionamiento. Solo restaría definir el formato de envío de datos al *front-end*, en caso de requerir un formato distinto al JSON (JavaScript Object Notation o Notación de objeto JavaScript).

Para un programador especializado en interfaces de usuario o solamente con conocimientos en desarrollo *front-end,* supondría un ahorro de coste significativo de tiempo de desarrollo e implementación, ya que, no se requieren conocimientos precisos de *back-end* para poner en funcionamiento la API y todos los *end-point* se encontrarían definidos, de forma que desde el *front-end* solo restaría realizar una llamada al *end-point* deseado.

Para un programador *full-stack*, es decir, con conocimientos de *front-end* y *back-end* este sistema sería realmente interesante, ya que al igual que en los dos casos anteriores supondría un ahorro significativo en el coste de desarrollo del proyecto. Sin embargo, este al igual que el programador *back-end* podría realizar los cambios que necesitara de una manera más sencilla.

Por otro lado, también aporta un grado mayor de realidad, veracidad y consistencia si se utiliza como un apoyo para la realización de las pruebas de un sistema, ya que, se acercaría más al entorno real de uso de dicho sistema, en caso de que este utilizara un *back-end* propio para la comunicación con el almacén de datos escogido.

1.5 Metodología

Para la realización de este proyecto se ha seguido la metodología en V. Este modelo se encuentra dentro del grupo de las metodologías tradicionales y es una evolución o extensión del modelo en cascada, pero en lugar de avanzar en forma lineal, las etapas del proceso se doblan al pasar la fase de codificación, adquiriendo forma de V para llevar a cabo los procesos de validación y verificación. Como todas las metodologías tanto tradicionales como ágiles, el modelo en V presenta una serie de ventajas y desventajas.

Ventajas:

- Gestión sencilla del proyecto gracias a la planificación inicial.
- Gran énfasis en las pruebas.
- Buen funcionamiento para proyectos pequeños.
- Mayor probabilidad de éxito que el modelo de cascada.

Desventajas:

- Estructura poco flexible a cambios durante el desarrollo debido al desarrollo lineal.
- Modelo simple para proyectos complejos y de gran magnitud.
- El producto es obtenido al final del ciclo de vida.
- Las pruebas pueden ser costosas.

Este modelo se divide en dos partes fundamentales (En la Figura 1: Metodología en V se muestra dicha estructura), la parte izquierda de la V y la parte derecha. En la parte izquierda se encuentran las fases iniciales del proyecto: fase de análisis, especificación, diseño del sistema, diseño del programa y codificación. En la parte derecha se encuentran las fases de control, también conocidas como fases de validación o verificación, las cuales son: pruebas unitarias, pruebas del sistema, verificación del sistema y validación.

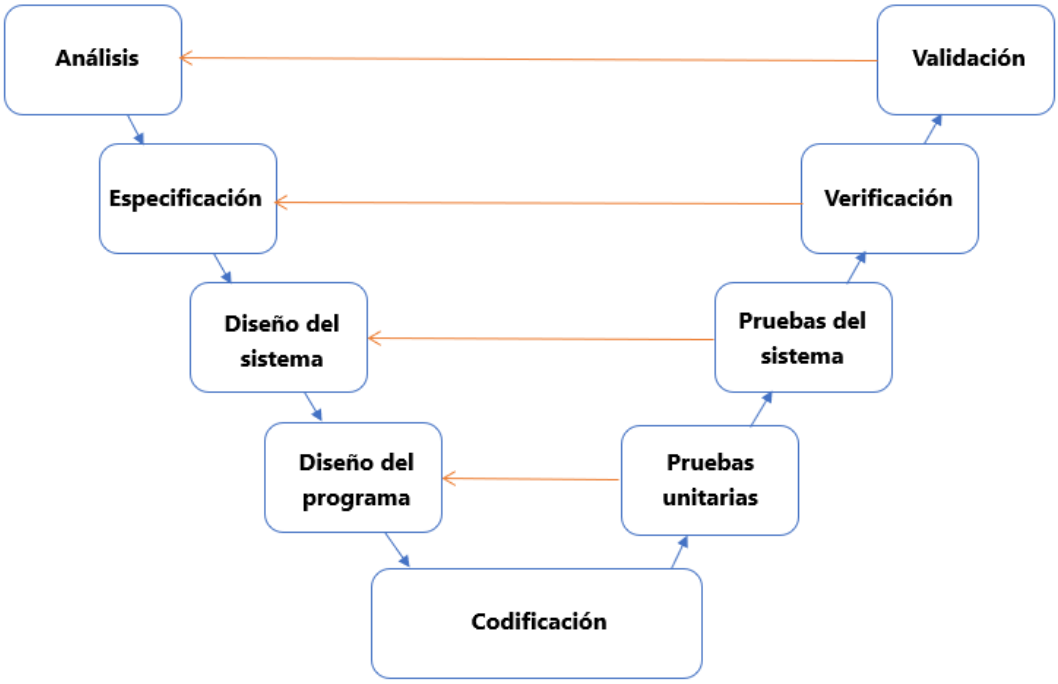

Figura 1: Metodología en V

1.6 Organización de la memoria

En este documento se recogen todos los aspectos relacionados con el desarrollo de este proyecto. A continuación, se describe como se ha estructurado la memoria y un pequeño resumen de que se puede encontrar en cada apartado:

Introducción: Punto actual donde se describe el marco del proyecto desarrollado en esta memoria, la motivación por la cual se lleva a cabo, los objetivos a cumplir por el sistema, el impacto esperado que se intuye tendrá, y por último la organización de la memoria que sirve como una guía para el lector de este documento.

Estado del arte: En esta sección de la memoria se describe el estado actual del desarrollo de software, una información general de las bases de datos y los grafos de conocimiento, los métodos de comunicación existentes más populares y las herramientas y/o sistemas actuales que podrían competir con la presentada en esta memoria.

Análisis del problema: En este punto se definen los requisitos que debe cumplir el sistema a desarrollar, tanto funcionales como no funcionales, se exponen los casos de uso de este, los diagramas de secuencia y un análisis de los costes y riesgos de proyecto.

Solución propuesta: Se presentan las soluciones que podrían llevarse a cabo para la realización y desarrollo del sistema. Se expone la solución elegida y las razones del descarte de las soluciones restantes.

Diseño de la solución: Este apartado tiene como finalidad presentar el diseño de la solución expuesta en el punto anterior. Para ello, se detalla la arquitectura del sistema escogida, y de realiza un diseño detallado del mismo mediante un diagrama de casos de uso, que ilustra, la relación entre las clases de la aplicación y su funcionamiento, una serie de *mockups* que representan los bocetos sobre los que se debe guiar el desarrollo de la interfaz gráfica y la estructura de directorios que debe presentar el sistema.

Desarrollo e implementación: En esta sección se presenta el desarrollo e implementación paso a paso del sistema definido. Dividido en 4 puntos: Desarrollo de las API Rest base, Desarrollo del sistema (el *back-end* de la herramienta, el punto más importante y costoso), Desarrollo de la interfaz gráfica o *front-end* y Desarrollo de la comunicación front-end y back-end.

Pruebas: En este punto se muestran las pruebas unitarias y de integración o del sistema realizadas al sistema. Además de la verificación y validación del mismo.

Conclusiones y trabajos futuros: En este punto final de la memoria se exponen los problemas que han surgido a lo largo del desarrollo del proyecto, las conclusiones respecto a este trabajo y los posibles trabajos futuros a realizar.

2. Estado del arte

En esta sección se presenta el estado de arte del ámbito del sistema a desarrollar en esta memoria, junto con las herramientas que se han considerado competidoras o similares con el sistema y una explicación de estas. Además, se presenta información relevante acerca de los grafos de conocimiento y las bases de datos.

2.1 Estado del arte

En la actualidad, el desarrollo software de una gran mayoría de las aplicaciones y páginas web está dividido en dos partes. Por un lado, encontramos el desarrollo centrado en el *front-end*, que es el encargado de la parte visual y de la interacción con el usuario, es decir, es la parte centrada en codificar toda funcionalidad de la interfaz mediante la cual se interactúa. Por otro lado, encontramos el desarrollo centrado en el *back-end*, que es el encargado de proporcionar los datos requeridos por el *front-end*, es decir, los datos que está solicitando el usuario a través de la interfaz, y de la comunicación con el almacén de datos donde se encuentran dichos datos.

La comunicación entre las aplicaciones de un sistema se realiza mediante un intermediario, una Application Program Interface (API). Se podría definir como una especificación formal que establece cómo un módulo perteneciente a un determinado sistema se comunica con otro. Una de las funciones principales de las API es el ahorro de recursos, como el tiempo y el dinero, a los desarrolladores de software, ya que, se pueden componer diversas partes de una aplicación mediante la utilización de API's de terceros. Un ejemplo muy común sería las pasarelas de pago que ofrecen muchas páginas web. Otra de sus funciones interesantes es utilizar intencionalmente las funciones de un servicio específico.

Este proyecto se centra en el desarrollo centrado en el *back-end* mediante una API, concretamente parte de la comunicación con las bases de datos y los grafos de conocimiento, con el fin de desarrollar un servicio con el que poder generar el código necesario para la creación de una API Rest totalmente funcional en el mínimo tiempo posible.

2.1 Bases de datos

Una base de datos es un conjunto de datos almacenados en memoria externa que están organizados mediante una estructura de datos (Marqués M, 2011). Esta se percibe como un gran almacén de datos que se define y crea solo una vez. Además, los datos que se almacenan se integran con una duplicidad mínima. Estos datos poseen una descripción de los mismos, que también se almacena en la base de datos, a los que se les denomina metadatos. Lo cual permite que exista una independencia lógico-física de estos.

La gestión y comunicación con una base de satos se realiza mediante un sistema de gestión de bases de datos (SGDB).

El sistema de gestión de la base de datos es una aplicación que permite a los usuarios definir, crear y mantener la base de datos, además de proporcionar un acceso controlado a la misma. Se denomina sistema de bases de datos al conjunto formado por la base de datos, el SGBD y los programas de aplicación que dan servicio a la empresa u organización. (Marqués M, 2011).

Hoy en día, existen multitud de bases de datos y sistemas de gestión de bases de datos disponibles. En la Figura 2: Las diez bases de datos más populares. Fuente: DB-Engines 2023 se muestran las diez bases de datos más utilizadas junto a su tipo.

Rank			DBMS	Database Model	Score		
Jun 2023	May 2023	Jun 2022			Jun 2023	May 2023	Jun 2022
1.	1.	1.	Oracle	Relational, Multi-model	1231.48	-1.16	-56.27
2.	2.	2.	MySQL	Relational, Multi-model	1163.94	-8.52	-25.27
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	930.06	+9.97	-3.76
4.	4.	4.	PostgreSQL	Relational, Multi-model	612.82	-5.08	-8.02
5.	5.	5.	MongoDB	Document, Multi-model	425.36	-11.25	-55.36
6.	6.	6.	Redis	Key-value, Multi-model	167.35	-0.78	-7.96
7.	7.	7.	IBM Db2	Relational, Multi-model	144.89	+1.87	-14.30
8.	8.	8.	Elasticsearch	Search engine, Multi-model	143.75	+2.11	-12.25
9.	↑ 10.	9.	Microsoft Access	Relational	134.45	+3.28	-7.36
10.	↓ 9.	10.	SQLite	Relational	131.21	-2.65	-4.22

Figura 2: Las diez bases de datos más populares. Fuente: DB-Engines 2023

La definición de lo que es una base de datos es clara y no surgen discrepancias de ella. Sin embargo, esto no sucede así con la definición de grafo de conocimiento.

2.2 Grafos de conocimiento

En la última década, los grafos de conocimiento se han convertido en una solución estándar en este ámbito, así como una tendencia de investigación en el mundo académico y en la industria (Kong, 2022).

A pesar de que la comunidad científica no se pone de acuerdo en una definición conjunta de grafo de conocimiento en la última década se han desarrollado varios estudios y trabajos relacionados con este concepto, así como diversas herramientas.

Su aplicación se extiende a varios campos, desde medicina, estructuración del lenguaje y patrimonio cultural.

Aunque la idea de representar el conocimiento mediante un grafo es antigua, y en 1956 se define como tal, el concepto de red semántica (Richens 1956). Es en la última década cuando el concepto de grafo de conocimiento se ha utilizado cada vez con más frecuencia, tanto en el ámbito de investigación como en el de negocio, generalmente relacionado con tecnologías de la web semántica (Ehrlinger, L., & Wöß, W. 2016). La introducción del

grafo de conocimiento de Google, en 2012 (Singhal 2012), fue un factor determinante para que el término fuera más popular (Sevilla J, 2022).

Algunos de los grafos de conocimiento más populares son (Chen, X., Jia, S., 2020):

- Grafo de conocimiento de Google (Singhal 2012), publicado en 2012 y es el más popular. Tiene más de 500 millones de instancias.
- Wikidata (Pellissier Tanon, Vrandečić, D. 2016), es una base de conocimiento estructurada, abierta y enlazada. Mantiene información de, aproximadamente, 280 lenguas de Wikipedia.
- Knowledge Vault (Dong, X., Gabrilovich, E., 2014), es un grafo de conocimiento creado por Google que se desarrolló con el objetivo de acceder al contenido de la web para aprender e incorporar nuevos datos. Este grafo de conocimiento usa análisis de texto, anotaciones humanas, junto con la estructura de las páginas, para acceder a la información de la web. Esta información es combinada con varias fuentes de datos, mediante técnicas de inteligencia artificial supervisada.

Tabla 1: Número de instancias, Hechos y tipos de Entidades y Relaciones de los grafos de conocimiento más populares. (Paulheim 2017)

Nombre	Instancias	Hechos	Entidades	Relaciones
Google K. Graph	500,000,000	3,500,000,000	1,500	35,000
Wikidata	15,759,256	43,189,154	23,157	1,203
Knowledge Vault	45,000,000	271,000,000	1,100	4,469

Aplicaciones del grafo de conocimiento

En la actualidad existen diversos trabajos publicados, que describen el estado del arte de los grafos de conocimiento, en los que se enumeran las diferentes aplicaciones que tienen.

Algunos de estos trabajos, exponen las aplicaciones de su uso que actúan principalmente sobre el mismo grafo de conocimiento para mejorarlo, y las que lo hacen hacia aplicaciones externas, a las que ofrecen servicios determinados. Esto queda reflejado en el trabajo de (Chen, X., Jia. S., 2020, Wang, Q., Mao, 2017).

