UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Proyecto de Fin de Grado en Ingeniería en Tecnologías de la Información

RDFplay: UN PLAYGROUND PARA FORMACIÓN EN TECNOLOGÍAS DE LA WEB SEMÁNTICA

MIGUEL EXPÓSITO MARTÍN

Dirigido por: JOSÉ LUIS FERNÁNDEZ VINDEL

Co-dirigido por: RAFAEL MARTÍNEZ TOMÁS

Curso: 2017-2018: 2ª Convocatoria

RDFplay: UN PLAYGROUND PARA FORMACIÓN EN TECNOLOGÍAS DE LA WEB SEMÁNTICA

Proyecto de Fin de Grado de modalidad oferta específica

Realizado por: Miguel Expósito Martín

Dirigido por: José Luis Fernández Vindel

Co-dirigido por: Rafael Martínez Tomás

Tribunal calificador

Presidente: D/Dª.

Secretario: D/Dª.

Vocal: D/Dª.

Fecha de lectura y defensa: 2/10/2018

Calificación:

Agradecimientos

A mi familia, a mis amigos, a mi Director de proyecto y a los que, sin querer, me animaron a afrontar este reto.

Resumen

El presente proyecto tiene como objeto el desarrollo de un sistema de *playground* para *SPARQL Protocol And Query Language* (SPARQL) y *Resource Description Framework* (RDF) que habrá de servir como herramienta educativa de introducción a las tecnologías de la Web Semántica. Su propósito es facilitar una interfaz de uso sencilla para que usuarios profanos puedan mantener una primera toma de contacto en este tipo de tecnologías a un nivel básico, permitiendo la realización de actividades académicas sobre manipulación sencilla de grafos o sobre consultas SPARQL al conjunto de datos de trabajo y a *endpoints* externos.

Si bien es cierto que la Web Semántica alcanza hoy en día un estado de madurez razonablemente avanzado, el ritmo de cambio de las tecnologías de *frontend* en *Javascript* (o JS) es, cuanto menos, vertiginoso. Unido a ello, se tiene que la mayor parte de las implementaciones de componentes de la Web Semántica han sido desarrolladas en tecnologías de *backend* como *Java* (*Apache Jena*) o *Python* (*rdflib*), no existiendo aún estándares o implementaciones maduras puramente en Javascript.

Se abordan, por tanto, tres retos: la necesidad de conseguir un producto sencillo y fácilmente utilizable por usuarios no expertos, la dificultad para encontrar componentes maduros que aúnen Web Semántica con *frontend* y la conveniencia de apostar por un *framework* de desarrollo en *Javascript* estable y con una curva de aprendizaje suave.

Para dar solución a estos problemas, se ha optado por desarrollar una *Single Page Application* (SPA) con *Vue.js* (un *framework* JS que se jacta de aglutinar las mejores características de *Angular* y *React*, sus principales competidores en el mercado) e integrarlo con las implementaciones recomendadas por el grupo de trabajo de bibliotecas *Javascript rdfjs* y con una plataforma de consultas para la web flexible y modular.

Dado el alto nivel de incertidumbre encontrado a la hora de implementar las necesidades del proyecto, se ha utilizado un enfoque metodológico incremental e iterativo basado en *Extreme Programming* y apoyado sobre tableros Kanban, lo que ha derivado en una mejor organización y evolución del proyecto.

El resultado final ofrece un producto básico de introducción a las tecnologías de la Web Semántica y permite pensar en un desarrollo del mismo tan ambicioso como las propias necesidades de los equipos docentes, dado el estado de madurez actual del *frontend* web.

Palabras clave: educación, *Javascript*, Web Semántica, *Vue.js*, RDF, SPARQL, ontologías.

RDFplay: a playground for education on the Semantic Web technologies

Abstract

The aim of this project is to develop a playground system for SPARQL and RDF that will serve as an educational tool for introducing Semantic Web Technologies. Its purpose is to provide a user-friendly interface for profane users to have their first contact in this kind of technologies at a beginner level, allowing them to carry out academic activities on simple graph manipulation or SPARQL queries to the work dataset and to external endpoints.

While it is true that the Semantic Web today reaches a reasonably advanced stage of maturity, the rate of change of frontend technologies in *Javascript* (or JS) is, at least, vertiginous. Moreover, most of the Semantic Web components implementations have been developed in backend technologies such as *Java* (*Apache Jena*) or *Python* (*rdflib*), with a lack of existing standards or pure *Javascript* mature implementations yet.

Three challenges are therefore addressed: the need to obtain a product both simple and easy to use by non-expert users, the difficulty of finding mature components combining the Semantic Web and frontend and the convenience of betting on a stable *Javascript* development framework with a smooth learning curve.

To solve these problems, the choice made is to develop a SPA with *Vue.js* (a JS framework that boasts of agglutinating the best features of *Angular* and *React*, its main competitors in the market) and integrate it with the *RDF Javascript libraries working group* recommended implementations and a flexible and modular web query platform.

Given the high uncertainty found when implementing the project needs, an incremental and iterative methodological approach based on *Extreme Programming* and supported on Kanban boards has been used, which has led to a better project organization and evolution.

The final result provides for a Semantic Web technologies introduction basic product and allows to think of a development as ambitious as the own teaching teams needs, given the current maturity state of the web *frontend*.

Keywords: education, *Javascript*, Semantic Web, *Vue.js*, RDF, SPARQL, ontologies.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

No cabe duda que uno de los retos a los que se enfrenta la Sociedad de la Información y del Conocimiento es la educación, una de las mejores inversiones en el futuro de Europa. Según la última *Comunicación de la Comisión al Parlamento, al Consejo, al Comité Económico y Social Europeo y al Comité de las Regiones*[1], el uso de las *Tecnologías de la Información y las Comunicaciones* (TIC) para propósitos educativos se queda atrás, tanto en adopción como en competencias del profesorado. La innovación en sistemas educativos, entendida como la interiorización de nuevos servicios y herramientas por parte de las propias organizaciones educativas, puede mejorar los resultados formativos y la eficiencia.

Siendo una de las prioridades del Plan de Acción propuesto por la Comisión hacer un mejor uso de la tecnología digital para el aprendizaje y la enseñanza, parece apropiado que instituciones como la *Universidad Nacional de Educación a Distancia* (UNED) en España y sus departamentos, como el de Inteligencia Artificial, se propongan aplicar los últimos avances tecnológicos para mejorar la calidad del sistema educativo y facilitar a los alumnos la adquisición de nuevas competencias digitales complejas.

Dentro de estas competencias se pueden encuadrar términos como Web de Datos, Web Semántica o Datos Enlazados, todos definidos por el padre de la Web, Tim Berners-Lee[2, 3]. No son más que la evolución lógica de la Web a la que todos estamos acostumbrados, pero entendida en su crecimiento como una solución a los problemas de la ingente cantidad de datos no estructurados o inconexos que se genera al día en 2018 (según Domo[4], 2,5 quintillones de bytes).

En concreto, este trabajo pretende acercar las tecnologías de la Web Semántica al alumnado siguiendo las recomendaciones del *W3C RDFJS Community Group*[5] a través de una sencilla aplicación desarrollada puramente en *Javascript* y lista para usar, que permita la exploración de facilidades de modelado y consulta sobre un conjunto de datos predefinido o aportado por el propio alumno, así como la carga y realización de actividades simples propuestas por el profesorado.

La memoria de este proyecto se estructura en los siguientes capítulos:

- **Capítulo 1. Introducción.** Esta introducción.
- **Capítulo 2. La Web Semántica.** Presentación de las tecnologías que conforman la Web Semántica: datos enlazados, RDF y SPARQL.
- **Capítulo 3. Motivación y objetivos.** Exposición de las razones que llevaron a la elección de este proyecto en concreto, así como las metas que se esperaban alcanzar.
- **Capítulo 4. Trabajos previos, estado actual y recursos.** Un estudio sobre el estado actual de las tecnologías en cuestión y los recursos que se estiman necesarios para llevar a cabo con éxito el proyecto.
- **Capítulo 5. Metodología y planificación.** Descripción de la metodología de desarrollo de software a utilizar y análisis de las planificaciones llevadas a cabo (inicial y efectiva).
- **Capítulo 6. Análisis.** Presentación del estudio de necesidades para el sistema de información concretadas en casos de uso detallados.
- **Capítulo 7. Diseño.** Descripción de las distintas arquitecturas del sistema, así como su estructura basada en componentes.
- **Capítulo 8. Implementación y pruebas.** Detalles de implementación en *Javascript* y de las pruebas unitarias y extremo a extremo llevadas a cabo.
- **Capítulo 9. Resultados.**
- **Capítulo 10. Conclusiones y líneas futuras de trabajo.**

Capítulo 2

La Web Semántica

Cabe comenzar el desarrollo de la presente memoria con una breve introducción a los fundamentos de la Web Semántica y las tecnologías que la sustentan: RDF y SPARQL. Asimismo, queda definido también el término de datos enlazados o *linked data*, estrechamente relacionado con estos conceptos y de gran importancia hoy en día en la difusión de datos estructurados a través de Internet.

2.1. Fundamentos

La Web Semántica es un término originalmente acuñado por Tim Berners-Lee en el año 2001[2]. Hasta la fecha, la *World Wide Web* había sido concebida como una idea de colaboración abierta en la que múltiples contribuciones de varios autores podían tener cabida y ser compartidas universalmente. Dichas contribuciones, realizadas en forma de documentos, estaban dirigidas a personas y no a máquinas o computadores. Berners-Lee vio más allá y propuso su extensión para lograr su manipulación automática; en resumidas cuentas, permitir que agentes inteligentes (programas de ordenador) fueran capaces de encontrar datos y su significado a través de hiperenlaces a definiciones de términos clave y reglas de razonamiento e inferencia lógica.

Para lograr tan ambicioso objetivo, los agentes inteligentes necesitarían tener acceso a contenido y conocimiento estructurado, así como a las reglas de inferencia necesarias. En este contexto, la Web Semántica podía apoyarse en las siguientes tecnologías ya existentes:

- **RDF**, que permite expresar conocimiento en forma de tripletas o ternas (a modo de sujeto, verbo y objeto en una oración) utilizando URIs (*Uniform Resource Indicators*[a]).
- **XML**, un metalenguaje que permite crear lenguajes de etiquetado o marcado y documentos a su vez convenientemente etiquetados y con estructura arbitraria.

[a] Cadenas de caracteres que identifican los recursos de una red de forma unívoca.

Sin embargo, un tercer pilar era necesario para resolver la problemática de la existencia de distintos identificadores en distintas bases de datos relacionadas con el mismo significado conceptual. Gracias a las **ontologías**, documentos que definen formalmente las relaciones entre distintos términos, estas diferencias podían ser salvadas bien mediante el uso de términos estandarizados bien mediante la definición de relaciones conceptuales de igualdad o similitud entre dichos términos.