Otros trabajos, como (Zou, 2020) se centran en el uso que se hace del grafo de conocimiento, en su funcionalidad, enumerando varias aplicaciones de los grafos en diferentes áreas, profundizando más en las diferentes aplicaciones desde el punto de vista del usuario final (Sevilla J, 2022).

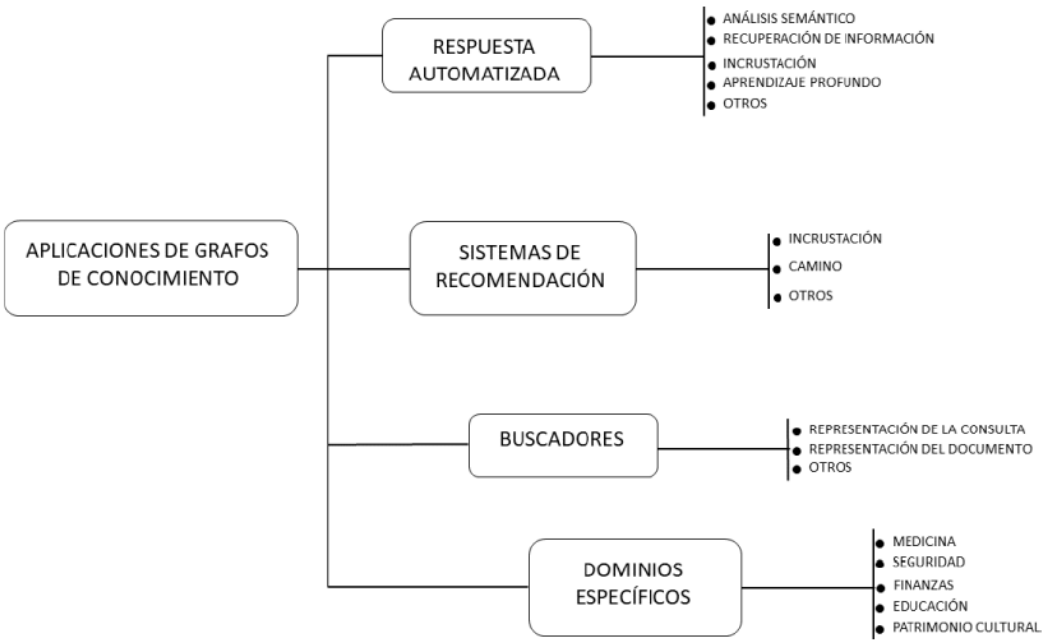

Figura 3: Aplicaciones de los grafos de conocimiento. Fuente: (Sevilla J, 2022)

2.3 Métodos de comunicación

Para lograr la comunicación entre las dos partes, el almacén de datos y la API Rest, estas se deben ser capaces de establecer una conexión y poder "entenderse mutuamente". Para ello, existen una serie de métodos mediante los cuales se puede desarrollar dicha comunicación.

A continuación, se describen los dos métodos de comunicación más usados en la actualidad y el método elegido para este proyecto.

REST (Representational State Transfer)

REST es un estilo arquitectónico software sencillo para sistemas distribuidos hypermedia como la WWW (World Wide Web o Red informática mundial). En la actualidad REST se utiliza para describir cualquier sistema que utilice el protocolo HTTP (Hypertext Transfer Protocol o Protocolo de transferencia de hipertextos) para la obtención de datos u operaciones sobre los mismos.

Utiliza el protocolo HTTP para comunicar datos entre servicios web en un entorno cliente-servidor, devolviendo esos datos en formatos muy específicos, como XML (Extensible Markup Language o Lenguaje de Marcado Extensible) y JSON.

Ventajas

- Permite separar el cliente del servidor. Esto quiere decir que nuestro cliente y servidor se pueden desarrollar mediante tecnologías y/o herramientas diferentes.
- Independencia del lenguaje de desarrollo.
- Tiene una gran comunidad. Muy útil en caso de buscar soluciones a problemas que pueden haberles surgido a otros y/o funciones ya desarrolladas.
- Ofrece la posibilidad de crear un diseño de un microservicio orientado a un dominio (DDD).
- Totalmente independiente de la plataforma, por lo que se puede hacer uso de REST tanto en Windows, Linux, Mac o el sistema operativo que escoja.
- Posibilidad de publicar la Aplicación de Interfaz del Programa (API) desarrollada.
- Escalable.

Desventajas

- No mantiene el estado, por lo que se debe montar una infraestructura propia para poder conservar el conjunto de la aplicación.
- Menor grado de seguridad comparado con SOAP. Admite el cifrado de seguridad de la capa de transporte (TLS).

SOAP (Simple Object Access Protocol)

SOAP es un protocolo estándar que se utiliza para el intercambio de datos mediante XML, fue creado por un conjunto de compañías entre las cuales se encuentran Microsoft e IBM (International Business Machines) y está basado en XML. Este protocolo es una derivación del protocolo XML-RPC que fue creado por Dave Winer en 1998.

Los mensajes SOAP son transportados principalmente a través de HTTP y sistemas de mensajería middleware, pero también pueden ser transmitidos mediante SMTP (Simple Mail Transfer Protocol) o JMS (Java Message Service o Servicio de mensajería Java).

Además, está formado por varias capas de especificación: MEP (Message Exchange Patterns) para el formato del mensaje, enlaces subyacentes del protocolo de transporte, el modelo de procesamiento de mensajes, y la capa de extensibilidad del protocolo. Y puede ser utilizado para la creación de protocolos más complejos y completos según las necesidades de los servicios que lo conforman.

Estructura de los mensajes

Un mensaje SOAP es un documento XML con una estructura definida en la especificación del protocolo. En la Figura 4: Estructura de los mensajes SOAP, se muestra un esquema con dicha estructura. Las siguientes partes conforman la estructura del mensaje:

- Sobre (Envelope): es el elemento raíz en cada mensaje SOAP y contiene dos elementos hijo, la Cabecera y el Cuerpo.
- Cabecera (Header): es un elemento opcional que se utiliza para pasar información relacionada con la aplicación que debe procesar los nodos SOAP a lo largo de la vía de acceso del mensaje. A su vez posee elementos hijo denominados bloques de cabecera.
- Cuerpo (Body): es un elemento obligatorio al contrario que la Cabecera, contiene la información relativa a la llamada y la respuesta.
- Error (Fault): es un subelemento del Cuerpo que contiene información relativa a errores que se hayan producido durante el procesado del mensaje y el envío de este.

Figura 4: Estructura de los mensajes SOAP

Ventajas

- Presenta una gran interoperabilidad. Gracias al uso de XML permite invocar procedimientos remotos de muchos lenguajes.
- Es escalable debido a que utiliza una comunicación vía HTTP.
- Puede ser implementado utilizando cualquier lenguaje y ejecutado en cualquier plataforma.
- Es posible utilizarlo mediante usuario anónimo y mediante autentificación.
- Tiene soporte para diversos protocolos de transporte capaces de transmitir texto, como SMTP o HTTP.
- Mayor grado de seguridad que REST. Debido al uso de protocolos integrados conocidos como la seguridad de los servicios web (WS Security).

Desventajas

- SOAP es más lento que otros middleware debido al uso de XML para la transferencia de mensajes.
- Depende de WSDL.
- Ciertos lenguajes no ofrecen un apoyo adecuado para su uso, ya sea a nivel de integración o de soporte IDE (Integrated Development Environment).
- Debido al alto nivel de seguridad, estas API requieren una mayor gestión.

¿Por qué REST?

Como se ha detallado anteriormente, SOAP y REST tienen una serie de ventajas y desventajas que los hacen más apropiados para un determinado uso. SOAP, por ejemplo, sería más adecuado en un entorno donde la seguridad de los datos fuera primordial. En este caso se ha escogido REST debido a que es menos complejo que SOAP en términos de codificación, utiliza por defecto el protocolo HTTP y ya se posee una experiencia previa con esta arquitectura. Además, como se muestra en la gráfica siguiente (Figura 5: Porcentaje de cuestiones en Stack Overflow REST vs SOAP), de la página web Stack Overflow, desde el año 2010 el porcentaje de cuestiones relacionadas con REST es considerablemente mayor al de SOAP. Lo que indica un mayor uso e interés por este estilo arquitectónico. Por lo que, hoy en día es se podría decir que es el "método de comunicación" más utilizado en la actualidad.

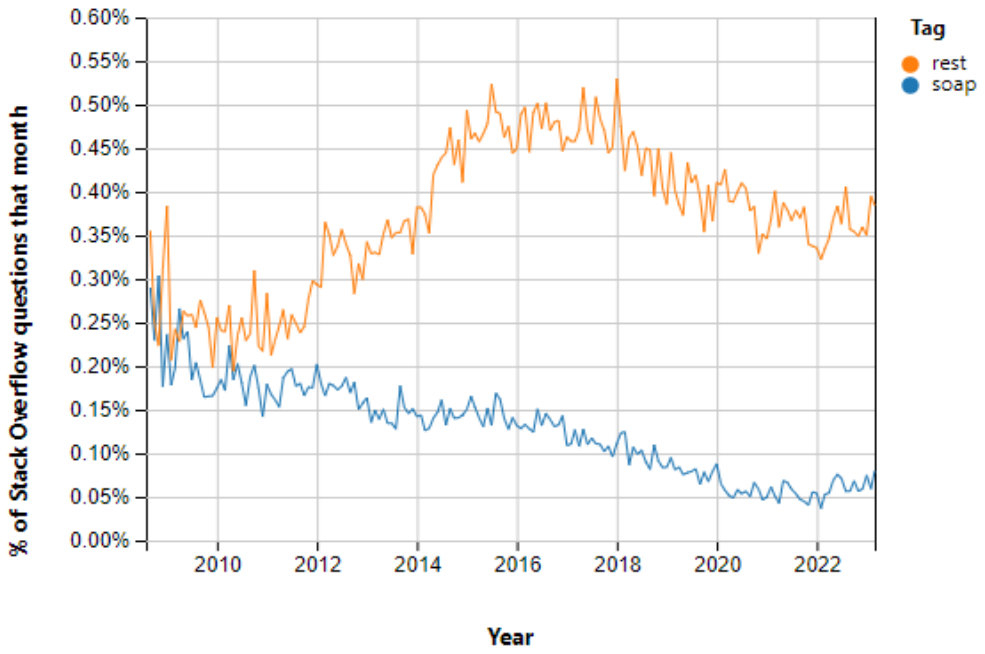

Figura 5: Porcentaje de cuestiones en Stack Overflow REST vs SOAP

2.4 Herramientas similares

Tras una larga investigación, no se han encontrado herramientas gratuitas para la generación automática de una API o servicio Rest. Pero se han encontrado dos posibles opciones que bien podrían competir con esta herramienta.

En primer lugar, encontramos una serie de páginas web que simulan el funcionamiento de una API Rest, a las que se las denomina comúnmente como “API simuladas”. Estas herramientas son muy interesantes, por ejemplo, para un entorno de pruebas, pero no hay ninguna posibilidad de poder descargar el código de estas y ejecutarlo en un servidor propio.

A continuación, se destaca una de ellas:

Postman mock server

Postman es una plataforma de API que permite a los desarrolladores diseñar, construir, utilizar, probar e iterar sus API. Postman comenzó siendo una extensión de los navegadores web como Chrome de Google o Firefox, que permitía, y aun lo permite, realizar peticiones de manera simple para probar API Rest tanto propias como de terceros. Pasado el tiempo Postman evolucionó y paso a convertirse en una aplicación con herramientas propias para una gran variedad de sistemas operativos.

Dentro de Postman se encuentra un servicio denominado *mock server*, un simulador de APIs con el cual se pueden diseñar servicios web, bajo la arquitectura Rest. Permite definir por completo la API, desde el método HTTP (GET, PUT, POST, DELETE y PATCH), parámetros de configuración, el cuerpo del mensaje y la respuesta.

Su uso es muy sencillo, en primer lugar, se crea un servidor denominado “mock server” y se configuran sus parámetros. Este servidor es donde se lanzan las peticiones, el cual puede ser creado a partir de una colección de datos existente o se puede crear de cero.

A continuación, se crea el tipo de petición deseada, entre los tipos disponibles mencionados anteriormente, y se configuran los parámetros (cabecera, cuerpo, etc.).

Por último, se manda la petición al servidor y se espera la respuesta.

Esta herramienta funciona como una API Rest pero sin tener que codificar la mayor parte del código. Además, no es posible enlazar con bases de datos externas.

Por otro lado, se han encontrado una serie de herramientas de pago para crear API Rest sobre entornos de desarrollo y bases de datos concretos. Al no ser gratuitas, no ha sido posible tener acceso a ellas por lo que no se ha probado su funcionamiento. Debido a esto, no se puede dar una explicación detallada ni tampoco ver a fondo que similitudes, desventajas o ventajas tienen respecto a la presentada en este documento. Aun así, cada herramienta posee un pequeño video adjunto en el que se puede observar la interacción con cada herramienta y el resultado que genera.

Dicho esto, se van a explicar una de las herramientas que tienen un mayor parecido a la desarrollada en este documento (Al no tener un nombre específico se exponen el nombre que aparece en la página web donde se encuentra publicada):

Generador de API REST NodeJS de MySQL + Postman Json + JWT Auth – Windows

Esta herramienta genera una API Rest a partir de la información de una base de datos. La interacción se realiza mediante una interfaz de escritorio, que se invoca desde un ejecutable, como se puede observar en la Figura 6: Interfaz gráfica herramienta NodeApi Generator.