En palabras del propio Berners-Lee: *«Con un diseño adecuado, la Web Semántica puede ayudar en la evolución del conocimiento humano como un todo.» ([2])*

2.2. Linked Data

Estrechamente relacionado con la Web Semántica se encuentra el concepto de datos enlazados o *Linked Data*, también propuesto por Berners-Lee en 2006[3] en la que se considera como su introducción oficial y formal. Se trata de un movimiento respaldado por el propio W3C[1] que se centra en conectar conjuntos de datos a lo largo de la Web y que puede verse como un subconjunto de la Web Semántica.

Se basa, por tanto, en dos aspectos clave:

- La publicación de conjuntos datos estructurados en línea.
- El establecimiento de enlaces entre dichos conjuntos de datos.

La Web Semántica es el fin y los datos enlazados proporcionan el medio para alcanzar dicho fin.

2.3. RDF

Las tecnologías de la Web Semántica permiten, por tanto, almacenar conocimiento en forma de conceptos y relaciones a través de ternas o tripletas, modelar dominios de conocimiento con vocabularios estándar y ofrecer potentes facilidades de consulta sobre dichos modelos. Dentro de

[1] https://www.w3c.es/

estas tecnologías, RDF es, sin duda, el bloque de construcción fundamental (como *HyperText Markup Language* (HTML) lo ha sido para la Web).

RDF fue propuesto por el W3C en 1999[6] como un estándar para crear y procesar metadatos con el objetivo de describir recursos independientemente de su dominio de aplicación, ayudando, por tanto, a promover la interoperabilidad entre aplicaciones.

Sin embargo, con la llegada de la Web Semántica, el alcance de RDF creció y ya no sólo se usa para codificar metadatos sobre recursos web, sino también **para describir cualquier recurso así como sus relaciones** en el mundo real.

El nuevo alcance de RDF quedó reflejado en las especificaciones publicadas en 2004 por el *RDF Core Working Group*[2], ahora actualizadas a fecha de 2014[3], de las que se pueden extraer las siguientes definiciones de RDF:

Recomendación	Definición
RDF Primer	Un marco de trabajo para expresar información sobre recursos.
RDF Concepts & Syntax	Un marco de trabajo para representar información en la Web.

Cuadro 2.1: Definiciones de RDF en las recomendaciones del W3C.

2.3.1. Modelo Abstracto de RDF

RDF ofrece un modelo abstracto que permite descomponer el conocimiento en pequeñas piezas llamadas sentencias (*statements*), tripletas o ternas y que toman la forma:

Sujeto - Predicado - Objeto

En donde *Sujeto* y *Objeto* representan dos conceptos o "cosas" en el mundo (también denominados **recursos**) y *Predicado* la relación que los conecta.

[2]Recomendaciones del W3C, https://www.w3.org/2001/sw/RDFCore/
[3]Versiones actualizadas de las recomendaciones del W3C, https://www.w3.org/2011/rdf-wg/wiki/Main_Page

Los nombres de estos recursos (así como de los predicados) han de ser globales y deberían identificarse unívocamente por un *Uniform Resource Identifier* (URI).

Por tanto, un modelo RDF puede expresarse como una colección de tripletas o un grafo, dado que aquellas no son más que grafos dirigidos. Los sujetos y objetos serían los nodos del grafo y los predicados sus aristas.

Otra nomenclatura utilizada para representar una terna sería la siguiente:

Recurso - Propiedad - Valor de la propiedad

En donde el valor de la propiedad u *Objeto* también puede ser un literal, que consiste simplemente en un dato textual bruto.

Cabe destacar que una terna RDF sólo puede modelar relaciones binarias. Para modelar una relación n-aria, suelen utilizarse recursos intermedios como nodos en blanco, que no son más que sujetos u objetos que no tienen un URI como identificador (nodos sin nombre o anónimos).

En resumen:

Sujeto	Puede ser un URI o un nodo en blanco.
Predicado o Propiedad	Debe ser un URI.
Objeto o Valor de la Propiedad	Puede ser un URI, un literal o un nodo en blanco.

Cuadro 2.2: Resumen conceptual de ternas.

A continuación se muestran dos ejemplos de tripletas:

Sujeto	Predicado	Objeto
http://www.uned.es/ia/example#Analista	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://www.uned.es/ia/example#Analista	http://www.w3.org/2000/01/rdf-schema#label	"Analista Informático"

Cuadro 2.3: Ejemplos de tripletas reales.

2.3.2. Aplicaciones prácticas de RDF

Gracias a RDF, es posible construir la Web Semántica y enlazar conjuntos de datos. A continuación se enumeran algunas de sus aplicaciones reales:

- Añadir información legible a los motores de búsqueda.
- Enriquecer un conjunto de datos enlazándolo con conjuntos de datos de terceros.
- Facilitar el descubrimiento de *Application Programming Interface* (API)s, en este caso referido a puntos accesibles a través de la Web a través de su entrelazado.
- Construir agregaciones de datos sobre determinados temas.
- Proporcionar un estándar de intercambio de datos entre bases de datos.
- Enriquecer, describir y contextualizar los datos mediante un enfoque rico en metadatos.

Su principal ventaja frente a los modelos y bases de datos relacionales es una mayor facilidad para evolucionar los grafos, algo que en un sistema relacional es complejo (dado que requiere de modificaciones en las estructuras de almacenamiento de datos y sus consultas asociadas). En otras palabras, su **flexibilidad para modelar lo inesperado**.

Un ejemplo claro a nivel global es *Wikidata*[4], una base de datos de conocimiento abierta que puede ser leída y editada tanto por personas como por máquinas. *Wikidata* actúa como un almacén centralizado para datos estructurados de otros proyectos hermanos como *Wikipedia*, *Wikisource*, etc. En el momento de redacción de esta memoria, cuenta con 50.035.140 elementos de datos que cualquiera puede editar.

De una forma más concreta, es destacable el importante rol de RDF y las tecnologías de la Web Semántica en ámbitos de **búsqueda y clasificación bibliográfica o documental**, pudiendo citarse como ejemplo la publicación en RDF del *Diccionario de Lugares Geográficos* y el *Diccionario y Tesauro de Materias* del Patrimonio Cultural de España, gestionado en la actualidad por el Ministerio de Educación, Cultura y Deporte del Gobierno de España[5].

[4] https://www.wikidata.org/wiki/Wikidata:Main_Page
[5] https://www.mecd.gob.es/cultura-mecd/areas-cultura/museos/destacados/2015/tesauros.html

2.4. SPARQL

Tal y como *Structured Query Language* (SQL) provee (al menos, relativamente) un de un estándar de consulta para las bases de datos relacionales, SPARQL ofrece un lenguaje de consulta estandarizado para grafos RDF.

SPARQL es un acrónimo recursivo para *SPARQL Protocol And RDF Query Language*[7]; es a la vez un lenguaje de consulta de grafos RDF y un protocolo, dado que permite exponer servicios web que acepten consultas en dicho lenguaje.

Entre sus características, cabe citar:

- Recuperación de valores de datos estructurados o semi-estructurados.
- Exploración de conjuntos de datos a través de consultas sobre relaciones desconocidas.
- Ejecución de uniones complejas sobre bases de datos dispersas en una única consulta.
- Transformación de datos RDF desde un vocabulario a otro.

La estructura básica de una consulta SPARQL es la siguiente:

```
# declaraciones de prefijos
PREFIX foo: <http://example.com/resources/>
...
# definición del conjunto de datos
FROM ...
# cláusula de resultado
SELECT ...
# patrón de la consulta
WHERE {
        ...
}
# modificadores de la consulta
ORDER BY ...
```

Listado de código 2.1: Estructura básica de consulta SPARQL.