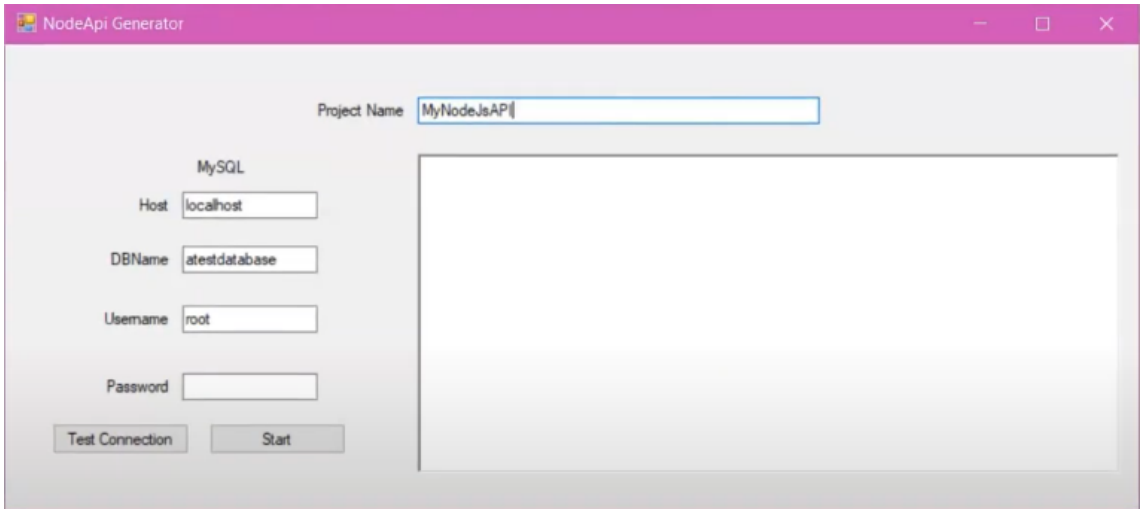

Figura 6: Interfaz gráfica herramienta NodeApi Generator

Al usuario se le solicitan los siguientes datos para poder generar la API: Project Name (Nombre del proyecto), Host, DBName (Nombre de la base de datos), Username (Nombre de usuario) y Password (Contraseña). Una vez introducidos, se pulsa el botón "Test Connection" y se comprueba si se ha podido establecer conexión con la base de datos con los valores introducidos. En caso afirmativo se mostrará el mensaje informando al usuario de que la conexión se ha establecido correctamente. En caso contrario, se intuye que se mostrará un mensaje de error.

Seguidamente, se debe pulsar en el botón "Start" para que comience la generación del código. Si la generación ha sido exitosa se mostrará un mensaje informando de la finalización de la generación. No se puede saber qué pasará en caso contrario.

Una vez terminada, se visualizará en la carpeta contenedora el proyecto generado. El cual, según se muestra en el video informativo que aparece en la web, se lanza en un servidor de Node.js.

Con la información proporcionada, se puede afirmar que esta herramienta permite generar una API Rest. En la grabación solo se muestran peticiones de tipo GET mediante el navegador, por lo que no se puede afirmar que esta herramienta realice las operaciones POST, PUT o DELETE. Además, por lo que parece, no es posible generar métodos propios.

Deficiencias de las herramientas

Tabla 2: Deficiencias de la herramienta Postman mock server

Postman mock server
No es posible utilizarla como API real, ya que, solamente simula su comportamiento.
No se puede utilizar con bases de datos externas.
Uso de la interfaz poco intuitivo en algunos casos.
No se puede utilizar con grafos de conocimiento.

Tabla 3: Deficiencias de la herramienta Generador de API REST NodeJS de MySQL + Postman Json + JWT Auth - Windows

Generador de API REST NodeJS de MySQL + Postman Json + JWT Auth – Windows
Aplicación de pago.
No se puede utilizar con grafos de conocimiento.
No es posible generar llamadas con consultas SQL personalizadas.

3. Análisis del problema

En este apartado se realiza un análisis de requisitos y se presentan los requisitos resultantes obtenidos a través de dicho análisis, se exponen los casos de uso del sistema y un análisis de los costes y riesgos del desarrollo de este.

3.1 Requisitos

En este apartado se describe en detalle los requisitos que deberá cumplir el generador automático de API o servicio Rest. Para ello, se han planteado una serie de requisitos funcionales y no funcionales con tal de cumplir los objetivos del proyecto.

Dentro de estos, no solo se han contemplado las características que afectan al desarrollo actual sino también las futuras mejoras que se podrían aplicar a este proyecto.

Primeramente, se presentan los requisitos funcionales, los cuales definen lo que debe hacer el sistema. Enumerados en la Tabla 4: Requisitos funcionales.

Tabla 4: Requisitos funcionales

Requisitos funcionales	
RF. 1	El usuario debe ser capaz de subir un archivo en formato JSON con un formato definido.
RF. 2	El usuario podrá intercambiar el archivo en formato JSON subido previamente por otro con el mismo formato.
RF. 3	El usuario debe ser capaz de descargar con éxito el archivo generado.
RF. 4	Se podrán configurar siguientes los parámetros de la herramienta mediante el archivo de configuración en formato JSON: • Puerto de escucha • Host • Nombre de usuario • Contraseña • Nombre de la base de datos • URL (Uniform Resource Locator o Localizador de Recursos Uniforme) de acceso al grafo de conocimiento • Métodos de la base de datos

	• Métodos del grafo de conocimiento
RF. 5	El usuario no debe ser capaz de pulsar el botón “Crear API Rest” sin haber subido el archivo de configuración.
RF. 6	El usuario deber ser capaz de pulsar el botón “Crear API Rest” una vez ha subido el archivo de configuración.
RF. 7	El usuario podrá descargar el código generado al pulsar en el botón “Descargar código”.
RF. 8	Se proporcionará un mensaje indicativo del resultado de la generación del código.
RF. 9	El botón que indica el inicio de la generación del servicio debe estar deshabilitado hasta que el usuario proporcione el archivo de configuración.
RF. 10	El botón que permite la descarga del servicio resultante debe estar oculto hasta que termine la generación del código.
RF. 11	Se debe cumplimentar al menos una de las dos opciones, base de datos o grafo de conocimiento.
RF. 12	El usuario podrá descargar la plantilla JSON y el manual de instrucciones al pulsar en el botón “Descargar la plantilla JSON y el manual de instrucciones”.

En segundo lugar, encontramos los requisitos no funcionales que son las restricciones impuestas a un sistema que definen sus atributos de calidad. Enumerados en Tabla 5: Requisitos no funcionales.

Tabla 5: Requisitos no funcionales

Requisitos no funcionales	
RNF. 1	El número máximo de archivos JSON que se pueden subir debe ser 1.
RNF. 2	El número máximo de proyectos generados a la vez debe ser 1.
RNF. 3	El servicio generado debe de ejecutarse en un servidor mediante el *framework* Node.js.
RNF. 4	El número máximo de métodos introducidos para la generación del código puede ser infinito.
RNF. 5	El número mínimo de métodos introducidos para la generación del código debe ser 1.
RNF. 6	El mensaje indicativo, mostrado durante o en la finalización de la generación del servicio, debe exponer si ha tenido éxito o fracaso. En caso de fracasar indicar el motivo.

3.2 Casos de uso

En este apartado se presentan los casos de uso obtenidos a partir de los requisitos funcionales mencionados anteriormente. Estos tienen la función de representar que hace el sistema, no como lo debe hacer.

En primer lugar, se identifican los actores, es decir, las entidades que van a operar sobre el sistema. La herramienta desarrollada cuenta con una interfaz a través de la cual se interactúa para proporcionar el archivo de configuración y descargar el código resultante, por lo que el sistema contará con un actor, el *usuario*.

A continuación, se muestra un diagrama de los casos de uso del sistema para ilustrar las relaciones entre los mismos:

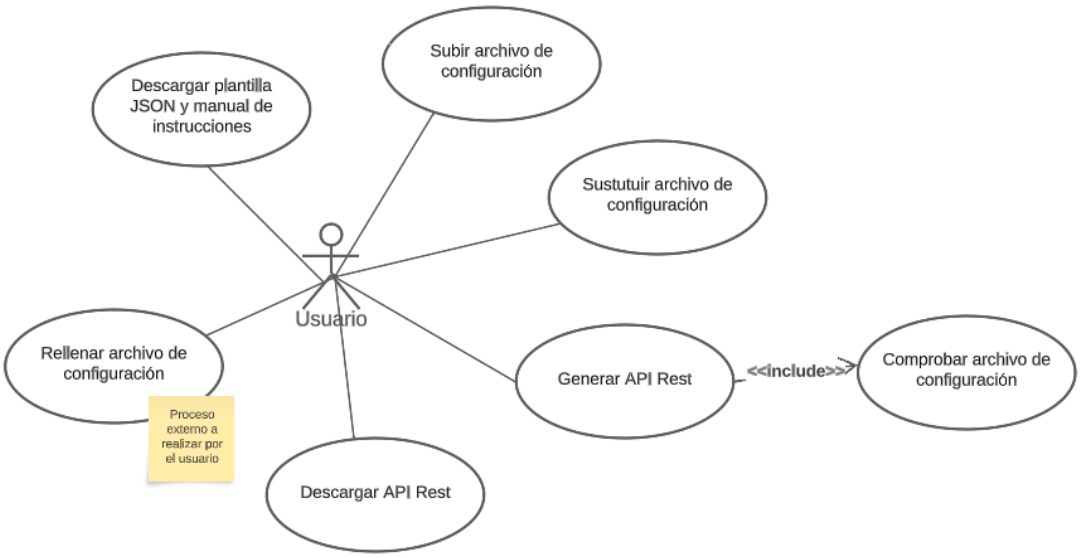

Figura 7: Diagrama de casos de uso del sistema

En las tablas siguientes se realiza la descripción de cada uno de los casos de uso reflejados en la Figura 7: Diagrama de casos de uso del sistema:

Tabla 6: Caso de uso: Descargar plantilla JSON y manual de instrucciones

Caso de Uso	Descargar plantilla JSON y manual de instrucciones
Actores	Usuario
Propósito	Obtener la plantilla del archivo JSON y el manual de instrucciones
Resumen	Se permite la descarga de una plantilla vacía con el formato requerido y un manual para cumplimentarla de manera adecuada
Tipo	Primario
Requisitos asociados	RF. 12
Flujo	1. Pulsar en el botón con el texto: "Descargar plantilla JSON y manual de instrucciones"

Tabla 7: Caso de uso: Subir archivo de configuración

Caso de Uso	Subir archivo de configuración
Actores	Usuario
Propósito	Proporcionar los datos para la generación de la API Rest
Resumen	Se comprueba el formato y la extensión del archivo de configuración proporcionado por el usuario
Tipo	Primario
Requisitos asociados	RF. 1
Flujo	1. Descargar el archivo de configuración predefinido 2. Rellenar el archivo de configuración 3. Pulsar en el recuadro o botón central de la pantalla 4. Escoger el archivo de configuración
Precondiciones	La ayuda para completar adecuadamente el archivo de configuración debe de haber sido leída previamente

Tabla 8: Caso de uso: Sustituir archivo de configuración

Caso de Uso	Sustituir archivo de configuración
Actores	Usuario
Propósito	Sustituir el archivo de configuración por otro
Resumen	Sustitución del archivo de configuración proporcionado por el usuario
Tipo	Secundario
Requisitos asociados	RF. 2
Flujo	1. Pulsar en el recuadro o botón central de la pantalla 2. Escoger el archivo de configuración sustituto

Tabla 9: Caso de uso: Generar API Rest

Caso de Uso	Generar API Rest
Actores	Usuario
Propósito	Iniciar el proceso de la generación del código
Resumen	Se inicia la generación del código del servicio Rest a partir de los datos proporcionados en el archivo de configuración
Tipo	Primario
Requisitos asociados	RF. 5, RF. 6, RF. 8 y RF. 9
Flujo	1. Pulsar en el botón “Crear API Rest” que inicia la generación del código
Precondiciones	El usuario debe de haber proporcionado el archivo de configuración

Tabla 10: Caso de uso: Descargar API Rest

Caso de Uso	Descargar API Rest
Actores	Usuario
Propósito	Obtener el código generado por el sistema
Resumen	Se inicia la descarga del código generado por la herramienta para su posterior implementación
Tipo	Primario
Requisitos asociados	RF. 3, RF. 7 y RF. 10
Flujo	1. Pulsar en el botón "Descargar API Rest" que inicia la descarga del código resultante
Precondiciones	La generación del código debe haber finalizado.

Tabla 11: Caso de uso: Rellenar archivo de configuración

Caso de Uso	Rellenar archivo de configuración
Actores	Usuario
Propósito	Cumplimentar archivo de configuración proporcionado
Resumen	Cumplimentación del archivo de configuración proporcionado que sirve como base para la generación del servicio Rest resultante
Tipo	Primario
Requisitos asociados	RF. 4 y RF. 11
Flujo	1. Descargar el archivo de configuración proporcionado junto con la ayuda 2. Revisar el formato del archivo de configuración (debe ser JSON) 3. Rellenar los campos indicados en el archivo según se indica en del documento de ayuda. 4. Guardar el archivo de configuración
Condiciones	El archivo debe de cumplimentarse siguiendo la guía indicada en la ayuda.

3.3 Costes y riesgos

Para el correcto desarrollo de todos los proyectos es necesario realizar un estudio de su viabilidad, un análisis previo de los costes económicos y temporales que estos suponen y un análisis de los riesgos.

Para ello, en este capítulo, se presentan las distintas etapas por que transcurrirá el trabajo junto con la estimación del tiempo que se debe invertir en cada una de ellas, y los costes económicos hipotéticos que se deben tomar en cuenta si este proyecto se realizara por un grupo de trabajo.

3.3.1 Costes temporales

Este proyecto de fin de grado se divide, a grandes rasgos, en seis etapas fundamentales:

Inicio: En esta etapa se realiza la propuesta del trabajo y tienen lugar las primeras reuniones con el tutor. Se llevan a cabo las primeras investigaciones para comprender el ámbito del problema y definir los objetivos de la herramienta a desarrollar. Coste asociado: 8 horas.

Análisis: Este punto abarca las tareas destinadas al análisis del sistema. Comienzan las tareas de definición de requisitos, estudio de los costes y riesgos y la definición de diagramas y modelos. Coste asociado: 33 horas.

Diseño: Comienza el diseño del sistema a desarrollar, divido en el diseño de la interfaz y el diseño del programa. Además de la búsqueda de herramientas para la implementación del trabajo. Coste asociado: 26 horas.

Desarrollo o implementación: Esta fase esta destina a seleccionar las herramientas que serán utilizadas para el proyecto. Además, abarca toda la etapa de la codificación de este. Coste asociado: 57 horas.

Pruebas: Abarca las pruebas del sistema, unitarias, la validación y la verificación del programa implementado. Coste asociado: 32 horas.

Redacción de la memoria: Engloba la redacción del trabajo realizado de inicio a fin. En el documento resultante queda reflejado el trabajo realizado en las fases anteriores y se da una conclusión sobre el trabajo realizado. Coste asociado: 152 horas (Esta etapa es transversal, por lo que en el diagrama presentado más adelante algunas de las horas asociadas en esta etapa están contabilizadas en las etapas anteriores).