No es objeto de este proyecto profundizar en la sintaxis del lenguaje, pero sí se incluye a continuación un ejemplo de consulta simple:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
        ?person foaf:name ?name .
}
```

Listado de código 2.2: Consulta devolviendo todos los nombres de personas mencionadas en un grafo.

Capítulo 3

Motivación y objetivos

Conocer los motivos que impulsan a llevar a cabo una determinada acción facilita la empatía entre seres humanos, promueve el contagio de ilusiones y ayuda a comprender. El presente capítulo cuenta con dos secciones bien diferenciadas, explicando la motivación que llevó a la elección de este proyecto y los objetivos que se intentaban conseguir con el mismo.

3.1. Motivación

En el ámbito académico, la Web Semántica se contextualiza e integra en materias como la *Gestión Avanzada de la Información y el Conocimiento*, donde las Tecnologías de la Información se insertan con la Inteligencia Artificial con el objeto de modelar conocimiento humano para un posterior procesamiento y aprendizaje automático.

Las tecnologías de la Web Semántica se caracterizan por un cambio de paradigma bastante brusco con respecto a otras tecnologías de manipulación de datos con las que un alumno muy probablemente pueda estar familiarizado a la hora de introducirse en este nuevo mundo. Es cierto que pueden establecerse ciertas comparativas conceptuales para facilitar su comprensión; como ejemplo, el cuadro 3.1 equipara la Web Semántica con un sistema basado en bases de datos relacionales.

Mundo Relacional	Mundo Semántico
Registro de tabla	Nodo RDF
Columna de tabla	RDF propertyType
Celda de tabla	Valor
Consulta SQL	Consulta SPARQL
Modelo de datos	Ontología

Cuadro 3.1: Comparativa paradigma relacional vs semántico.

Sin embargo, determinados indicadores pueden dar lugar a entender que la interiorización de los conceptos asociados a la Web Semántica no es sencilla. Por ejemplo: su uso casi exclusivamente académico con poca penetración en el mundo empresarial, su lenta velocidad de propagación, su soporte limitado en determinadas plataformas tecnológicas o incluso el tímido interés mostrado en el ámbito público (excluyendo honrosas excepciones como el académico o universitario, las iniciativas de datos abiertos o lo relativo a la biblioteconomía). Un ejemplo muy concreto, real y tangible de esta escasa penetración es el reducido número de artículos (cuatro) presentado en *workshops* como *SemStats*[1][8], de cuyo *Program Comittee* este autor forma parte:

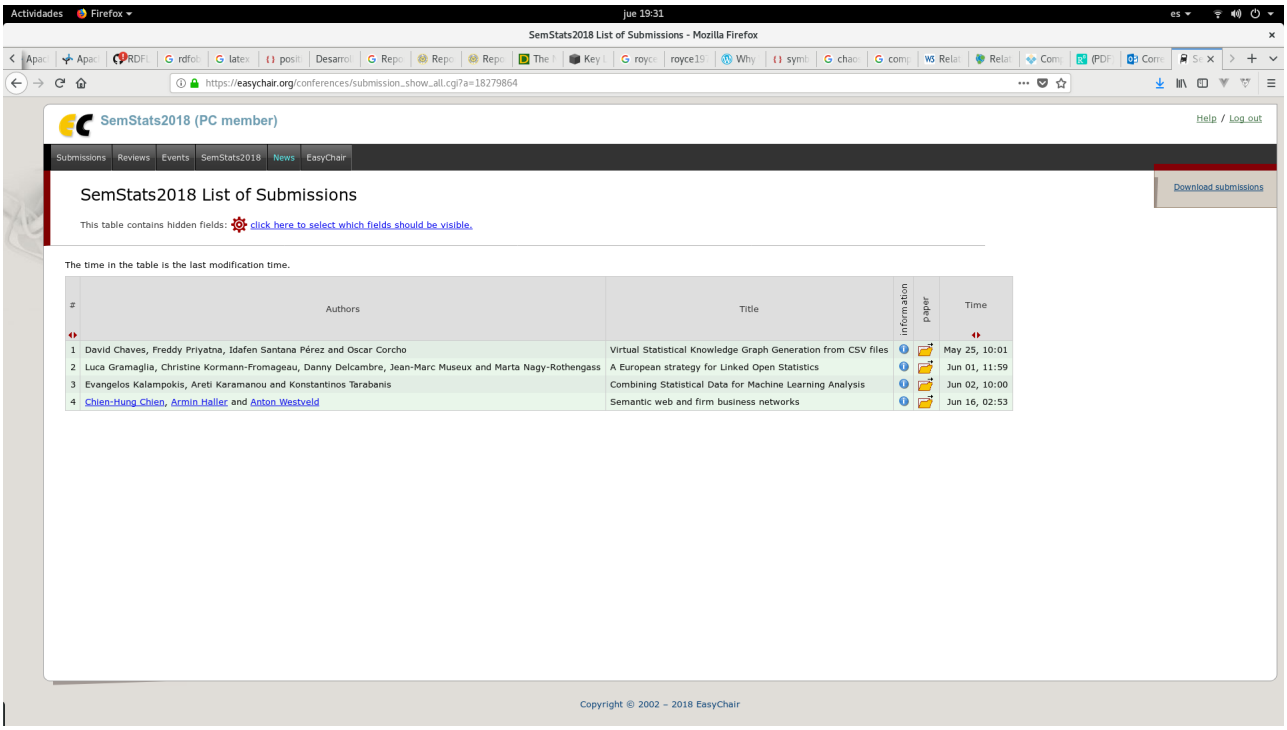


Figura 3.1: Trabajos enviados a *Semstats* 2018.

[1] *SemStats* es un *workshop* de celebración anual centrado en el estado actual de las tecnologías de la Web Semántica aplicadas a la estadística pública. Ver http://semstats.org/

En este caso, resulta cuando menos sorprendente que las Oficinas Estadísticas Públicas, encargadas de generar y publicar datos estadísticos sobre economía, población y sociedad en general, no estén mostrando mayor interés a la hora no ya de publicar sus datos en formatos enlazables, sino de construir aplicaciones semánticas para explotar y relacionar mejor estos datos.

Es esta **situación de dificultad en el aprendizaje y en la interiorización de la utilidad de las tecnologías de la Web Semántica la que motiva**, en última instancia, la idea de desarrollar herramientas que acerquen este tipo de tecnologías al público en general y faciliten una toma de contacto gradual con las mismas, ayudando así a su divulgación.

3.2. Objetivos

Si bien es cierto que existen, por una parte, varias aproximaciones gratuitas en la red a modo de *playground* que permiten el lanzamiento de consultas SPARQL a determinados *endpoints* y por otra, editores y herramientas de modelado del calibre de *Protégé*[2] (analizados con detalle en la sección 4.1), no parece haber en el mercado referencias de aplicaciones que ofrezcan simultáneamente subconjuntos reducidos de ambas funcionalidades. Llenar este hueco y **conseguir una herramienta formativa única de modelado y consulta en tecnologías de la Web Semántica** que permita a los equipos docentes de las asignaturas relacionadas con tecnologías de la Web Semántica ofrecer a los alumnos (o a cualquier no iniciado en estas tecnologías) un *playground* o entorno de pruebas que les ayude a consolidar su aprendizaje teórico con una base práctica adaptada al mundo real **es el objetivo principal de este proyecto**.

De forma más específica, es posible enumerar los siguientes objetivos detallados:

1. Facilitar el aprendizaje y la **familiarización con tecnologías de la Web Semántica** a usuarios inexpertos.
2. Reducir al máximo los requisitos de uso de la herramienta para alcanzar el mayor público objetivo posible.
3. Ofrecer a los equipos docentes la posibilidad de **distribuir actividades formativas** autocontenidas para que los alumnos trabajen sobre ellas en la herramienta.

[2]Un editor de ontologías y marco de trabajo para construir sistemas inteligentes. Ver https://protege.stanford.edu/

4. Agrupar en un solo producto **funcionalidades básicas de edición y consulta sobre grafos RDF**, de cara a cubrir las expectativas educativas en este ámbito.

5. Permitir a los alumnos comprobar *in situ* los efectos de sus acciones sobre un grafo RDF y ofrecerles la **flexibilidad suficiente** como para permitirles dar rienda suelta a su creatividad e inquietudes.

   Y por último, y no menos importante,

6. **Obtener un *Minimum Viable Project* (MVP)**[3] que pueda ser evolucionado y poner su código a disposición de la comunidad educativa para revertir sobre ella su valor aportado.

[3] Un producto con las suficientes características como para satisfacer a sus usuarios tempranos.

Capítulo 4

Trabajos previos, estado actual y recursos

Tras la introducción a los fundamentos teóricos, es conveniente resumir cuál es la situación actual (del inglés *state of the art*) de las distintas tecnologías relacionadas con este proyecto. A continuación se revisa un conjunto de herramientas existentes relacionadas en algún modo con los objetivos presentados, así como los estados de implementación de las tecnologías de la Web Semántica en plataformas como *Java*, *Python* o *Javascript*.

4.1. Trabajos previos

Para enfocar el presente proyecto se llevó a cabo un estudio previo de los trabajos existentes que pudieran estar relacionado con los objetivos del mismo. En las siguientes subsecciones se analizan las familias de herramientas analizadas.

4.1.1. Herramientas de consulta

La herramienta de consulta web que se asemeja más a los resultados obtenidos con este proyecto es el SPARQL *playground* desarrollado por el *Swiss Institute of Bioinformatics*[1].

Se trata de una aplicación web *standalone* y multiplataforma cuyo propósito fundamental es el aprendizaje de SPARQL. Sus autores proporcionan una demostración en línea[2] además de documentación y consultas de ejemplo para que cualquiera pueda familiarizarse con este lenguaje.

[1] https://www.isb-sib.ch
[2] http://sparql-playground.sib.swiss/

Si bien sus funcionalidades de consulta son muy completas, carece de facilidades de edición de grafos propios o de consulta a *endpoints*[3] externos, por ejemplo.

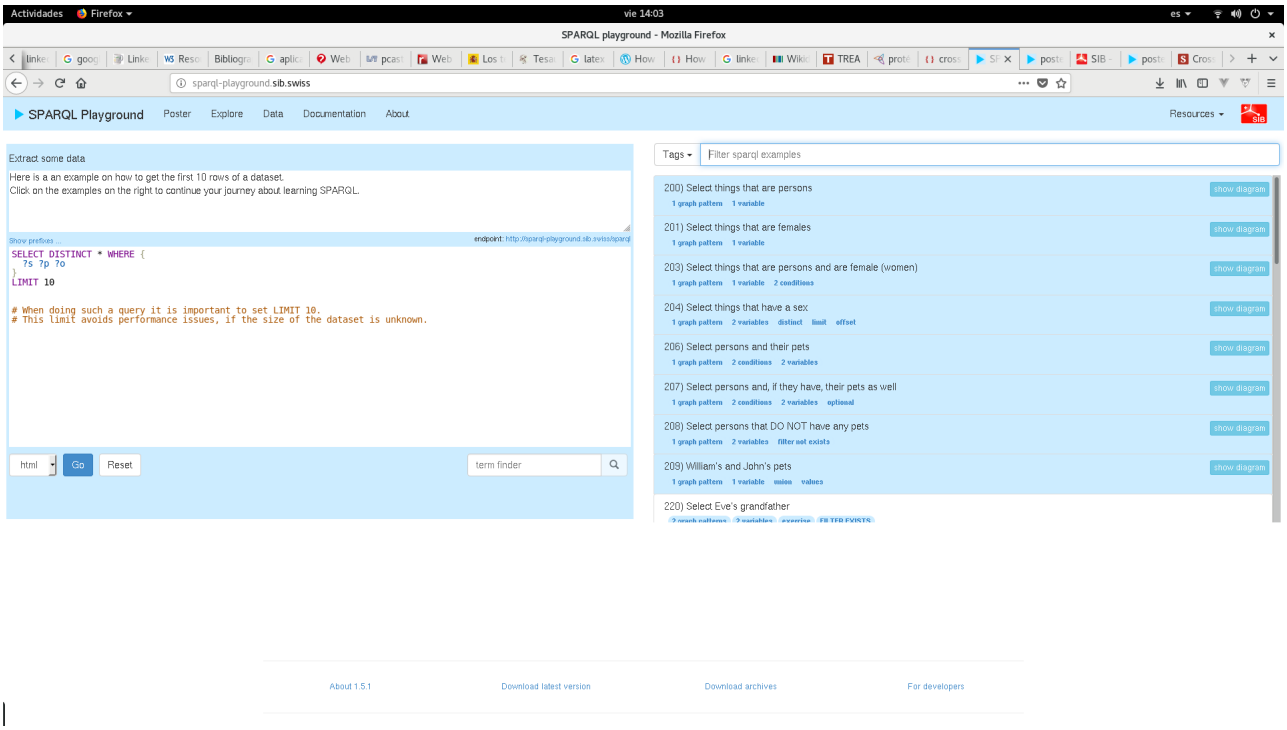


Figura 4.1: Interfaz de SPARQL *playground*.