Estas llevan asociadas un coste temporal global, y cada una de ellas se encuentra dividida en subetapas o subtareas con su coste temporal asociado (El coste global de cada etapa es la suma del coste temporal de las subtareas asociadas a la misma).

Para ilustrar el coste temporal del proyecto, que en total suma 294 horas (En total suman 308 horas, pero algunas tareas comienzan al mismo tiempo que otras por lo que ese periodo solo se contabiliza una vez), se ha realizado un Diagrama de Gantt, especificado en la Ilustración 1: Diagrama de Gantt.

Ilustración 1: Diagrama de Gantt

3.3.2 Costes económicos

Los costes económicos de este proyecto van asociados al hardware y software utilizado para su desarrollo e implementación y al coste del personal.

Costes del hardware

Para el desarrollo de esta herramienta se han utilizado tres componentes hardware: un ordenador de escritorio de gama media/alta, donde se ha escrito y desarrollado el proyecto, un monitor y una unidad de estado sólido o SSD (Solid State Drive) donde se ha almacenado.

Tabla 12: Costes del hardware

	Ordenador de escritorio	Monitor Asus	Unidad de estado sólido
Coste	900 €	110 €	55 €

Para el cálculo del coste de los componentes en relación con este proyecto se ha calculado el coste amortizado siguiendo la formula siguiente:

Coste Amortizado = Número de días del proyecto * Precio del componente / Vida útil

Según los coeficientes de amortización lineal, aportados por la Agencia Tributaria, se obtiene que el número mínimo de años en el que se puede amortizar es cuatro y máximo de ocho. Por lo que, considerando que la vida útil es seis, la media entre el mínimo y el máximo, se obtienen los siguientes costes:

Tabla 13: Coste amortizado del hardware

	Ordenador de escritorio	Monitor Asus	Unidad de estado sólido
Coste amortizado	31,3 €	3,8 €	1,9 €

Costes del software

El software utilizado para la realización del proyecto ha sido:

Tabla 14: Coste del software

Software	Licencia	Coste
Visual Studio Code	Gratuita	0 €
Node.js	Gratuita	0 €
XAMPP	Gratuita	0 €
GanttPRO	Gratuita	0 €

Canva	Gratuita	0 €
Microsoft Office	Gratuita (Aportada por la UPV)	0 €
Jest	Gratuita	0 €

Costes del personal

Este proyecto se ha llevado a cabo solamente por el autor de este documento. Por ello, se ha asumido solamente el rol de programador. El coste de este perfil, extraído del portal de empleo InfoJobs, durante el tiempo de desarrollo de este proyecto sería el siguiente:

Programador: Su salario aproximado es de 25500 euros brutos al año, 2125 euros al mes, ajustado a los 4 meses de duración del proyecto, teniendo en cuenta 4 horas laborables por día, su coste sería de 4250 euros.

Se ha consultado la tabla de cotizaciones proporcionada por la Seguridad Social para calcular el coste derivado de las contrataciones. Dado que las horas de trabajo se clasifican como "comunes", se debe aplicar un porcentaje de contingencia del 23.6%. Por lo tanto, el coste total del personal ascendería a:

Coste del Personal = Programador + Programador * 0,236 = 4250 + 4250 x 0,236 = 5253 euros

Coste total

Una vez finalizado el análisis económico de cada una de las partes se obtiene el coste total del proyecto. Cabe destacar que en caso de que este proyecto se realizase en un local, se deben contabilizar los costes asociados al mismo, como podría ser, luz, agua, alquiler, etc. Una estimación del coste indirecto supondría un añadido del 12,5% de los costes directos ya calculados. Por lo que el total ascendería a:

Coste Total = Coste del Hardware + Coste del Software + Coste del Personal + (Coste del Hardware + Coste del Software + Coste del Personal) * 12,5% = 37 + 0 + 5253 + (37 + 0 + 5253) * 12,5% = 5951 euros.

3.3.3 Riesgos

En este apartado se realiza un análisis de los riesgos potenciales que pueden surgir en el transcurso del proyecto. Este análisis tiene como objetivo prevenir los errores que puedan ir surgiendo y las acciones que se debe llevar a cabo para que el impacto de estos se anule o se reduzca. Los posibles riesgos que podrían darse en este trabajo son los siguientes:

Tabla 15: Tabla de riesgos del proyecto

Id	Evento	Probabilidad	Impacto	Clasificación
1	No se definen claramente los objetivos del proyecto	40%	8	Alto
2	Surgen nuevos requisitos a lo largo del desarrollo del proyecto	70%	8	Alto
3	El desarrollo del sistema cuesta más tiempo de lo planificado	50%	8	Alto
4	Surgen fallos en las fases de prueba obligan a realizar cambios en el diseño del sistema	50%	8	Alto
5	La redacción del proyecto cuesta más tiempo de lo planificado	50%	4	Medio
6	El alcance del proyecto es incorrecto	35%	8	Medio
7	El aprendizaje de las herramientas cuesta más tiempo de lo esperado	15%	4	Bajo
8	Las herramientas seleccionadas no se adecuan con precisión a las necesidades del proyecto	15%	16	Medio
9	No se realiza un control y seguimiento adecuado del proyecto	30%	4	Bajo
10	El usuario no es capaz de rellenar los campos correctamente	20%	4	Bajo
11	Otras tareas ocupan tiempo de desarrollo del proyecto	25%	4	Bajo
12	El desarrollador se pone enfermo	10%	8	Bajo

Una vez obtenidos los riesgos potenciales, se proponen una serie de medidas de mitigación. Estas tienen como objetivo reducir la probabilidad de que se produzca el riesgo y deben tomarse desde el inicio y a lo largo del desarrollo del trabajo. Las medidas de mitigación que propuestas para los riesgos clasificados como nivel Alto y Medio se especifican en la siguiente tabla:

Tabla 16: Medidas de mitigación

Riesgo	Medidas de mitigación
1 y 2	Se debe realizar previamente un estudio sobre el tema que abarca el trabajo para definir de la manera más correcta y precisa los objetivos y requisitos necesarios.
3, 5 y 6	Se debe realizar un análisis lo más adecuado posible del coste temporal del proyecto.
4 y 8	Se debe dedicar un periodo de tiempo más prolongado para el estudio de las herramientas seleccionadas y su implementación.

En caso de que alguno de los riesgos acabe sucediendo, es necesario aplicar una serie de medidas de contingencia. Estas tienen como finalidad reducir el impacto del riesgo que pueda darse, de manera que se puedan cumplir los costes temporales y económicos que se habían considerado en la mayor medida posible. Algunas de las medidas de contingencia que se podrían llegar a implementarse en caso de que surjan algunos de los riesgos listados son las siguientes:

Tabla 17: Medidas de contingencia

Riesgo	Medidas de contingencia
1 y 2	Se deben adaptar los nuevos objetivos y requisitos al tiempo disponible, así como reducir el alcance del proyecto.
3 y 5	Se debe hacer uso del tiempo adicional previsto en la planificación del proyecto.
8	Se deben buscar nuevas herramientas que se adapten al proyecto, en el periodo de tiempo adicional contemplado, o usar la combinatoria de otras herramientas para compensar los flecos de las iniciales.

4. Solución propuesta

En esta sección se expone la solución propuesta para el desarrollo de una herramienta la cual se capaz de generar automáticamente una API Rest o servicio Rest con el fin de minimizar el tiempo de desarrollo de un sistema software. Por otro lado, se identifican otras posibles soluciones y una justificación del rechazo de desarrollo de estas.

4.1 Solución propuesta

La solución propuesta en esta memoria para minimizar el tiempo de desarrollo de un sistema software y automatizar la generación de una API Rest, es la creación de una herramienta capaz de generar todo el código necesario de una API capaz de comunicarse con una base de datos y/o un grafo de conocimiento. Esta herramienta posee una interfaz web mediante la cual el usuario interactúa con el sistema proporcionado la información solicitada por el este, que en este caso es un archivo de configuración a cumplimentar con los datos pertinentes para la correcta generación del código resultante. El funcionamiento del sistema deber ser similar al expuesto a continuación.

En primer lugar, el usuario debe cumplimentar de manera correcta la información solicitada y proporcionársela al sistema. Seguidamente, este comenzará el proceso de creación de la API. Por último, en cuanto el proceso de generación haya finalizado, en caso de éxito, se mostrará un mensaje indicativo por pantalla y un botón mediante el cual se procede al inicio de la descarga del código generado. En caso de error, es decir, los datos proporcionados por el usuario no son correctos o haya sucedido un fallo durante el proceso de creación que no permita la generación del código, se visualizará tal fallo y/o error a través de la interfaz. En la Figura 8: Ilustración del sistema se muestra a grandes rasgos el funcionamiento del sistema.

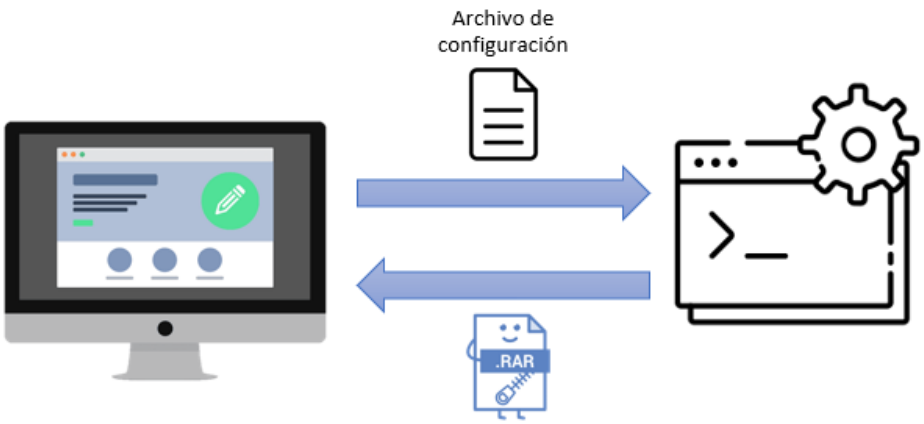

Figura 8: Ilustración del sistema

El desarrollo de esta solución se plantea inicialmente en tres fases generales. En primer lugar, el desarrollo iniciará con la creación dos API Rest totalmente funcionales las cuales sean capaces de conectarse una base de datos y a un grafo de conocimiento. Y sea posible realizar llamadas GET, PUT, POST y DELETE a dichos sistemas y comprobar su correcto funcionamiento.

En segundo lugar, se inicia el desarrollo de la funcionalidad interna del sistema, es decir, la parte asociada al *back-end* de la herramienta. En esta parte se encuentran las plantillas de código que sirven como base para la creación de la API Rest resultante, además de todas las clases y funciones que hacen posible la generación de este código a partir de las API Rest desarrolladas en la fase inicial. Esta es la fase del desarrollo junto a la inicial que más tiempo de codificación es necesario emplear por ello son las fases iniciales, además de ser el centro fundamental del sistema.

Una vez completadas las fases anteriores da comienzo el desarrollo del *front-end*, es decir, la codificación de la interfaz de usuario y su funcionalidad para poder comunicarse con el *back-end* y trasmitirle el archivo aportado por el usuario para iniciar la generación del código.

4.2 Identificación y análisis de otras soluciones posibles

Se han identificado 2 soluciones diferentes a la propuesta anteriormente. A continuación, se exponen las mismas y la razón de su descarte.

Solución 1: Interfaz web con formulario y API Rest separadas

Esta solución difiere en dos puntos clave de la propuesta anterior. En primer lugar, en el modo en el que el usuario introduce los datos necesarios en el sistema, es decir, difiere en la manera en que se codifica el *front-end* del sistema. La cumplimentación de los datos se realizaría mediante un formulario dentro de la misma página web, el cual comprobaría cada campo a cumplimentar de manera automática una vez introducida la información en él.

En segundo lugar, se crearía una API Rest diferente para cada opción, es decir, una API Rest para las bases de datos y otra diferente para los grafos de conocimiento.

Las ventajas de esta opción en lo referido a la interfaz de usuario son notables, ya que, el usuario no deberá descargar el archivo de configuración y cumplimentarlo localmente para más tarde proporcionarlo de vuelta, en lugar de eso los datos serían introducidos directamente en la interfaz web. Además, los errores que hayan podido surgir durante la cumplimentación solo serán notificados una vez el usuario haya proporcionado de vuelta el archivo. Sin embargo, en esta solución, si se cometen durante la introducción de los datos las correcciones que deban aplicarse se notificarían al instante.

Descarte solución 1

Esta solución ha sido descartada debido a dos razones:

- La codificación del *front-end* supondría un mayor tiempo de pruebas y trabajo.
- La generación de dos API Rest distintas ya es posible en la solución propuesta.

Solución 2: Aplicación de escritorio

En esta solución la herramienta se desarrollaría para ser una aplicación de escritorio con la cual poder generar API Rest de manera local. El funcionamiento sería el mismo que el de la solución propuesta puesto que la lógica del sistema no difiere en nada.

Esta propuesta es muy interesante, ya que permite la generación automática de API Rest de forma local y sin necesidad de conexión a internet. Aunque solamente en caso de poseer una base de datos y/o grafo de conocimiento locales.

Descarte solución 2

Esta solución ha sido descartada debido a dos razones:

- La codificación de la interfaz de usuario supondría un mayor trabajo, además de un mayor énfasis en las pruebas al igual que la solución 1.
- El aprendizaje de una nueva herramienta para codificación de aplicaciones de escritorio, debido a que solo se posee experiencia con aplicaciones destinadas a dispositivos móviles.

5. Diseño de la solución

En este punto de la memoria se presenta la arquitectura software del sistema desarrollado, la razón de la elección de dicha arquitectura y sus principales ventajas y desventajas. Se detalla el diseño de la herramienta mediante un diagrama de clases y *mockups*. Y, por último, se describen las tecnologías empleadas en el desarrollo de este proyecto.

5.1 Arquitectura del sistema

El patrón arquitectónico escogido para el desarrollo de este proyecto ha sido el patrón cliente-servidor. Este es muy utilizado para el diseño y desarrollo de servicios en la web.

El modelo cliente-servidor se compone de tres partes fundamentales, un cliente, que representan a las entidades que necesitan servicios, un servidor, que es la entidad que proporciona tales servicios, y un middleware, que se encarga de la comunicación entre cliente y servidor. El funcionamiento sería el siguiente, cliente realiza una petición de un servicio al servidor, este recibe la petición, la procesa y devuelve la respuesta al cliente. En la Figura 9: Patrón cliente-servidor queda representado este funcionamiento.

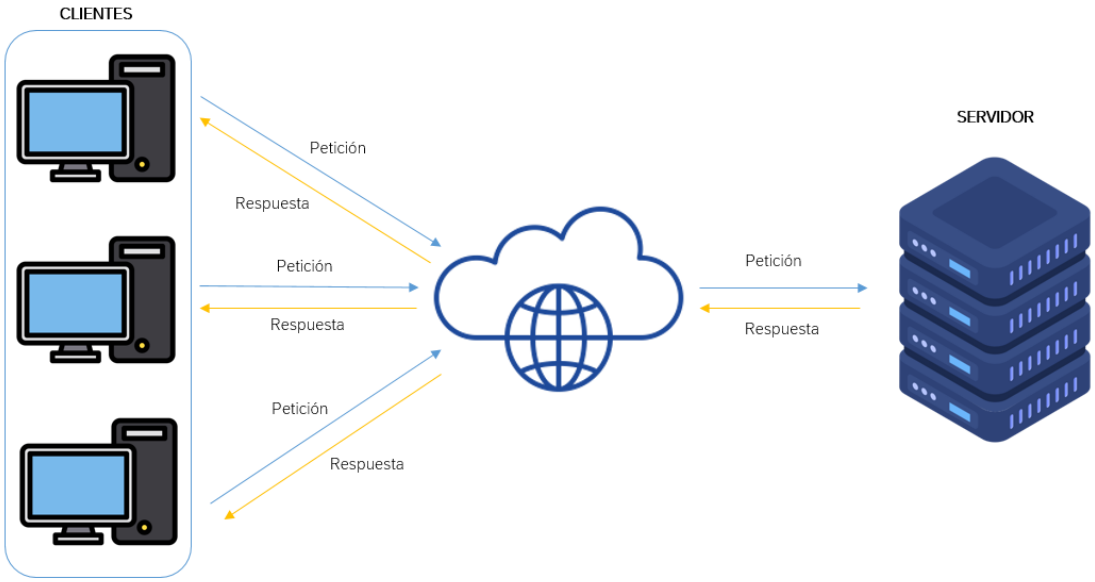

Figura 9: Patrón cliente-servidor

Entre las ventajas más destacables de este patrón encontramos:

- Escalabilidad: Permite la adición de más nodos servidor o la migración a una máquina más potente.
- Integridad: Los datos y el código del servidor se mantienen centralizados, de modo que el mantenimiento requiera de un menor coste.
- Transparencia de localización: El cliente y el servidor pueden residir en máquinas diferentes de la red o en la misma. Un programa puede actuar tanto de cliente como de servidor o como los dos de manera simultánea.
- Independencia de la plataforma: Este patrón es independiente del hardware y software donde se implemente por lo que permite que el cliente y el servidor puedan codificarse en plataformas diferentes.

Entre las desventajas más destacables se encuentran:

- Disponibilidad: En caso de depender de un solo servidor para resolver las peticiones de los clientes, es necesario que este se encuentre disponible en todo momento.
- Congestión del tráfico: En caso de demasiadas peticiones se podrá llegar a dar una congestión, lo cual provocaría una pérdida funcional del servicio.
- Requisitos necesarios: El software y utilizado en un servidor es determinante en la mayoría de ocasiones. Debido a la cantidad de peticiones que se pueden recibir de manera simultánea.

5.2 Diseño detallado

En este punto, se detallan en profundidad los diferentes archivos que componen el sistema desarrollado y su interacción mediante un diagrama de clases. Por otro lado, se exponen los mockups utilizados como referencia para el diseño de la interfaz web que forma el *frontend* de la herramienta.

5.2.1 Diagrama de clases UML

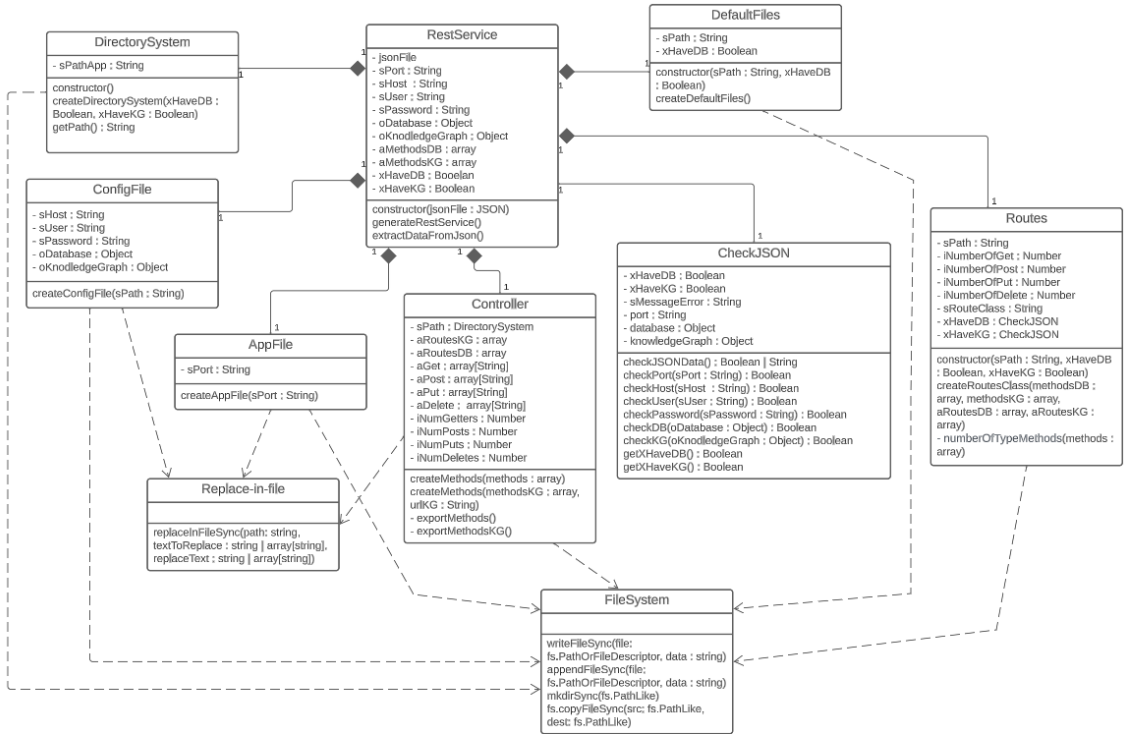

Figura 10: Diagrama de clases UML del sistema

RestService: Esta clase contiene la funcionalidad que genera y distribuye toda la estructura para la creación de la API Rest.

AppFile: Esta clase contiene la funcionalidad que genera el archivo app.js, encargado de poner en marcha la API resultante en el puerto indicado.

Routes: Esta clase contiene la funcionalidad para generar el archivo routes.js que contiene las rutas de los métodos generados por la clase Controller, la ruta de cada método es el nombre del método definido en el archivo de configuración solicitado.

CheckJSON: Esta clase contiene la funcionalidad para comprobar los campos cumplimentados por el usuario en el archivo JSON recibido. Mediante el método *checkJSONData*, se comprueban todos los campos, en caso de error se devuelve lo que ha provocado dicho error y en caso contrario de devuelve *true* indicando que los campos del archivo son correctos.

DirectorySystem: Esta clase contiene la funcionalidad para generar el sistema de directorios del sistema resultante. Esta hace uso del módulo FileSystem como se puede observar en el diagrama de clases uml anterior.

ConfigFile: Esta clase genera el archivo config.js, que mantiene la persistencia de los datos, mediante el método *createConfigFile*.

DefaultFiles: Esta clase contiene la funcionalidad para generar los archivos generales para la puesta en marcha y funcionamiento del sistema. Mediante el método de clase *createDefaultFiles* se generan dichos archivos.

Controller: Esta clase contiene la funcionalidad que genera los archivos controller.js y controllerKG.js del servicio o API Rest resultante. Posee un método de clase para generar cada archivo, *createMethods* para generar el archivo controller.js y *createMethodsKG* para generar el archivo controllerKG.js.

5.2.2 Estructura de directorios

En las siguientes figuras se diferencian dos estructuras, la estructura de directorios de la API generada y la estructura del sistema desarrollado.

Estructura del sistema

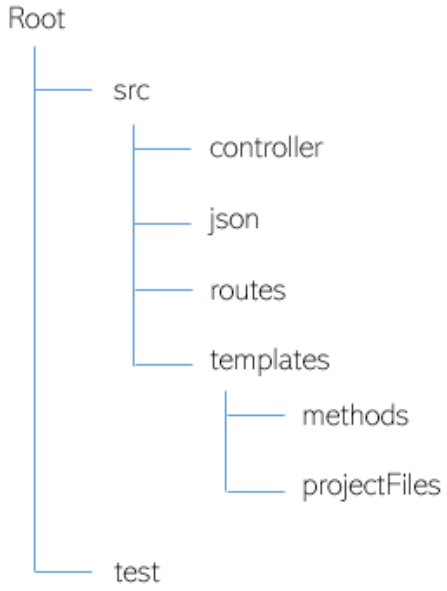

Figura 11: Estructura de directorios del sistema

Root: carpeta contenedora del sistema, en la que se encuentran los archivos package.json, package-lock.json y la carpeta node-modules.

src: carpeta donde se encuentra los archivos que realizan la funcionalidad del sistema, dentro de esta encontramos los archivos AppFile.js, ConfigFile.js, DefaultFiles.js, DirectorySystem.js, Replace.js, RestService.js e index.js.

controller: capeta que contiene la clase Controller.js que genera los archivos get.js, post.js, put.js y delete.js de la API resultante.

json: capeta que contiene la clase CheckJSON.js.

routes: carpeta contenedora la clase Routes.js, encargado de generar el archivo routes.js.

templates: carpeta que contiene las plantillas de los archivos necesarios para la generación de la API.

methods: contiene los archivos method.js y methodKG.js, plantillas de los métodos CRUD que realizan las llamadas a la base de datos y al grafo de conocimiento.

projectFiles: contiene las plantillas de los archivos básicos para la puesta en marcha de la API.

Estructura de la API

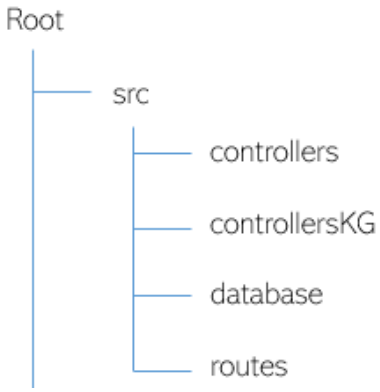

Figura 12: Estructura de directorios de la API Rest

Root: carpeta contenedora de la API, en la que se encuentra el archivo package.json.

src: carpeta donde se encuentra los archivos que realizan la funcionalidad de la API.

controllers: capeta que contiene los archivos con los métodos CRUD de la base de datos.

controllersKG: capeta que contiene los archivos con los métodos CRUD del grafo de conocimento.

Database: carpeta contenedora del archivo database.js que realiza la conexión con la base de datos.

routes: carpeta contenedora del archivo routes.js que contiene las rutas de las llamadas CRUD.

5.2.3 Mockups

El diseño de la interfaz de la herramienta presentada en esta memoria inicia con la creación de unos bocetos de esta previamente a su implementación, lo cual es una parte fundamental en la parte de diseño de cualquier proyecto. Estos han sido creados mediante la página web Canva (Una plataforma de diseño y comunicación visual).

Las figuras siguientes muestran las diversas pantallas y funcionalidades que presenta la interfaz del sistema.