También es posible encontrar bibliotecas de consulta para el *frontend*, como Jassa (*Javascript Suite for Sparql Access*)[4]. Se trata de una biblioteca modular que comprende desde una API RDF sobre una capa de abstracción de servicios hasta una capa de navegación por facetas. Sin embargo, desarrolla un modelo propio del grafo RDF no basado en estándares y se integra con versiones muy antiguas de *Angular*, lo que hace que pierda interés para su uso en producción.

4.1.2. Herramientas de modelado

4.1.2.1. Herramientas web

Existen pocas herramientas con funcionalidades de edición de grafos RDF en la Web. Al margen de la versión web de *Protégé*, cuya versión de escritorio (más representativa) se comenta en el punto 4.1.2.2, el resto de trabajos encontrados son fundamentalmente bibliotecas o paquetes de edición

[3]Puntos de acceso en la red, en este caso a través del protocolo SPARQL.
[4]http://aksw.org/Projects/Jassa.html

de formularios, como *rdfforms*[5]. Se trata de una biblioteca *Javascript* cuyo propósito es facilitar la construcción de editores RDF basados en formularios en un entorno web. Para ello, se basa en un mecanismo de plantillas que describe cómo generar un formulario HTML y cómo mapear expresiones específicas en un grafo RDF a sus correspondientes campos.

Unas pruebas iniciales con la biblioteca sugirieron falta de madurez por fallos en su funcionamiento en un navegador de uso común. Además, la biblioteca implementa su propio modelo de grafo RDF, que no está relacionado con los estándares actuales (ver sección 4.2).

Figura 4.2: Ejemplo de formulario desarrollado con *rdfforms*.

4.1.2.2. Herramientas de escritorio

La herramienta de modelado de ontologías en RDF por antonomasia es *Protégé*, un editor de ontologías *opensource*[6] gratuito y un marco de trabajo para construir sistemas inteligentes soportados por una fuerte comunidad de usuarios académicos, gubernamentales y corporativos. Se utiliza en áreas tan diversas como la biomedicina, el comercio electrónico, la predicción meteorológica o el modelado organizacional.

Protégé se instala como una aplicación de escritorio multiplataforma y tiene una arquitectura basada en *plugins* o complementos que permite extender fácilmente su funcionalidad. Tiene soporte para razonadores sobre las distintas ontologías que se modelan y facilidades de representación y edición de grafos RDF.

De cara a un primer encuentro con las tecnologías de la Web Semántica, puede parecer abrumador por su gran flexibilidad y potencia. Además, no cuenta con facilidades de consulta sobre los grafos modelados.

[5] https://rdforms.org/
[6] Software de fuentes abiertas, ver https://opensource.org/faq#osd

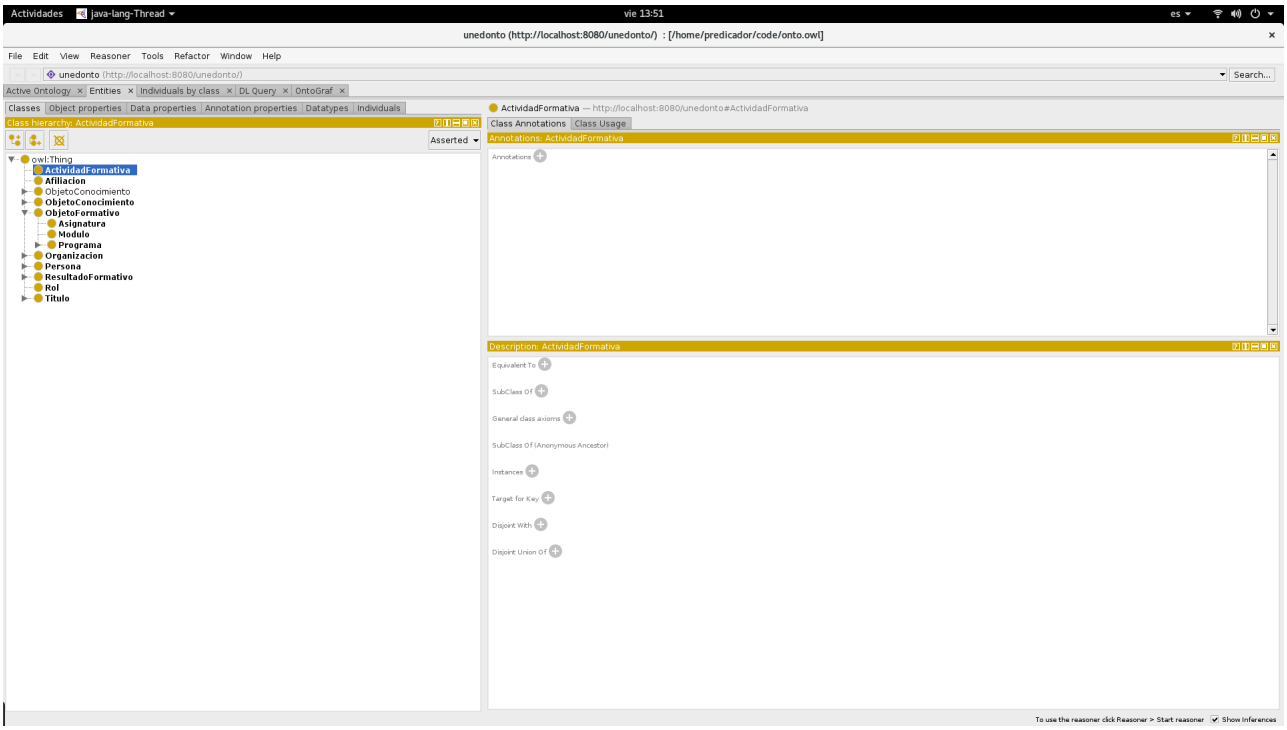


Figura 4.3: Interfaz de *Protégé*.

4.2. Estado actual

El estado actual de las tecnologías de la Web Semántica alcanza distintos niveles de madurez en sus implementaciones. Complementando el listado de herramientas ya presentado en el estudio de trabajos previos (4.1), se analizan a continuación las plataformas o bibliotecas más representativas dentro de las principales tecnologías.

4.2.1. RDF en Java

Así, para la plataforma *Java* se cuenta con *Apache Jena*[9], un marco de trabajo o *framework opensource* para construir aplicaciones para la Web Semántica o sobre datos enlazados. De entre sus componentes y funcionalidades, cabe destacar:

- **TDB**: una base de datos de tripletas nativa y de alto rendimiento, con excelente integración con el resto de APIs de Jena.
- **ARQ**: un motor SPARQL compatible con su versión 1.1 que soporta consultas federadas y búsqueda por texto libre.

- **API RDF**: una API principal que permite interactuar con RDF para crear y leer grafos y tripletas, así como serializarlas a los formatos más comunes (*Turtle, XML*, etc.).
- **Fuseki**: un *endpoint* SPARQL que permite exponer el grafo de trabajo y ofrece interacción tipo Representational State Transfer *(REST) con tripletas RDF.*
- ***Otras APIs***: como las de ontologías e inferencias, que permiten añadir más semántica al modelo y razonar sobre reglas por defecto o personalizadas.

Apache Jena es una solución robusta y muy utilizada en entornos académicos y de producción empresarial con más de quince años de vida.

4.2.2. RDF en Python

RDFLib[7] es una biblioteca *opensource* ligera pero funcionalmente completa para trabajar con RDF desde plataformas *Python*. Permite a las aplicaciones acceder a estructuras RDF a través de construcciones idiomáticas en *Python*, lo que facilita un acercamiento de la tecnología al programador experimentado; por ejemplo, un grafo no es más que una colección de tripletas *<sujeto, predicado, objeto>*.

Entre el resto de sus características, destacan:

- Contiene procesadores y serializadores para *XML, N3, Turtle, RDFa*, etc.
- Presenta una interfaz para un grafo que puede soportarse sobre multitud de implementaciones de almacenes.
- Incluye una implementación de SPARQL v1.1.
- Presenta una arquitectura modular basada en *plugins* o complementos.

Su desarrollo es estable y ha sido utilizada en referencias bibliográficas como *Programming the Semantic Web*[10], de *O'Reilly*.

[7] https://github.com/RDFLib/rdflib

4.2.3. RDF en Javascript

Dado que uno de los objetivos del proyecto es reducir al máximo los requisitos de uso de la herramienta para alcanzar el mayor público objetivo posible (ver 3.2), ***Javascript* se presenta como una tecnología de sumo interés para su implementación**, ya que permite producir aplicaciones ejecutables al cien por cien en un navegador web estándar.

Sin embargo, el **estado actual de implementación de RDF en Javascript puede calificarse de inmaduro y e inestable**. Se han llevado a cabo intentos aislados de construir APIs RDF y herramientas para gestión de grafos y consultas SPARQL como las citadas en la sección 4.1, pero aún no existe un estándar único consolidado.

4.2.3.1. Manejo de RDF

En el W3C puede consultarse la mejor comparativa existente de bibliotecas desarrolladas en *Javascript* para trabajar con RDF[11].

Sin duda, el esfuerzo más destacable hasta ahora para conseguir un estándar de RDF en Javascript es el proyecto ***rdflib.js***[8], iniciado por el mismísimo Tim Berners-Lee. Se trata de una biblioteca que proporciona una API local para lanzar consultas a un almacén RDF, soporta parte de la especificación SPARQL y permite importar y escribir formatos como RDF/XML, *Turtle* y N3.

En el propio repositorio de *rdflib.js* se cita al *RDFJS Representation Task Force*[9], un grupo dentro del *RDFJS Community Group*[10] encargado de diseñar las especificaciones de interfaz con el objetivo de conseguir que las distintas implementaciones de conceptos RDF en *Javascript* sean interoperables. La especificación de la interfaz existe en forma de borrador a fecha de 14 de Agosto de 2017[11]. En ella se definen interfaces para términos, cuaternas, factorías de datos, nodos en blanco, literales, etc. así como para *streams*[12] de *Javascript*.

Consultando el sitio web del *Community Group* se puede encontrar una noticia del 23 de Abril de 2018 (publicada, por tanto, en pleno desarrollo de este proyecto) anunciando la creación de

[8] https://github.com/linkeddata/rdflib.js
[9] https://github.com/rdfjs/representation-task-force
[10] https://www.w3.org/community/rdfjs/
[11] http://rdf.js.org/
[12] API para acceder a flujos de datos asíncronos en JS. Ver https://developer.mozilla.org/en-US/docs/Web/API/Streams_API

RDF.js, que tratará de unir los esfuerzos de distintas entidades (como la Universidad de Gante o el *Massachusetts Institute of Technology* (MIT)) y alinear y publicar bajo un único paraguas proyectos como:

- El citado *rdflib.js*.
- ***RDF-Ext***[13], una biblioteca *Javascript* para trabajar con RDF y datos enlazados que cumple con la especificación del *RDFJS Representation Task Force*.
- ***N3.js***[14], otra biblioteca *Javascript* que implementa la especificación de bajo nivel RDF.js y permite leer, procesar, almacenar y exportar ternas RDF de forma asíncrona en *Javascript*.

Ante esta situación confusa y repleta de distintas implementaciones, se optó por utilizar canales de comunicación directos con sus autores. Existen varios grupos de discusión en *Gitter*[15], tales como:

- *rdfjs/public*[16]
- *rdf-ext/discussions*[17]

En ellos, el autor de este proyecto tuvo la oportunidad de recibir indicaciones directamente de Thomas Bergwinkl y Michael Luggen, importantes miembros del grupo, quienes confirmaron que *RDF-Ext* es el camino a seguir en caso de tener que desarrollar un proyecto en *Javascript* que trabaje con RDF. Luggen también indicó que *rdflib.js* es la biblioteca más antigua de las citadas, iniciada por el propio Tim Berners-Lee (que, como se puede comprobar en la figura 4.4, aparece también en el grupo de conversación.)

4.2.3.2. Manejo de SPARQL

Como se ha visto en la subsección anterior, la situación actual de la implementación de RDF en *Javascript* es compleja. Con SPARQL, esta línea se mantiene.

[13] https://github.com/rdf-ext/rdf-ext
[14] https://github.com/rdfjs/N3.js
[15] Una plataforma de *chat* en tiempo real para programadores.
[16] https://gitter.im/rdfjs/public
[17] https://gitter.im/rdf-ext/discussions

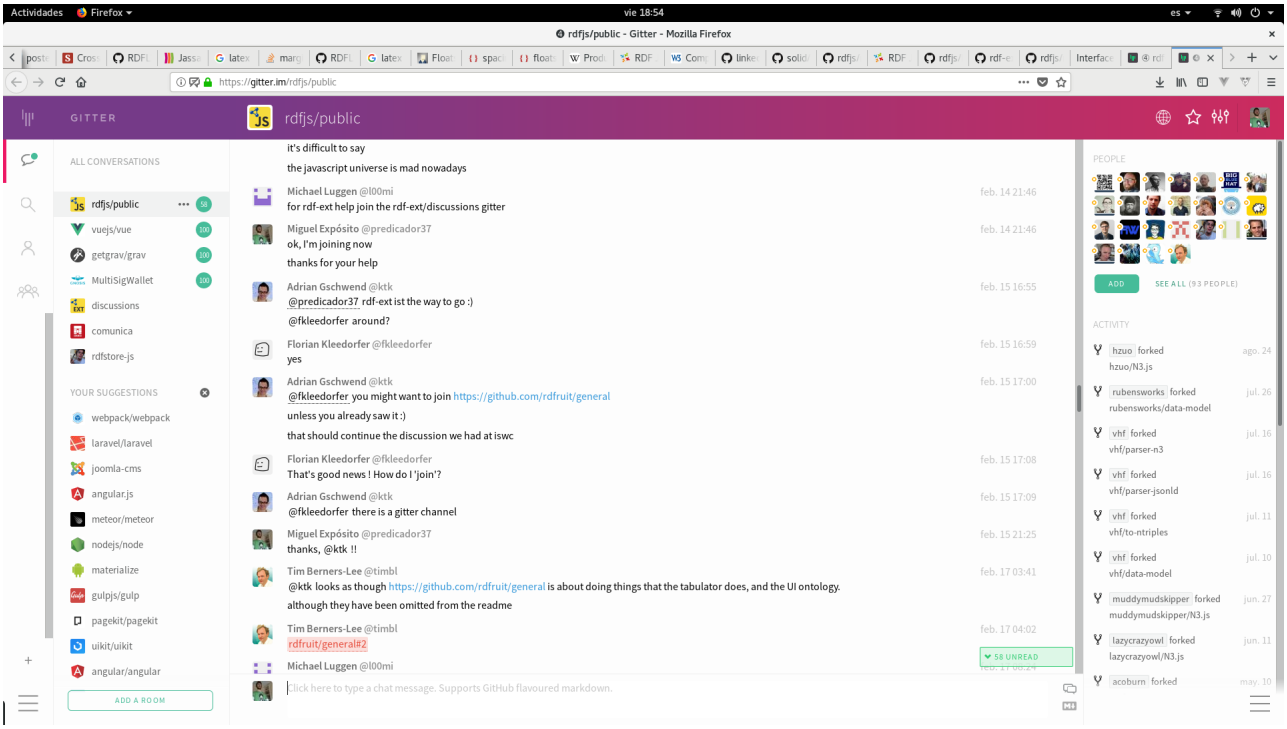


Figura 4.4: Charla con miembros del *W3C RDFJS Community Group* (¡y @timbl!).

El primer trabajo desarrollado en esta línea es **rdfstore-js**, de Antonio Garrote[18], que consiste en una implementación pura en *Javascript* de un almacén de grafos RDF con soporte para el lenguaje de consulta y manipulación de datos SPARQL.

Lamentablemente, tal y como indica el propio autor, la implementación actual está lejos de estar completa. Por otra parte, revisando el repositorio, se puede comprobar que la última actualización del código fuente es de hace dos años con respecto a la fecha de redacción de la presente memoria. Existe un grupo de *Gitter* sobre el proyecto, pero carente de actividad. Por tanto, no es descabellado asegurar que la biblioteca está obsoleta y ha sido discontinuada, lo que hace inviable su inclusión en el proyecto.

Por otro lado, Ruben Taelman, miembro del grupo de discusión en *Gitter* sobre *RDF-Ext*, lidera otro proyecto, *Comunica*[19]: una plataforma de motores de consulta en *Javascript* altamente modular y flexible para la web. Esta plataforma permite enviar consultas a distintos motores, entre ellos SPARQL. Entre sus características se encuentra la capacidad para lanzar consultas SPARQL 1.1 a *endpoints* externos y también a un almacén local en *Javascript*, lo que lo hace ideal para un proyecto como el que se trata en esta memoria.

[18] https://github.com/antoniogarrote/rdfstore-js
[19] https://github.com/comunica/comunica

No obstante, es un producto en pleno desarrollo en el que muchos de sus módulos aún no han sido publicados en una versión estable y, por tanto, **inestable y con defectos potenciales** (si bien sus autores tienen previsto incorporar gran parte de la funcionalidad estándar esperada para finales de año).

4.2.4. Tecnologías de frontend web

La interfaz de usuario será un aspecto muy importante del proyecto, dado que pretende facilitar el aprendizaje de RDF y SPARQL a alumnos sin experiencia. Ello, unido a la necesidad de facilitar lo máximo posible la puesta en marcha de la herramienta, hace que las tecnologías de *frontend* web cobren un papel especialmente relevante en este análisis.

El mundo del *frontend* web ha cambiado mucho en los últimos años, y de hecho, la comunidad de desarrolladores aún se encuentra inmersa en una vorágine de cambios cuya velocidad es, en la mayor parte de los casos, infinitamente mayor que la capacidad de los proyectos y los propios desarrolladores para asimilarla.

Atrás quedaron los días de servir interfaces de usuario en el servidor a través de *Java Server Pages* (JSP)[20] o *Java Server Faces* (JSF)[21] y de esas tímidas incursiones de *Javascript* en la capa de presentación a través de bibliotecas como *JQuery*[22] o *Ext Js*[23].

En estos momentos, son tres los marcos de trabajo que aglutinan la mayor parte del desarrollo de *frontend* web: ***Angular***, ***React*** y **Vue**. En las siguientes subsecciones se analizan someramente.

4.2.4.1. *Angular*

Angular es un marco de trabajo basado en *Typescript* y creado por *Google* en 2009, pensado para la construcción de aplicaciones web altamente interactivas. Entre sus características se encuentran las siguientes:

[20] Tecnología para crear páginas web dinámicas en servidor en Java
[21] Tecnología que simplifica el desarrollo de interfaces de usuario en aplicaciones *Java EE*
[22] https://jquery.com/
[23] https://www.sencha.com/products/extjs/#overview

- Orientado a componentes[24].
- Uso de *Typescript*.
- Sistema de plantillas en HTML con directivas.
- Basado en el patrón *Model View Controller* (MVC)[25].
- Incluye inyección de dependencias[26].
- Varias opciones para la gestión de estado.
- Curva de aprendizaje pronunciada.
- Tamaño pesado.
- Publicado bajo licencia MIT[27].

4.2.4.2. *React*

React es una biblioteca *Javascript* para construir interfaces de usuario desarrollada y mantenida por *Facebook* desde 2013.

- Orientado a componentes.
- Uso de *Javascript* ES6.
- Sistema de plantillas en *Javascript Syntax eXtension* (JSX)[28].
- Basado en el patrón MVC.
- Virtual *Document Object Model* (DOM): permite organizar documentos HTML en un árbol más sencillo de procesar por un navegador web.
- Gestión de estado con *Redux*.
- Curva de aprendizaje pronunciada.

[24] Un componente es una unidad de encapsulado de código y responsabilidades que facilita la reutilización
[25] Ver https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture
[26] Patrón en el cual un objeto (servicio) proporciona las dependencias necesarias a otro objeto (cliente)
[27] https://opensource.org/licenses/MIT
[28] Una extensión de sintaxis para *Javascript*. Ver https://reactjs.org/docs/introducing-jsx.html

- Tamaño medio.
- Publicado bajo licencia MIT.

4.2.4.3. *Vue*

Vue es uno de los marcos de trabajo que más rápidamente está creciendo en popularidad y uso desde 2014. Su primera versión fue publicada por un antiguo trabajador de *Google*: Evan You.

- Orientado a componentes.
- Uso de *Javascript* ES5/ES6.
- Sistema de plantillas en HTML con directivas.
- *Single File Components*, con separación de responsabilidades.
- Basado en el patrón *Model View View-Model* (MVVM)[29].
- Virtual DOM.
- Gestión de estado con *Vuex*.
- Curva de aprendizaje suave.
- Tamaño ligero.
- Publicado bajo licencia MIT.

4.2.4.4. Comparativa de marcos de trabajo