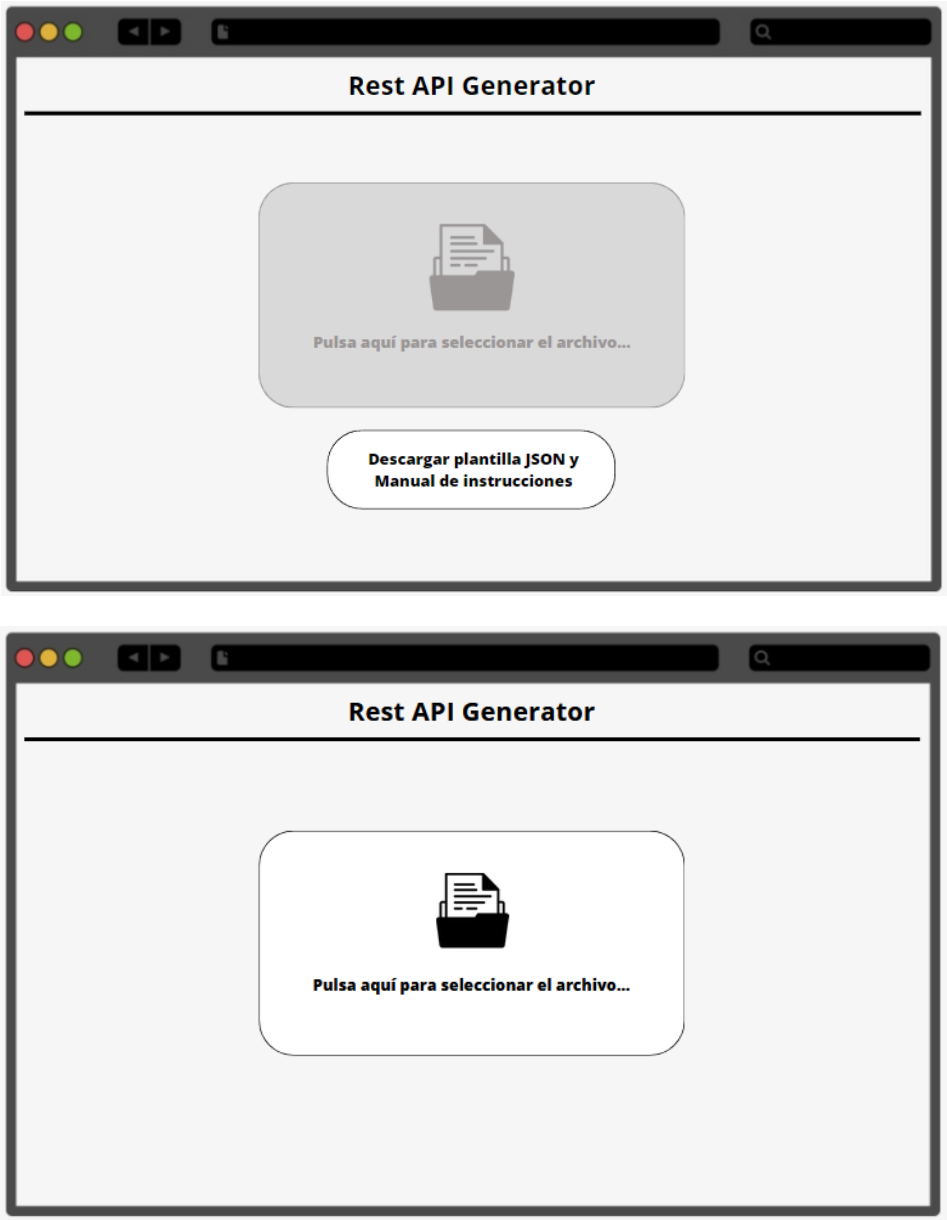

Figura 13: Pantalla inicial

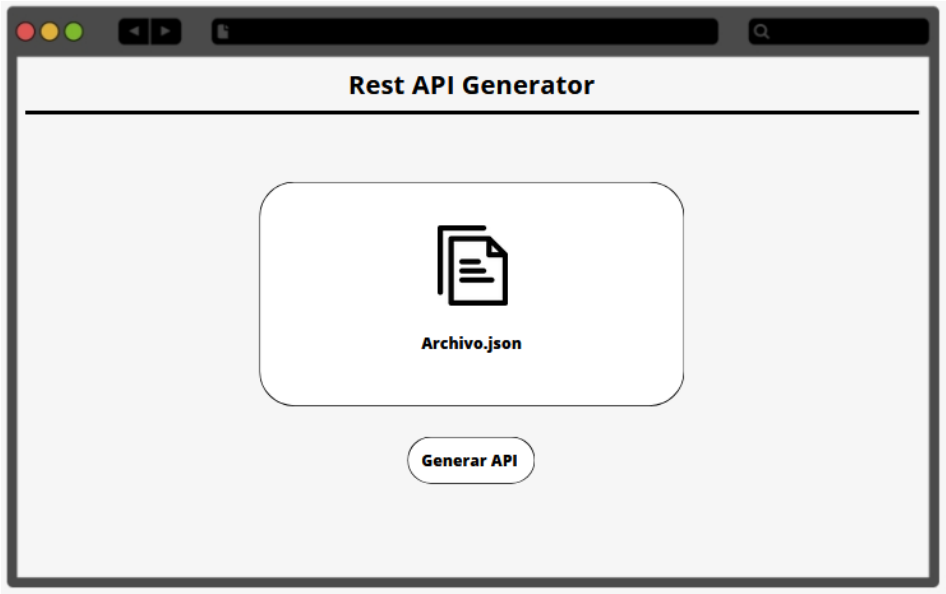

Figura 14: Pantalla inicial - estado 1 - archivo adjunto

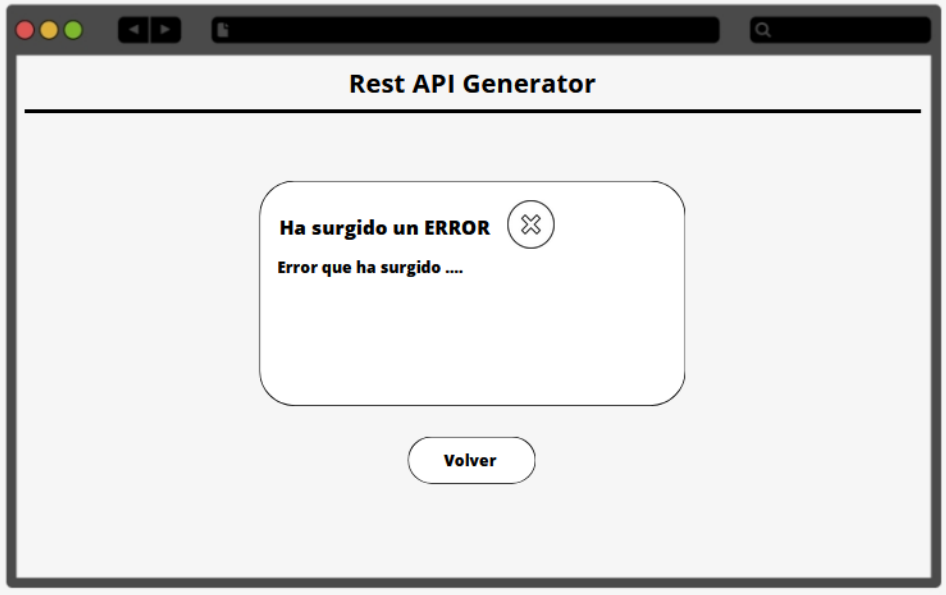

Figura 15: Pantalla de error

Figura 16: Pantalla de espera

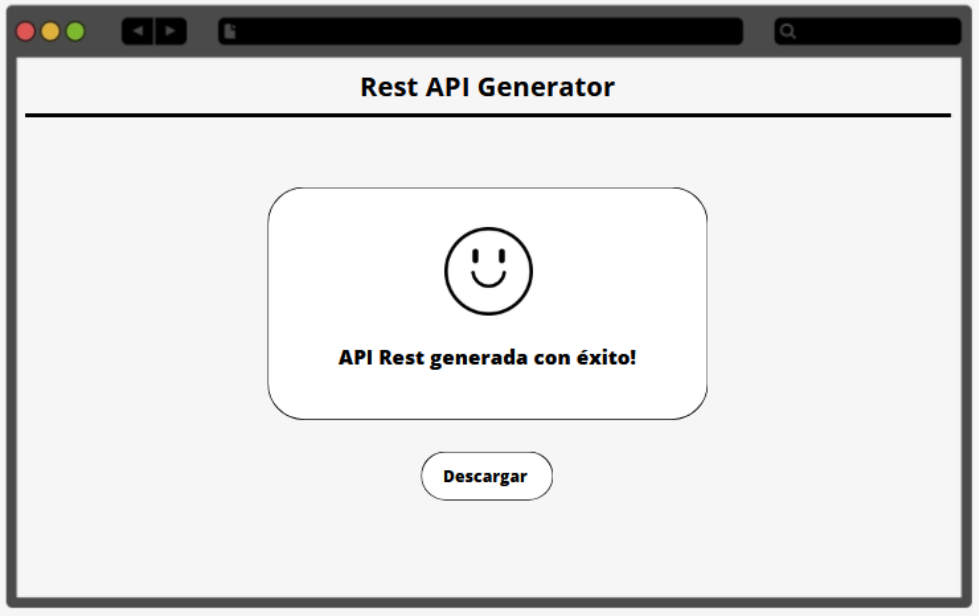

Figura 17: Pantalla de descarga

5.3 Tecnologías utilizadas

En este apartado se describen las herramientas utilizadas para el desarrollo e implementación del sistema.

JavaScript

Es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, es versátil y ampliamente utilizado. Su uso destaca principalmente en el desarrollo web del lado del cliente, aunque también es usado en una gran variedad de entornos, entre los cuales destaca el desarrollo del lado del servidor y dispositivos móviles. Una de sus características más destacables es su tipado dinámico, que permite flexibilidad a la hora de manipular datos.

Node.js

Es un entorno en tiempo de ejecución basado en el motor V8 de Google, de código abierto y multiplataforma que permite ejecutar código JavaScript fuera del navegador.

Visual Studio Code

Es un editor de código fuente, versátil y personalizable, desarrollado por Microsoft para diversos sistemas operativos, tales como Windows, Linux y macOs. Entre sus características más destacables se encuentran la depuración de código, control de versiones con Git, autocompletado inteligente de código, resaltado de sintaxis errónea y refactorización de código.

FileSystem

Es un módulo de Node.js que permite interactuar con los archivos del sistema. Permite tanto la creación, modificación, edición y eliminación de archivos como de directorios de trabajo. Además de poder utilizar llamadas síncronas, asíncronas y promesas.

Express

Es un *framework* que permite crear aplicaciones web y APIs. Proporciona una serie de funciones y herramientas que facilitan la creación de rutas, el manejo de peticiones HTTP, el uso de middleware y la integración con diversos motores de plantillas. Es uno de los *frameworks* más populares de Node.js.

Request

Es una biblioteca que simplifica el envío de solicitudes HTTP/HTTPS desde un servidor o una aplicación Node.js. Permite configurar y personalizar las solicitudes HTTP, pudiendo establecer encabezados, enviar datos en el cuerpo de la solicitud, etc. Además, admite el uso de *proxies*, autenticación básica y *digest*, y tiene soporte para solicitudes multipartes y *streams*.

WebSocket

Es una biblioteca de JavaScript que permite la comunicación persistente en tiempo real entre cliente y servidor. Ambos extremos pueden tanto enviar como recibir información en forma de mensajes, puesto que la comunicación es bidireccional.

Babel

Es una herramienta utilizada para la conversión de código ECMAScript2015+ en una versión compatible con versiones anteriores de JavaScript, permitiendo que el código sea ejecutado en versiones y entornos más antiguos. Garantizando la compatibilidad y rendimiento adecuado del código en muchos entornos.

Promise-mysql

Es un módulo de Node.js que proporciona una interfaz basada en promesas para interactuar con bases de datos MySQL. De manera que simplifique las consultas e intercambio de datos en MySQL evitando el uso de *callbacks* y de esta manera facilitando el manejo de operaciones asíncronas.

Replace in file

Es una herramienta desarrollada en TypeScript para reemplazar textos en uno o más archivos. Funciona de forma síncrona o asíncrona, a elección del usuario, con promesas o *callbacks*. Permite reemplazar diferentes partes de texto de forma simultánea o por separado.

Adm-zip

Es un módulo de Node.js que permite generar un archivo comprimido en formato *zip* a partir de un archivo o directorio y salvarlo en una ruta definida.

XAMPP

Es una distribución gratuita de Apache que incluye MariaDB, PHP y Perl. Esta herramienta permite configurar de manera rápida y sencilla un entorno de desarrollo local. Los componentes utilizados han sido Apache, un servidor web ampliamente utilizado, que permite alojar aplicaciones web y MariaDB una distribución SQL de bases de datos.

6. Desarrollo e implementación

En esta sección se presenta el desarrollo completo que se ha llevado a cabo para implementar el sistema mencionado en esta memoria. Inicialmente se expone el desarrollo de las API Rest desarrolladas que servirán como base para la API resultante. A continuación, se describe el desarrollo del sistema perteneciente a la parte del servidor, el desarrollo de la parte cliente, tanto funcionalidad como interfaz de usuario, y, el desarrollo e implementación de la comunicación entre ambos.

6.1 Desarrollo de las API Rest base

El primer paso para iniciar la creación del sistema ha sido el desarrollo de dos API Rest independientes. De modo que, la primera sea capaz de conectarse a una base de datos mientras que la segunda debe conectarse a un grafo de conocimiento. Ambas deben poder realizar los denominados métodos CRUD (Create, Read, Upload y Delete), es decir, implementar las llamadas GET, POST, PUT y DELETE. Las dos están codificadas en lenguaje JavaScript mediante Node.js, el *framework* Express.js, Request.js y Babel.

Una vez implantadas las dos API, se prepara el entorno para poder comprobar su funcionamiento. En primer lugar, se genera una base de datos con los datos de prueba (los utilizados en este proyecto han sido extraídos de la base de datos de SEMAP), que se encuentra alojada en la distribución SQL MariaDB proporcionada por la herramienta XAMPP. A continuación, se cumplimentan las variables de los archivos necesarios para poder poner en marcha la API, el puerto y *host* donde se va a alojar, el nombre, usuario, contraseña y *host* de la base de datos a la que debe realizar las peticiones.

Por otro lado, se extrae la URL del grafo de conocimiento al que de desea realizar las peticiones, en este caso el grafo escogido es SILKNOW, y, al igual que la primera API, se cumplimentan las variables necesarias para su puesta en marcha, el puerto y *host* donde se va a alojar y la URL del grafo.

Finalmente se comprueba el funcionamiento de las API Rest desarrolladas y la respuesta de las llamadas realizadas con cada una de ellas. Sin embargo, en caso de la segunda API Rest solo se ha podido comprobar el funcionamiento del método GET puesto que no se disponen de permisos para realizar llamadas con los métodos restantes, pese a esto el código desarrollo que realiza la llamada al grafo de conocimiento es el mismo, de modo que la única diferencia entre un tipo de llamada u otro es el valor introducido en el parámetro *query* y el nombre del método.

6.2 Desarrollo del sistema

Una vez implementadas las dos API Rest y comprobado su correcto funcionamiento, se realizan las plantillas de código (basadas en las API Rest desarrolladas) que conforman las bases para la posterior generación de la API resultante. Estas se encuentran en la carpeta **templates** del proyecto, donde se observan las plantillas correspondientes a los archivos comunes (index.js, app.js, babel.rc, database.js y config.js) y las dos plantillas de los métodos, una para los métodos correspondientes a la API de la base de datos y otra para los del grafo de conocimiento que se encuentran en la carpeta **methods**. En las Figura 18 y Figura 19 se muestran las plantillas utilizadas como base para los métodos CRUD.

```
const METHOD_NAME = async(request, response)=>{
    try {
        const connection = await getConnection();
        const result = await connection.query("QUERY");
        response.json(result);
    } catch (error) {
        response.status(500);
        response.send(error);
    }
};
```

Figura 18: Plantilla - Método de acceso a la base de datos

```
const METHOD_NAME = async (req, res) => {
    request("URL_KG" + encodeURIComponent("QUERY"), (error, response, body) => {
        if(error){
            res.json(error);
        } else {
            res.json(body);
        }
    });
};
```

Figura 19: Plantilla - Método de acceso al grafo de conocimiento

De manera paralela se codifica la funcionalidad para la extracción de los elementos del archivo de configuración en formato JSON que debe aportar el usuario, la cual se encuentra implementada en la clase RestService.js, estos valores guardan en variables locales para su posterior uso. La funcionalidad para comprobar los campos se codifica en la clase CheckJSON.js, se comprueba en primera instancia que los campos no estén vacíos, es decir, que al menos el nombre del campo este en al archivo y tenga un valor asignado, y posteriormente se realizan las comprobaciones pertinentes a cada campo.

A continuación, comienza el desarrollo de la funcionalidad para generar de manera automática la API resultante a partir de las plantillas. En primer lugar, se implementa la funcionalidad para crear el sistema de directorios del sistema, codificada en la clase DirectorySystem.js el cual crea la arquitectura de directorios, mencionado en el punto *Diseño detallado*, mediante el módulo FileSystem de Node.js.