Tras un análisis de las características de los tres marcos de trabajo, se puede comprobar que técnicamente no hay grandes diferencias en cuanto a funcionalidad provista o patrones empleados, pero sí en cuanto a ciertos aspectos del diseño y facilidad de uso. *Angular* es pesado y complejo, *React* es el más demandado laboralmente (ver 4.6) pero su curva de aprendizaje es pronunciada y *Vue* es el más ligero y sencillo de usar y aprender, siendo recomendado para proyectos con equipos pequeños o unipersonales.

[29]Facilita la separación del desarrollo de la interfaz de usuario de la lógica de negocio o *backend*

En cuanto a popularidad según el número de estrellas en *Github* (cuadro 4.1), se puede concluir que *React* y *Vue* son los más populares, quedando *Angular* a una distancia considerable.

Por otra parte, a la vista una encuesta de *stateofjs.com* en 2017 (ver figura 4.5) sobre una muestra de más de 28.000 desarrolladores de todo el mundo, si bien *React* es el jugador dominante, *Vue* está haciendo grandes progresos en detrimento de *Angular*, lo que podría suponer un desafío para el liderazgo de *React* en 2018 y años posteriores.

Framework	Nº de estrellas en *Github*
Angular	40 133
React	110 129
Vue	112 650

Cuadro 4.1: Estrellas en *Github* de los principales marcos de trabajo *Javascript* (09/2018).

Figura 4.5: Encuesta sobre el estado de Javascript en 2017 por stateofjs.com[12].

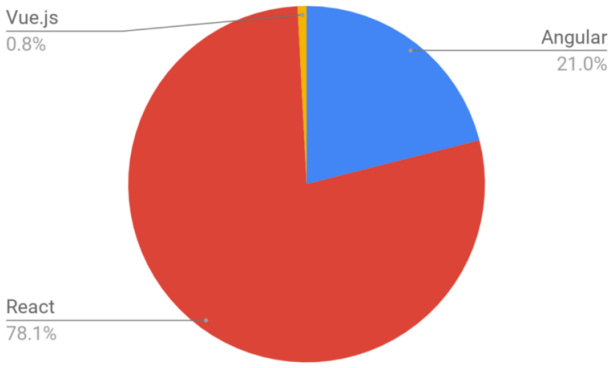


Figura 4.6: Ofertas de trabajo por *framework* JS según Medium.com a través de Indeed.com[13].

Cabe destacar en esta memoria que ***Vue* fue propuesto como alternativa para el desarrollo del proyecto por los miembros del *W3C RDFJS Community Group***, como puede comprobarse en la figura 4.7.

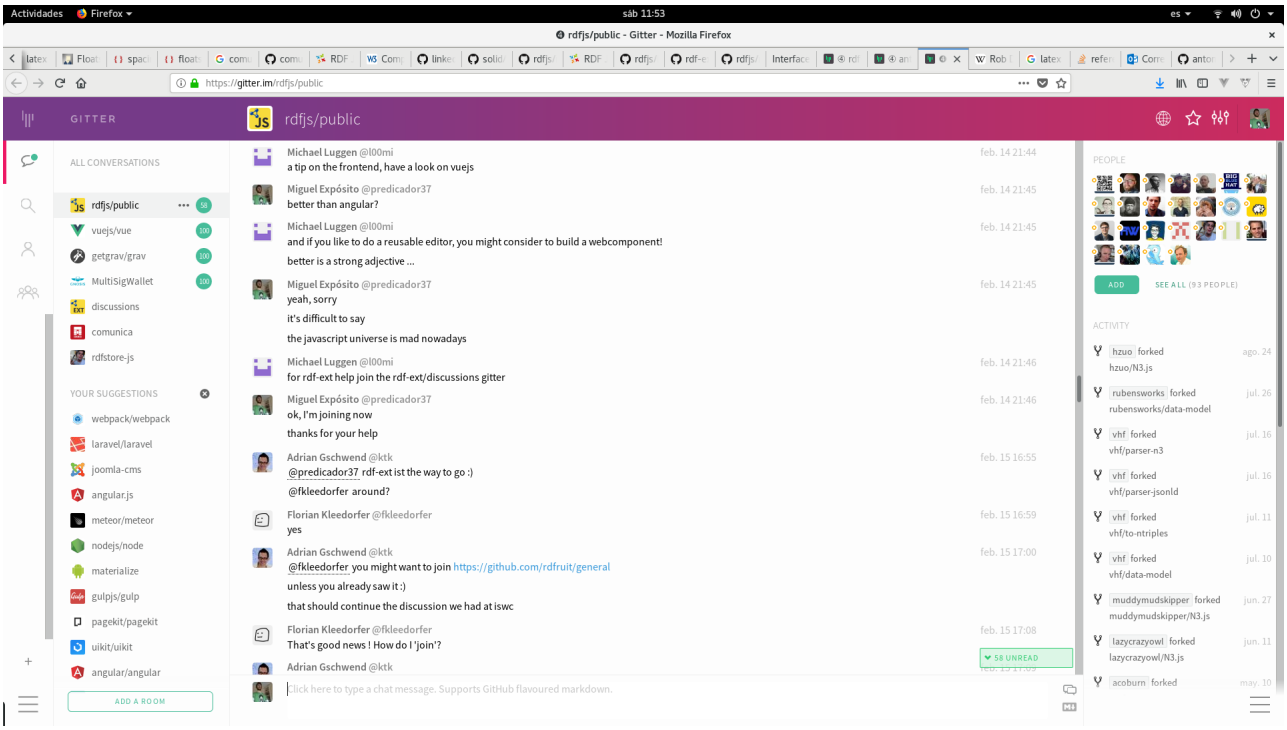


Figura 4.7: Recomendación de *Vue* en la charla con miembros del *W3C RDFJS Community Group*.

4.3. Recursos

Una vez analizados los trabajos previos existentes en 4.1 y la situación actual en 4.2, se discute en esta sección qué recursos que serán necesarios para el desarrollo del proyecto.

- En primer lugar, será necesario utilizar un sistema de control de versiones para el código desarrollado. Actualmente, el más utilizado es *git*, y la plataforma con mayor número de desarrolladores, *Github* (28 millones, según la propia *Github*[30]). Una cuenta gratuita bastará para alojar el código fuente y la documentación del proyecto, si bien *Github* ofrece cuentas privadas con duración limitada para estudiantes universitarios.
- Para llevar a cabo la planificación y evolución diaria de las tareas, se utilizará un plan gratuito de *Trello* (lo cual se detalla en profundidad en la subsección 5.2.2)
- A la vista de los objetivos del proyecto, *Javascript* ES6 parece la tecnología más adecuada para su desarrollo. Dada la inexperiencia del autor en esta tecnología, se proponen las siguientes referencias bibliográficas y de formación *online*:
  - *Get Programming With Javascript Next*, JD Isaacks[14].
  - Curso en la plataforma Udemy[31]: *ES6 Javascript: The Complete Developer's Guide*, Stephen Grider[15].
  - Curso en la plataforma *Udemy*: *Working with Javascript Streams*[16].
- Analizadas las distintas implementaciones de RDF y SPARQL en *Javascript* y, siguiendo las recomendaciones del *W3C RDFJS Community Group*, se propone el uso inicial de los siguientes recursos:
  - *RDF-Ext* como biblioteca para trabajar con la API de RDF, siguiendo los estándares existentes.
  - *Comunica* para las consultas SPARQL, dado que aún con los riesgos inherentes que conlleva la utilización de un producto aún en desarrollo, es la única plataforma que puede responder a las necesidades del proyecto.
- Tras el estudio de la situación actual de los marcos de trabajo de *frontend Javascript* y, teniendo en cuenta asimismo las recomendaciones del *W3C RDFJS Community Group*, se opta por *Vue*

[30] https://github.com/search?q=type:user&type=Users
[31] https://www.udemy.com

para el proyecto dada su gran proyección y su curva de aprendizaje suave. En concreto, se propone utilizar los siguientes recursos:

  - Curso en la plataforma *Udemy*: *Vue JS 2 - The Complete Guide*, Maximilian Schwarzmüller[17], un referente en la comunidad de desarrolladores de *Vue*.
  - Curso en la plataforma Udemy: *The Ultimate Vue JS 2 Developers Course*, Anthony Gore[18]. Otro referente importante.
  - *Vue.js Design Patterns and Best Practices*, Paul Halliday[19].

- Como entorno de desarrollo, el autor de este proyecto ya contaba con una licencia de estudiante de *WebStorm* de *Jetbrains*[32], uno de los *Integrated Development Environment* (IDE)s más completos y modernos para *Javascript* existentes en la actualidad. Soporta *Javascript* de forma nativa, depuración con el navegador y marcos de trabajo como *Vue*, *React* o *Angular*, además de facilidades para pruebas unitarias, análisis estático de código, análisis de rendimiento, etc.

- Como plataforma colaborativa para comunicación con el Director del proyecto, se optó por *Skype*[33] en su versión gratuita.

[32] https://www.jetbrains.com/webstorm/
[33] https://www.skype.com/es/

Capítulo 5

Metodología y Planificación

La metodología y la planificación son obligadas en cualquier proyecto de ingeniería. No obstante, cabe aquí citar la siguiente frase atribuida a Dwight D. Eissenhower: *«Plans are worthless, but planning is everything.» ([20])* Hay que tener en cuenta que una planificación es tan sólo una guía inicial para conseguir una organización más depurada durante el tiempo de desarrollo. La planificación debe ser una herramienta flexible de ayuda, y no un instrumento rígido que condicione otras características del sistema tales como su calidad. Las metodologías ágiles ofrecen mecanismos para conseguir flexibilizar el proceso de planificación. En las siguientes secciones se analizan con más detalle estos asuntos.

5.1. Metodología

Para acometer este proyecto, se han valorado dos familias de metodologías de desarrollo de software:

1. Metodologías en cascada (*Waterfall*)
2. Metodologías ágiles (incrementales e iterativas)

La metodología de desarrollo en cascada surgió como idea en un artículo de Winston W. Royce en 1970[21]. Históricamente, este modelo se ha extendido tanto en ámbitos académicos como profesionales, siendo estas sus principales características:

- Gestión predictiva de proyectos llevada al software.

- Toma como modelo la forma de proceder en el resto de ingenierías.
- Intenta llenar el vacío del *code & fix*.
- Cada fase se realiza, en principio, una única vez.
- Cada fase produce un entregable que será entrada de la siguiente.
- Los entregables no son, en principio, modificables.

Es decir, se basa en la separación entre diseño y construcción (o entre creatividad y repetición).

La propuesta de Royce, tal y como se desprende de la lectura del artículo original, describía el modelo en cascada como la *«descripción más simple» ([21])* que solo funcionaría para los proyectos más sencillos. Irónicamente, este mensaje malentendido ha sido el origen de la popularidad de dicha metodología, que hoy en día se sigue promoviendo en muchos casos por inercia, desconocimiento, comodidad o ilusión de control sobre el proyecto.

Lamentablemente, en la Ingeniería de Software los pesos de diseño y construcción están invertidos con respecto a otras ingenierías, siendo el software un dominio de cambio y alta inestabilidad. El desarrollo de software es, intrínsecamente, una labor creativa; y la creatividad no es fácilmente predecible. Esto ha dado lugar a que los desarrollos tradicionales adolezcan de ciertos problemas[22]:

- Existencia de muchos requisitos vagos o especulativos y diseño detallado por adelantado.
- Están fuertemente asociados con las tasas de fallo más altas en proyectos.
- Se encuentran promovidos históricamente por creencia más que por evidencia estadística significativa.
- Su rigidez incrementa el riesgo de fracaso, pospuesto hasta las fases finales del proyecto.
- Asume que las especificaciones son predecibles, estables y completas.
- Pospone integración y pruebas hasta fases tardías.
- Se basa en estimaciones y planificación “fiables”.

Entre los estudios que ratifican las afirmaciones anteriores, cabe citar, entre otros, los siguientes:

- *Informe Chaos 2015*[23].
- *Dr. Dobb's Journal article The Non'Existent Software Crisis: Debunking the Chaos Report*[24].
- Encuesta de *Gartner* (2012)[**?**].

Por tanto, atendiendo a esta exposición y considerando que el proyecto en cuestión presentaba un alto nivel de incertidumbre debido a un importante componente en investigación del estado tecnológico actual, se ha optado por utilizar un enfoque metodológico incremental e iterativo, puesto que:

- Facilita llevar a cabo proyectos pequeños.
- Fomenta la interacción entre el desarrollador y el usuario.
- Fuerza a que los inevitables cambios en requisitos sucedan en fases tempranas del proyecto.

Entre las características de este enfoque incremental e iterativo, es posible citar:

- Se trabaja sobre subconjuntos de funcionalidad (*features*).
- Los incrementos permiten añadir funcionalidad al producto (mejora del proceso).
- Las iteraciones permiten rediseñar, revisar y refactorizar el producto (mejora del producto).
- Se basa en entregas frecuentes y ciclos prueba/error.
- Ofrece flexibilidad a la hora de gestionar el cambio.

La referencia más clara en este ámbito es *Extreme Programming*, de Kent Beck[25]. *Extreme Programming* es *«un estilo de desarrollo de software centrado en la aplicación excelente de técnicas de programación, comunicación clara y trabajo en equipo que permite conseguir objetivos antes impensables» ([25])*. Se trata de una metodología basada en valores como la comunicación, realimentación, simplicidad, valentía y respeto, soportada sobre un cuerpo de prácticas útiles y con un conjunto de principios complementarios, además de contar con una comunidad de usuarios que comparte todo lo anterior.

Su aplicación al desarrollo de este proyecto ha sido flexible; por ejemplo, no se han definido ciclos estrictos por la propia naturaleza inestable de la dedicación al desarrollo del mismo, pero sí que ha realizado un diseño evolutivo soportado sobre un número suficiente de pruebas unitarias así como una planificación incremental y adaptativa a las problemáticas que iban surgiendo.

Prueba del acierto a la hora de elegir esta metodología es el cambio de necesidades y requisitos funcionales que tuvo lugar en la reunión presencial mantenida con el tutor, donde **la metodología aportó que no fuera necesario descartar ningún desarrollo o trabajo realizado hasta la fecha**. Permitió realizar una gestión del cambio efectiva, reorientando el trabajo a tiempo sin impactar en el diseño ni en la implementación existente. Finalmente, tanto la organización como la planificación de las tareas han sido lo suficientemente flexibles para conseguir un ritmo adecuado de desarrollo, reduciendo los puntos de bloqueo.

5.2. Planificación

5.2.1. Planificación global

Para realizar la planificación del proyecto, y dada la metodología de desarrollo incremental e iterativa elegida, no se ha seguido un modelo típico en fases de *Análisis, Diseño, Implementación, etc.* sino que se ha optado por un enfoque orientado a tareas relacionadas con la funcionalidad del producto final esperado.

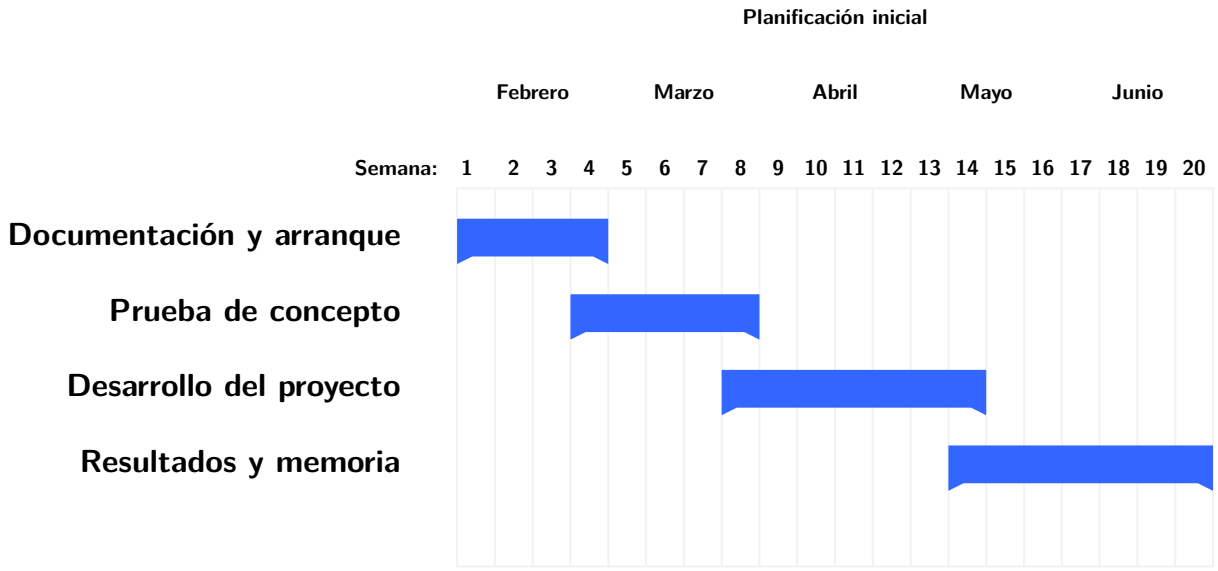


Figura 5.1: Planificación inicial.

La figura anterior (5.1) muestra la planificación propuesta para la elaboración del anteproyecto.

En esta propuesta era clave la revisión de la prueba de concepto con el tutor para comprobar que efectivamente se habían comprendido las necesidades desprendidas del análisis y para poder continuar con la especificación funcional de forma más refinada y precisa.

En la práctica, la evolución del proyecto ha sido bien distinta. En la figura 5.2 se puede comprobar cuál ha sido la planificación efectiva, producto esta de cambios sobre la inicial para resolver los distintos imprevistos que han ido surgiendo durante la ejecución del mismo.

Con respecto a la planificación inicial, se tiene que:

- El proyecto se ha retrasado un total de ocho semanas.
- El período de formación llevó al menos dos semanas más de lo esperado (en realidad, el aprendizaje del marco de trabajo *Vue* ha estado presente a lo largo de prácticamente todo el desarrollo del proyecto).
- El desarrollo del producto ha consumido seis semanas más de las previstas inicialmente.
- La memoria se ha redactado en dos semanas menos con respecto a la estimación inicial.

Las desviaciones surgidas con respecto a la planificación inicial tienen su explicación en los

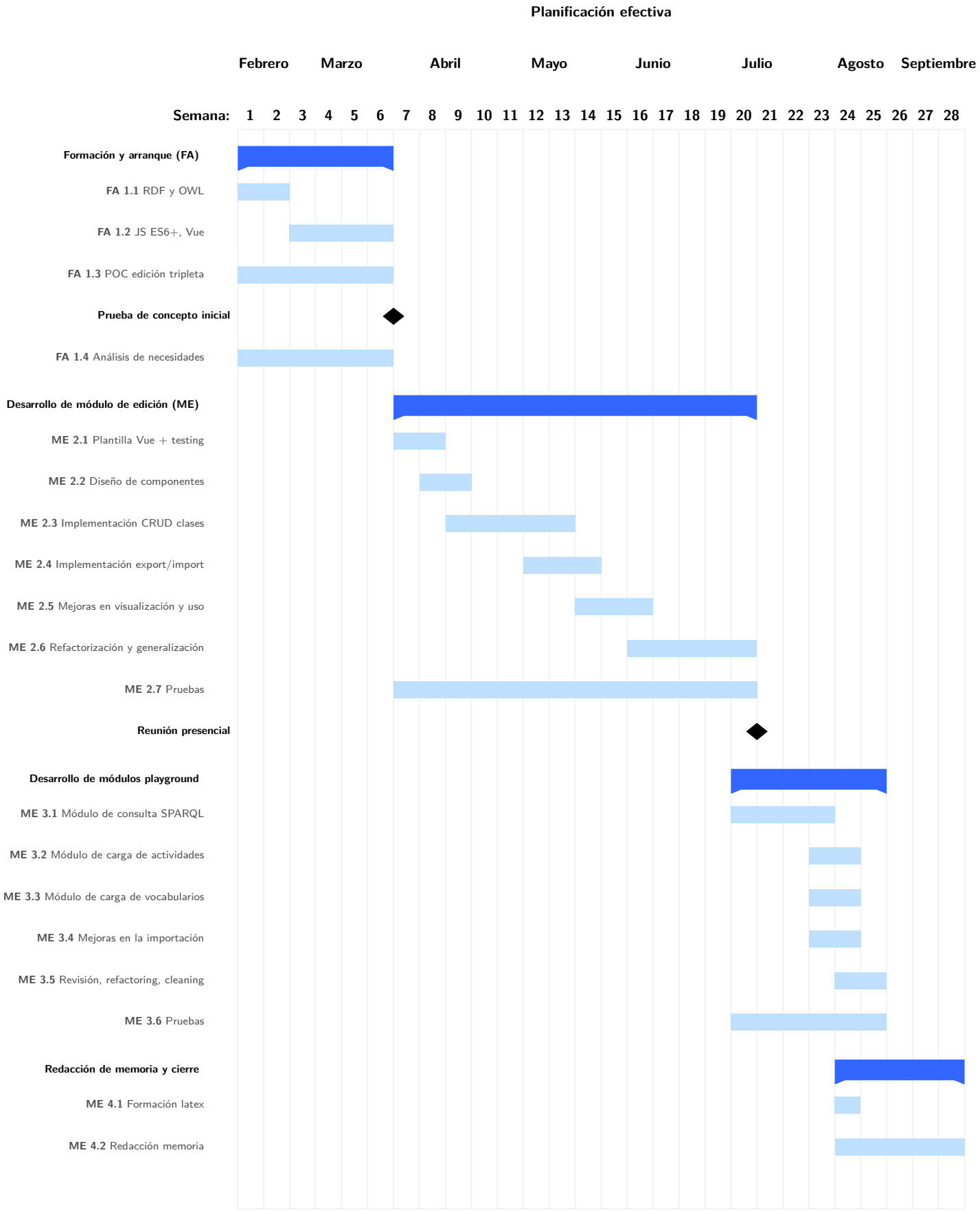


Figura 5.2: Planificación efectiva.

siguientes motivos:

- El alto grado de incertidumbre a la hora de estimar la planificación inicial, dado que se desconocía cuál era la situación actual del ecosistema tecnológico de *frontend* web y su integración con las tecnologías de la Web Semántica.
- La curva de aprendizaje de *Vue*, si bien es considerada más suave que la de sus competidores (*React, Angular*) fue mayor de lo esperado. El poco o ningún grado de familiarización del autor del proyecto con las tecnologías de *frontend* y, especialmente, *Javascript ES6+*, no facilitó precisamente el aprendizaje.
- El **bajo nivel de madurez de las bibliotecas existentes** para la manipulación de RDF en *Javascript*, así como la heterogeneidad y poca estabilidad de estas implementaciones, ha suscitado muchas dudas sobre cuáles utilizar y cómo enfocar su integración con marcos de trabajo más maduros como *Vue*.
- La **práctica ausencia de bibliotecas** o componentes de interacción con SPARQL **conformes a la especificación estándar de la interfaz** *rdf.js*[26] (que, por otra parte, es un borrador de 2017) ha puesto en peligro la viabilidad del proyecto. La plataforma finalmente utilizada, *Comunica*[27], aún no tiene una versión estable en muchos de sus módulos (concretamente, el *endpoint* SPARQL utilizado para el proyecto se encuentra en esta situación), con lo que ha sido necesario estar en contacto directo con sus autores y colaborar con ellos en la revisión de defectos o *bugs* y dependiendo, por tanto, de sus tiempos de respuesta (hay que tener en cuenta que la plataforma es *opensource* y no existe acuerdo de nivel de servicio alguno.)
- La dificultad encontrada en llevar a cabo un análisis de requisitos a través de una plataforma de colaboración en línea como puede ser *Skype*: para constatar este hecho no hay más que verificar el cambio de rumbo del proyecto una vez mantenida la reunión presencial, que sirvió para definir objetivos más claros y comprender mejor las necesidades del Departamento.

A pesar de todo ello, el autor de este proyecto está satisfecho con el nivel de conocimiento adquirido en el ámbito de todas las tecnologías empleadas y el tiempo consumido para obtener como resultado un producto desplegado en producción y listo para utilizar.

5.2.2. Planificación ágil