En segundo lugar, se implementa la funcionalidad para la generación de los archivos necesarios para la implementación y funcionamiento del sistema resultante, codificada en las clases CreateDefaultFiles.js, ConfigFile.js y AppFile.js. La plantilla del archivo app.js contiene el puerto donde se ejecutará el servidor que contiene la API, por lo que este debe ser sustituido por el aportado por el usuario en el campo LISTENING_PORT del archivo de configuración. Para ello se hace uso del módulo *replace-in-file* de Node.js con el cual se reemplaza el texto "PORT" por el valor recibido. Del mismo modo, la plantilla del archivo config.js contiene el host, user, password, database y knowledgeGrapUrl que se reemplazan simultáneamente con el mismo módulo por los valores obtenidos del archivo de configuración.

Seguidamente, se codifica la funcionalidad para implementar los métodos obtenidos por el usuario tanto de la base de datos como del grafo de conocimiento. Para ello, se crea un archivo auxiliar por cada tipo de llamada y se reemplazan el texto METHOD_NAME y la QUERY del mismo, este se va sobrescribiendo con cada método. Una vez finalizado, se obtienen los archivos get.js, post.js. put.js y delete.js que contiene el número de métodos respectivo (En caso de no que se requiera un tipo de llamada, no se genera el archivo correspondiente), tanto de la base de datos como del grafo de conocimiento y se encuentran en las carpetas **controllers** y **controllersKG** respectivamente. Paralelamente, se implementa la funcionalidad para definir las rutas en la clase Routes.js mediante las cuales realizar las llamadas, codificada en el archivo routes.js generado. El cual contiene una ruta independiente por cada método.

Por último, se comprueba que la API Rest resultante se genera y funciona de manera correcta. Una vez verificado, se codifica la funcionalidad para generar un archivo comprimido en formato ZIP a partir de un directorio o carpeta mediante el módulo Adm-zip (Adicionalmente se probaron las herramientas JSZip y Zip-dir pero se descartaron debido a la sencillez y rapidez que aporta Adm-zip).

6.3 Desarrollo de la interfaz gráfica o *front-end*

El *front-end* del sistema se desarrolla en html junto a un archivo JavaScript que aporta la funcionalidad y un archivo css que da estilo a la interfaz. Los iconos que aparecen en la interfaz se han extraído del conjunto de herramientas Font Awesome por medio de un elemento <link> y una URL CDN (Content distribution network o Red de distribución de contenido).

La interfaz se divide fundamentalmente en 4 pantallas, como se ha indicado en los *mockups* del punto *Diseño detallado*.

En primera instancia, la pantalla inicial en la cual se permite descargar la plantilla del archivo de configuración, junto con un manual de instrucciones para cumplimentarla adecuadamente, subir el archivo de configuración en formato JSON e iniciar la generación de la API resultante. Esta pantalla presenta un elemento <form> que contiene elemento <input> que permite la elección de archivos, además, presenta un capo "accept" que indica el tipo del archivo que está permitido seleccionar, en este caso

este campo presenta el valor “application/json”, utilizado para archivos de tipo JSON. Una vez seleccionado el archivo se visualiza un botón (Codificado mediante su atributo “display” con los valores *block*, el botón se visualiza en pantalla, o *none*, el botón se invisibiliza o destruye y no ocupa espacio en la interfaz, dependiendo de la pantalla que se quiera mostrar), con el texto “Generate API”, que manda un mensaje al servidor para iniciar el proceso de generación de la API.

En segundo lugar, la pantalla de espera, que es ni más ni menos una pantalla de carga mientras se genera el código. Esta es visible una vez se ha pulsado el botón para iniciar la generación del servicio Rest. Presenta un texto informativo indicando que la generación del código está en marcha. Una vez finalizada la generación esta pantalla se sustituye por la pantalla de descarga o fallos en función del valor enviado por el servidor una vez ha finalizado el proceso o abortado el mismo.

En caso de surgir un error, se mostrará la pantalla de error, donde se visualizan los errores que puedan surgir de la comprobación de los campos del archivo de configuración.

Finalmente, la pantalla de descarga, donde se muestra un botón de descarga para obtener la API enviada en formato Zip por el servidor.

6.4 Desarrollo de la comunicación front-end y back-end

La comunicación entre el *front-end* y *back-end* del sistema se realiza mediante el uso del protocolo WebSocket de JavaScript. Esta herramienta ofrece un método de intercambio de datos entre el navegador (Cliente) y el servidor por medio de una conexión persistente y bidireccional.

El funcionamiento es el siguiente, el navegador envía una petición al servidor para establecer la conexión, una vez se establezca dicha conexión, el navegador enviara un mensaje al servidor con el archivo de configuración aportado por el cliente. En el lado del servidor, este recibirá el archivo, lo guardará localmente e iniciará el proceso de generado de la API. En caso de surgir algún error durante el proceso, este finalizará y el servidor mandará un mensaje al cliente con la información del error que ha surgido. En caso contrario el mensaje contendrá el archivo “api.zip”, el cual se enviará mediante un *buffer*.

7. Pruebas

En este apartado se presentan las pruebas realizas al sistema para conocer si cumple con los objetivos propuestos y si existen fallos en su codificación. A continuación, se presentan las pruebas unitarias, del sistema, verificación y validación realizadas, junto con la herramienta utilizada para la realización de algunas de ellas.

7.1 Herramienta

Jest

Es una herramienta de *testing* desarrollada por Facebook, pensada en primera instancia para React y React Native (Un *framework* de desarrollo de aplicaciones creado por Facebook), pero puede usarse en cualquier marco que utilice JavaScript como lenguaje de codificación.

Esta herramienta ha sido escogida debido a las características mencionadas a continuación:

- Velocidad: Cuando las pruebas están asociadas a la CPU (Central Processing Unit o Unidad Central de Procesamiento), Jest es muy rápido debido al uso de paralelización, la ejecución de las pruebas más lentas en primera instancia, y en caso del uso combinado con Babel, el almacenamiento en memoria caché las transformaciones del código, reduciendo las transformaciones intensivas en CPU.
- Configuración e instalación sencilla: Se instala utilizando yarn o npm en la carpeta contenedora del proyecto o en un directorio vacío y las pruebas pueden crearse en cualquier directorio. Los archivos deben llevar la extensión “nombre.test.js”.
- Facilidad de uso: La creación de un test y la comprobación de su resultado es muy sencilla y mediante la ejecución del comando “npm jest” se lanzan todos los test a ejecución.
- Informes de cobertura: Permite la creación de informes para comprobar la cobertura de las pruebas en diferentes puntos del sistema.

Se había considerado realizar pruebas de rendimiento de la aplicación mediante el uso de la herramienta JMeter, para simular de manera simultánea varios usuarios usando el sistema, pero finalmente esta idea ha sido descartada debido a que el rendimiento depende del servidor donde se encuentre alojada la aplicación y por ello se mediría el rendimiento del servidor no de la aplicación en sí misma.

7.2 Pruebas unitarias

En este punto se presentan las pruebas unitarias del sistema, estas son pruebas realizadas a los diferentes módulos del sistema y comprueban si estos proporcionan el resultado esperado y cumplen las funciones requeridas.

En primer lugar, se para verifican los métodos de comprobación de la clase CheckJSON.js. En total son 17 test para comprobar su funcionamiento, el código mostrado en la Figura 20: Test de la clase CheckJSON.js indica la implementación de estos. En segundo lugar, se prueban métodos de la clase CreateRestService.js, de modo que la salida devuelta sea la esperada, debido a que la salida de esta clase es la creación y/o edición de diferentes archivos no es posible comprobar de manera certera el funcionamiento. Por lo que, se han comprobado mediante *testing* solamente algunas partes y otras de forma manual, como la comprobación de la generación de un archivo o la correcta edición de un archivo.

```
import { CheckJSON } from "../src/CheckJSON.js";
const checkJson = new CheckJSON();

test('Check JSON Test: comprobando el parametro LISTENING_PORT', () => {
    const result = checkJson.checkPort("1ab234");
    expect(result).toBe(false);
});

test('Check JSON Test: comprobando el parametro LISTENING_PORT', () => {
    const result = checkJson.checkPort("ABCD");
    expect(result).toBe(false);
});

test('Check JSON Test: comprobando el parametro LISTENING_PORT', () => {
    const result = checkJson.checkPort("90000");
    expect(result).toBe(false);
});

test('Check JSON Test: comprobando el parametro LISTENING_PORT', () => {
    const result = checkJson.checkPort("-1");
    expect(result).toBe(false);
});

test('Check JSON Test: comprobando el parametro LISTENING_PORT', () => {
    const result = checkJson.checkPort("1234");
    expect(result).toBe(true);
});
```

Figura 20: Test de la clase CheckJSON.js

7.3 Pruebas del sistema o pruebas de integración

En este punto se exponen las pruebas del sistema que comprueban el diseño del sistema desarrollado, es decir, si cada uno de los componentes del sistema interactúa de manera correcta con el resto.

Este sistema se compone de dos componentes el *front-end* y *back-end*. De manera que, en primer lugar, se comprueba que las distintas clases que componen el *back-end* interactúen de manera correcta entre sí. Para revisar dicho funcionamiento, se realizan una serie de pruebas con Jest que van generando de manera progresiva el servicio resultante. Estas pruebas no tienen resultado, ya que, Jest no tiene manera de comprobar la generación de archivos o directorios, por lo que se ha comprobado de manera manual.

Por otro lado, se comprueba si se realiza de manera adecuada la comunicación entre estos. Para ello, se ponen en marcha los dos componentes y realizan una serie de intercambios de mensajes mediante la biblioteca WebSocket.js corroborando que los dos componentes interactúan entre sí y de manera correcta.

Para en el *back-end* se ha levantado un servidor en local y escuchando en el puerto 8080. Una vez este establezca una conexión mostrará por consola el mensaje "Nueva conexión establecida". Y se mantendrá a la espera de un mensaje por parte del cliente. En caso de ser el primer mensaje, es decir, el cliente envíe un mensaje con el contenido "Saludo", el servidor enviará un mensaje de vuelta con el contenido "Saludo de vuelta". En caso de que el cliente envíe el mensaje "Despedida", el servidor cerrará la conexión con el cliente.

En la parte cliente, este enviará un mensaje con el contenido "Saludo" una vez realizada la conexión y un mensaje con el contenido "Despedida" cuando reciba por parte del servidor el mensaje "Saludo de vuelta". La Figura 21: Salida del terminal del servidor y Figura 22: Salida del terminal del cliente representan la funcionalidad mencionada.

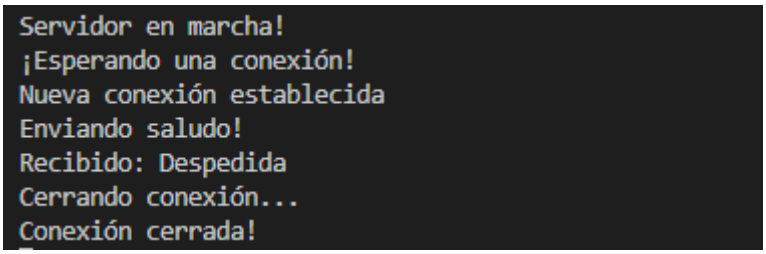

Figura 21: Salida del terminal del servidor

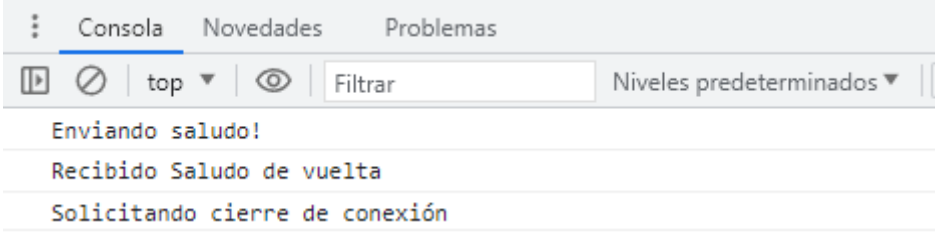

Figura 22: Salida del terminal del cliente

7.4 Verificación

En este punto se verifica si se han cumplido los requisitos del sistema definidos durante el diseño de la arquitectura. Para ello hay que revisar los requisitos y comprobar uno a uno que el sistema funciona de acuerdo con la definición de estos. Estas pruebas se han realizado de forma manual siguiendo una serie de pasos establecidos en lugar de utilizando Jest.js.

En primera instancia, se verifican los requisitos funcionales:

Tabla 18: Verificación de Requisitos Funcionales

RF Probado	**Pasos**	**Resultado**
RF. 1: El usuario deber ser capaz de subir un archivo en formato JSON con un formato definido.	1. Creación de un archivo con formato JSON 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Al subirlo debe aparecer un botón con el texto: “Generate API Rest”	OK
RF. 2: El usuario podrá intercambiar el archivo en formato JSON subido previamente por otro con el mismo formato.	1. Creación de un archivo con formato JSON 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Al subirlo debe aparecer un botón con el texto: “Generate API Rest” 6. Pulsar de nuevo sobre la zona marcada para subir el archivo 7. Seleccionar otro archivo con formato JSON	OK
RF. 3: El usuario deber ser capaz de descargar con éxito el archivo generado.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: “Generate API Rest” 6. Pulsar en el botón con el texto: “Download API” 7. Comprobar que se ha descargado correctamente el archivo en formato ZIP	OK

RF. 4: El usuario no deber ser capaz de pulsar en el botón "Crear API Rest" sin haber subido el archivo de configuración. *RF. 9*: El botón que indica el inicio de la generación del servicio debe estar deshabilitado hasta que el usuario proporcione el archivo de configuración.	1. Acceder a la página web 2. No se visualiza el botón con el texto: "Generate API Rest". Ni sucede ninguna acción al pulsar donde se debería encontrar el botón.	OK
RF. 6: El usuario debe ser capaz de pulsar en el botón "Crear API Rest" una vez ha subido el archivo de configuración.	1. Creación de un archivo con formato JSON 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Debe de aparecer la pantalla de carga	OK
RF. 7: El usuario podrá descargar el código generado al pulsar en el botón "Descargar código".	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga 7. Visualización de la pantalla de finalización 8. Visualización el botón con el texto: "Download API" 9. Pulsar en el botón con el texto: "Download API"	OK
RF. 8: Se proporcionará un mensaje indicativo del resultado de la generación del código.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga	OK

	7. Visualización de la pantalla de finalización con el mensaje de éxito o visualización de la pantalla de error con el error acontecido.	
RF. 10: El botón que permite la descarga del servicio resultante debe estar oculto hasta que termine la generación del código.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga 7. Visualización de la pantalla de finalización con el mensaje de éxito y el botón con el texto "Download API". Hasta esta pantalla este botón no debe haber aparecido en la interfaz.	OK
RF. 11: Se debe cumplimentar al menos una de las dos opciones, base de datos o grafo de conocimiento.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente solamente con la opción de bases de datos. 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga 7. Visualización de la pantalla de finalización con el mensaje de éxito y el botón con el texto "Download API".	OK
RF. 11: Se debe cumplimentar al menos una de las dos opciones, base de datos o grafo de conocimiento.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente solamente con la opción de grafo de conocimiento. 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga	OK

	7. Visualización de la pantalla de finalización con el mensaje de éxito y el botón con el texto "Download API".	

A continuación, se verifican los requisitos no funcionales:

Tabla 19: Verificación de Requisitos no Funcionales

RNF Probado	**Pasos**	**Resultado**
RNF. 1: El número máximo de archivos JSON que se pueden subir debe ser 1.	1. Creación de dos o más archivos con formato JSON 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar más de un archivo. 5. Al subirlo debe aparecer un botón con el texto: "Generate API Rest" y solamente debe aparecer el nombre del último archivo seleccionado.	OK
RNF. 3: El servicio generado debe de ejecutarse en un servidor mediante el *framework* Node.js.	1. Creación de un archivo con formato JSON y cumplimentarlo correctamente 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga 7. Visualización de la pantalla de finalización 8. Visualización el botón con el texto: "Download API" 9. Pulsar en el botón con el texto: "Download API" 10. Extraer la carpeta src del archivo descargado e introducirla en la carpeta deseada 11. En caso de no tener instalado Node.js intalarlo 12. Ejecutar en la raíz del proyecto el comando de node: *npm i* 13. Ejecutar el comando: *npm run*	OK

RNF. 5: El número mínimo de métodos introducidos para la generación del código debe ser 1. *RNF. 6*: El mensaje indicativo, mostrado durante o cuando finalice la generación del servicio, debe exponer si ha tenido éxito o fracaso. En caso de fracasar indicar el motivo	1. Creación de un archivo con formato JSON y cumplimentarlo sin métodos en cualquiera de las dos opciones. base de datos o grafo de conocimiento 2. Acceder a la página web 3. Pulsar en la zona marcada para subir el archivo 4. Seleccionar el archivo y subirlo 5. Pulsar en el botón con el texto: "Generate API Rest" 6. Visualización de la pantalla de carga 7. Visualización de la pantalla de error, mensaje indicando que el campo "METHODS_XX" está vacío (XX = DB o KG)	OK

7.5 Validación

Por último, se realiza la validación del sistema. En este punto se comprueba el sistema desde el punto de vista del cliente, de manera que se realizan los pasos simulando como el cliente final interactuaría con el sistema. La validación hubiera sido mejor que se realizara por personal ajeno al desarrollo.

1. Para ello se realizan los pasos mencionados a continuación:
2. Pulsar en el botón con el texto "Descargar plantilla JSON y manual de instrucciones".
3. Cumplimentar la plantilla según indica el manual.
4. Pulsar en el recuadro marcado para subir la plantilla en formato JSON cumplimentada.
5. Seleccionar la plantilla.
6. Pulsar en el botón con el texto "Generate API Rest"
7. Una vez finalizada la generación, pulsar en el botón con el texto "Download API"
8. Extraer el código generado en la carpeta deseada y en caso de no tener Node.js instalarlo.
9. Abrir la API con un editor como Visual Studio Code y abrir una terminal, o, entrar en la carpeta donde se encuentre el código mediante la consola de comandos.
10. Introducir el comando "npm i".
11. Una vez finalizada la instalación introducir el comando "npm run".
12. En la terminal debe aparecer la línea "Server running".

8. Conclusiones y trabajos futuros

En esta sección final de memoria se extraen las conclusiones y revisión de objetivos del trabajo realizado durante el desarrollo e implementación del sistema, los problemas que han surgido durante el tiempo de desarrollo de este proyecto y por último los trabajos futuros a realizar.

8.1 Problemas a lo largo del proyecto

Los principales problemas encontrados durante la realización de este proyecto final de carrera han sido las paralizaciones del desarrollo debido a una serie de problemas médicos presentados a lo largo de la realización del proyecto y como consecuencia un retraso en la estimación temporal inicial.

8.2 Conclusiones y revisión de objetivos

En primer lugar, antes de proceder con las conclusiones del proyecto, se realiza una revisión de los objetivos definidos en el punto 1.3 para comprobar si estos se han cumplido satisfactoriamente.

Los objetivos, disminuir la carga de trabajo y agilizar el desarrollo, se podrían dar como cumplidos, ya que, el código de la API Rest se genera de manera automática una vez se ha cumplimentado la plantilla de manera correcta.

El objetivo, minimizar el número de errores cometidos, también se podrá dar como cumplido, ya que, el código se ha realizado mediante una serie de plantillas obtenidas de API Rest, codificadas con anterioridad y verificando su correcto comportamiento. A pesar de esto, la posibilidad de que surjan errores siempre está presente, pues no se puede garantizar una cobertura del 100%.

El objetivo, reducir los conocimientos necesarios del equipo de trabajo, también podría darse como cumplido, ya que, la puesta en marcha de este sistema requiere únicamente de la ejecución de dos comandos y una vez ejecutados el sistema ya se encontraría en funcionamiento.

El objetivo, avanzar un paso en la automatización de servicios, se podrá dar como cumplido ya que el código de la API resultante se genera automáticamente, a partir de una serie de parámetros, de manera correcta.

Una vez revisados los objetivos iniciales, se exponen las conclusiones del proyecto presentado en esta memoria.

La realización de este proyecto me ha permitido llevar a cabo el desarrollo de un proyecto de software desde el inicio hasta el final pasando por todas sus fases intermedias y acercándome un paso más a la realidad del trabajo que conllevan todo este proceso.

Desde el punto de vista técnico, pese a no ser un proyecto de gran magnitud, es un sistema realmente útil y práctico que puede ser usado por todos los programadores y que supone un ahorro de tiempo y recursos considerable.

Por otro lado, desde el punto de vista profesional, me ha permitido aprender muchas tecnologías y herramientas que desconocía, además de poner en práctica muchos métodos de programación que se han ido enseñando a lo largo de la carrera. Me ha enseñado a mejorar mi capacidad de análisis, síntesis, redacción y trabajar bajo unos márgenes de tiempo establecidos, los cuales tienen repercusiones si estos no son alcanzados satisfactoriamente que ocasionan cambios durante el proyecto.

Por último, desde un punto de vista retrospectivo del trabajo realizado, me parece que algunos de los aspectos del proyecto podrían haberse mejorado, como la planificación y estimación del coste temporal asociado a cada tarea; poder realizar una interfaz web mucho más completa y consistente; ofrecer muchas más opciones de personalización como la elección del formato de vuelta de las llamadas a la base de datos o grafo de conocimiento. A pesar de esto, y aun siendo una primera versión de este sistema, que bajo mi punto de vista es de gran utilidad, siento que el trabajo desarrollado ha sido una buena solución al problema planteado.

8.3 Trabajos futuros

En el trabajo futuro relacionado con la mejora y/o adición de nueva funcionalidad a este sistema se pueden considerar las siguientes propuestas:

- Creación de una aplicación local para ordenadores, para poder generar el código de manera local y sin necesidad de acceso a la red.
- Añadir la funcionalidad que permita la generación de la API en diferentes lenguajes como PHP o TypsScript. De manera que puedan existir diversos tipos de plantilla y montaje según el tipo de lenguaje de programación escogido.
- Introducir un mayor grado de personalización, permitiendo elegir el formato de respuesta de la información proporcionada por una base de datos o un grafo de conocimiento.
- Mejorar la interfaz web añadiendo un formulario para introducir los datos necesarios para la generación del código e indicar los errores de cumplimentación del mismo de manera inmediata.

Referencias

Chen, X., Jia, S. and Xiang, Y. (2020). A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*.

Dave Winer. (2007). XML-RPC Specification. UserLand Software, Inc.

Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmann, T., Sun, S. and Zhang, W. (2014). Knowledge vault: a web-scale approach to probabilistic knowledge fusion. KDD '14: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. New York, NY, USA: Association for Computing Machinery.

Ehrlinger, L., & Wöss, W. (2016). Towards a Definition of Knowledge Graphs. Semantics.

InfoJobs. Buscador de salarios. Disponible en: https://salarios.infojobs.net/

Ji S, Pan S, Cambria E. (2021). A survey on knowledge graphs: representation, acquisition, and applications. IEEE Trans Neural Netw Learn Syst

Kong Y, Liu X, Zhao Z. (2022). Bolt defect classification algorithm based on knowledge graph and feature fusion. Energy Rep.

Marqués, M. (2011). Bases de datos. Universitat Jaume I, Servei de Comunicació i Publicacions. Disponible en: https://dspace-libros.metabiblioteca.com.co/handle/001/353

Pellissier Tanon, T., Vrandečić, D., Schaffert, S., Steiner, T. and Pintscher, L. (2016). From Freebase to Wikidata: The Great Migration. WWW '16: Proceedings of the 25th International Conference on World Wide Web. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee.

Richens, R.H. (1956). Preprogramming for mechanical translation. *Mechanical Translation*.

Sevilla Peris, Javier. (2022). VISUALIZACIÓN DEL CONTENIDO DE GRAFOS DE CONOCIMIENTO DEL PATRIMONIO CULTURAL. [Tesis doctoral, Universistat de València]. https://roderic.uv.es/handle/10550/83829

Singhal, A. (2012) Introducing the knowledge graph: things, not strings. Disponible en: https://blog.google/products/search/introducing-knowledge-graph-things-not/

Statista. (2022). Disponible en: https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/ (Accedido el 20/05/2023)

Bibliografía

Adm-zip. ADM-ZIP for NodeJS with added support for electron original-fs. Disponible en: https://www.npmjs.com/package/adm-zip

Babel. What is Babel?. Disponible en: https://babeljs.io/docs/

DB-Engines. DB-Engines Ranking, June 2023: https://db-engines.com/en/ranking (Accedido el 01/06/2023)

Douglas Karr. Automatice la construcción de su API REST, panel de administración y documentación del cartero, 2021. Disponible en: https://es.martech.zone/rest-api-generator-tools/

FileSytem. Node.js v4.9.1 Documentation. Disponible en: https://nodejs.org/docs/latest-v4.x/api/fs.html#fs_file_system

IBM. ¿Qué es SOAP?, 2022. Disponible en: https://www.ibm.com/docs/es/integration-bus/10.0?topic=services-what-is-soap

IBM. La estructura de un mensaje SOAP, 2022. Disponible en: https://www.ibm.com/docs/es/integration-bus/10.0?topic=soap-structure-message

Jestjs.io. Getting Started. Disponible en: https://jestjs.io/

Marqués, Mercedes. Bases de datos. Universitat Jaume I. Servei de Comunicació i Publicacions, 2011. Disponible en: https://dspace-libros.metabiblioteca.com.co/handle/001/353

Node.js. About Node.js. Disponible en: https://nodejs.org/en/about

Postman. Setting up mock servers. Disponible en: https://learning.postman.com/docs/designing-and-developing-your-api/mocking-data/setting-up-mock/

Replace in file. Adam Reis. Disponible en: https://www.npmjs.com/package/replace-in-file

Request - Simplified HTTP client. Disponible en: https://www.npmjs.com/package/request

Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000. Disponible en: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Sandra Milena Velásquez Restrepo, Juan David Vahos-Montoya, Marta Ester Gómez-Adasme, Ana Alexandra Pino – Martínez, Erika Juliett Restrepo-Zapata & Sebastián Londoño-Marín. A comparative review about traditional and modern software development methodologies, 2019. Disponible en: https://revistas.pascualbravo.edu.co/index.php/cintex/article/view/334

Stack Overflow. Stack Overflow Trends, 2022. Disponible en: https://insights.stackoverflow.com/trends?tags=soap%2Crest

Visual Studio Code. Getting Started. Microsoft. Disponible en: https://code.visualstudio.com/docs

WebSocket. Mdn web docs_. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

XAMPP. What is XAMPP?. Apache Friends. Disponible en: https://www.apachefriends.org/

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

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

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

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

El trabajo presentado en esta memoria guarda relación únicamente con un ODS, Industria, innovación e infraestructuras (ODS. 9). Ya que, el sistema desarrollado, se presenta como una innovación parcial de algunos modelos ya existentes y un avance en lo referido a la automatización de sistemas, puesto que, el resultado que se espera obtener, tras la realización de unos simples pasos, es una aplicación completa que pueda comunicarse con almacenes de datos y grafos de conocimiento sin necesidad de codificar una sola línea de código y sin conocimientos necesarios de las herramientas que la componen.

No guarda relación con ningún ODS más debido a que este sistema no se ha realizado con ningún objetivo pensado para la sostenibilidad medioambiental tanto marina como terrestre, no es posible relacionarlo con alguna desigualdad por motivo de género y no está destinado a alguna acción social.

En algún aspecto podría estar relacionado con el ODS. 8, Trabajo decente y crecimiento económico. Debido a que, uno de los objetivos de este sistema es reducir el coste de trabajo de los proyectos y el coste económico de estos.

VNIVERSITAT DE VALÈNCIA

Yo, Javier Sevilla Peris, con NIF 52644948H, profesor de la Escuela Técnica Superior de Ingeniería de la Universidad de Valencia (ETSE) y miembro del equipo del proyecto SEMAP, desarrollado por la Universidad de Valencia y el Instituto de Robótica y Tecnologías de la Información y Comunicación (IRTIC) cedo los datos utilizados en el proyecto SEMAP a Diego Ramírez Carrilero, con NIF 53876406X.

La cesión de dichos datos se limita exclusivamente a la investigación y desarrollo del Trabajo Fin de Grado que Diego Ramírez Carrilero está desarrollando en la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politécnica de Valencia.

Firmado:

SEVILLA PERIS VICENTE JAVIER - 52644948H

Firmado digitalmente por SEVILLA PERIS VICENTE JAVIER - 52644948H
Fecha: 2023.09.06 12:56:30 +02'00'

Javier Sevilla Peris

Valencia, 6 de septiembre de 2023.