Si bien para la planificación global del proyecto se han utilizado herramientas tales como el *diagrama de Gantt*, para gestionar el trabajo del día a día ha sido necesario emplear otro enfoque.

En el contexto de metodologías ágiles de desarrollo de software de tipo incremental e iterativo como *Extreme Programming*, **las tareas de planificación adquieren un carácter adaptativo** muy distinto del que presenta una planificación tradicional con una metodología de desarrollo en cascada, por ejemplo.

La planificación ágil es un proceso en continua evolución, también iterativo e incremental como las metodologías a las que pertenece, basada en ciclos del tipo:

1. Añadir tareas.
2. Estimar tareas.
3. Priorizar tareas.

En este caso, la estimación de tareas se llevó a cabo de forma heurística, utilizando la experiencia del autor y el conocimiento del contexto existente en el momento de la estimación. En base a ello, la priorización (ordenación) de las tareas tenía lugar inmediatamente, relegando a la estimación a un segundo plano.

Para gestionar estas tareas, se optó por utilizar como herramienta *Kanban*[1]. Un tablero *Kanban* puede definirse como un dispositivo de señalización que introduce el flujo de trabajo de un proceso a un ritmo manejable. Presenta las siguientes características[28]:

- Solo envía trabajo cuando lo ordena el cliente o usuario (en este caso, el propio autor del proyecto).
- Indica específicamente qué trabajo debe hacerse.
- Controla la cantidad de trabajo en progreso.
- Regula las interrupciones y orquesta el ritmo de trabajo.

Básicamente, consiste en utilizar una tabla con varias columnas para visualizar el estado de una tarea a lo largo de las distintas fases que se consideren. Para el caso de la realización de este proyecto, se crearon las siguientes columnas:

[1] https://es.atlassian.com/agile/kanban

Columna	Descripción
To-Do	Tareas por realizar en orden de prioridad descendente
Work in progress	Tareas realizándose en un momento dado. No más de dos o tres.
Stand-by	Tareas a la espera por motivos ajenos al autor del proyecto.
Done	Tareas finalizadas.
Discarded	Tareas descartadas debido a cambios de diseño, requisitos, etc.
Doubts	Dudas planteadas a lo largo del desarrollo del proyecto.
Resources	Recursos documentales en la web útiles para del desarrollo del proyecto.

Cuadro 5.1: Diseño del tablero *Kanban*.

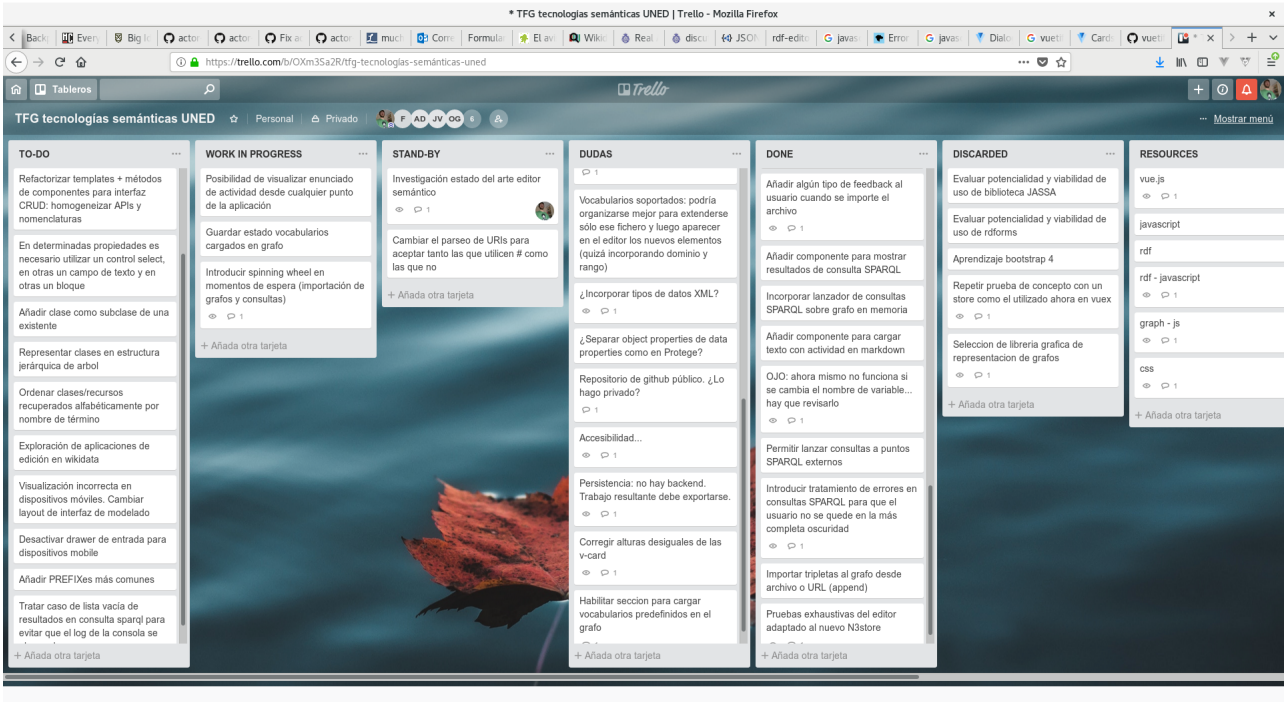


Figura 5.3: Tablero *Trello* del proyecto.

Capítulo 6

Análisis

La captura de requisitos es quizá una de las labores más complicadas a la hora de desarrollar un sistema de información, debido a las "diferencias de impedancia" existentes entre un analista y un usuario de negocio. En el caso particular de este proyecto, el hecho de que el usuario fuera a su vez el Director del mismo ha supuesto una ayuda considerable. No obstante, a continuación se detallan las estrategias que supusieron una mejora sustancial en los niveles de concreción del alcance a delimitar.

6.1. Captura y documentación de requisitos

6.1.1. Captura de requisitos

La técnica de captura de requisitos utilizada para este proyecto ha sido, fundamentalmente, **la entrevista** con el Director de proyecto. Se eligió esta técnica por los siguientes motivos:

- La entrevista, bien a distancia o presencial (y especialmente esta última), permite una mayor implicación del usuario en la captura de requisitos.
- Combinada con una maqueta o prueba de concepto, una entrevista presencial puede dar lugar a la aparición de nuevos requisitos de producto, cambios en las especificaciones e incluso en el enfoque y objetivos del mismo.
- Permite la práctica de la escucha activa y la sugerencia de ideas por parte del analista, aportando un valor añadido que enriquece la simple captura de requisitos.
- Es claramente la técnica más obvia, directa y accesible en el contexto de la realización del proyecto.

Concretamente, se han llevado a cabo varias entrevistas utilizando la plataforma colaborativa *Skype* y una presencial, en la que el autor de este proyecto se desplazó a la sede del departamento en Madrid con objeto de conseguir una comunicación más fluida y un mayor entendimiento a la hora de consensuar las necesidades y funcionalidades requeridas del producto.

La primera entrevista a distancia propició un intercambio de documentos e ideas que desembocó en la elaboración del documento del anteproyecto. El resto de entrevistas a través de *Skype* sirvieron para concretar en mayor medida las tareas a realizar y el enfoque del proyecto. Sin embargo, no fue hasta que no tuvo lugar la reunión presencial cuando realmente se orientaron los trabajos definitivamente, con unos objetivos claramente definidos y una posibilidad para cumplir los hitos propuestos.

A continuación se resumen todas las sesiones de captura de requisitos:

id	Fecha	Resumen de la entrevista
1	18/10/17	Primer contacto y comunicación de ideas iniciales para la confección del anteproyecto
2	21/02/17	Consolidación de ideas y aportación de más documentación (vídeos sobre prototipos de cuaderno, documentos de texto con descripciones, etc.)
3	28/03/18	Primera demo a modo de *Proof Of Concept* (POC) con un entorno capaz de añadir tripletas.
4	10/07/18	Reunión presencial con demostración *in-situ* de los módulos de modelado e importación/exportación. Tiene lugar una tormenta de ideas y se enfoca el proyecto de otro modo, modificando sus objetivos hacia una herramienta formativa.
5	4/08/18	Revisión de los últimos avances con la integración de un endpoint SPARQL en el *frontend* y planificación del resto de funcionalidades requeridas.

Cuadro 6.1: Sesiones de entrevistas

6.1.2. Documentación

Para documentar la captura de requisitos, se utilizará la técnica de casos de uso. Se descarta la incorporación de diagramas *Unified Modeling Language* (UML) de casos de uso, dado que dichos

diagramas carecen de información esencial sobre los mismos (como qué actor lleva a cabo cada paso, o notas sobre el orden de ejecución de los pasos). Si bien pueden ser útiles como resumen o índice de contenidos, se decide prescindir de ellos dado que el número de casos de uso contemplados en el proyecto es manejable.[29]

Se utilizará una plantilla propuesta por Cockburn[30]: el estilo *Rational Unified Process* (RUP)[31], atractivo y fácil de seguir pese al elevado número de apartados, modificado para plasmar los aspectos más relevantes del proyecto (por ejemplo, no se incluirá un campo *ámbito* porque siempre va a estar referido al mismo sistema o aplicación). El motivo de no utilizar una tabla es meramente subjetivo, ya que el autor de esta memoria opina que puede oscurecer el contenido.

La plantilla sigue la siguiente estructura:

1. Nombre del caso de uso
    a) Descripción breve
    b) Actores, entre los que estará el actor principal. Presentan comportamiento.
    c) Disparadores: acciones sobre el sistema que inician los casos de uso.
2. Flujo de eventos
    a) Flujo básico: escenario principal de éxito.
    b) Flujos alternativos: qué puede pasar que no sea el flujo principal.
        1) Condicion 1
        2) Condición 2
        3) ...
3. Requisitos especiales (si se dieran): plataforma, etc.
4. Precondiciones: qué debe ser cierto antes de ejecutar el caso de uso.
5. Postcondiciones: qué debe ser cierto después de ejecutar el caso de uso.

Los requisitos fueron capturados inicialmente como notas manuscritas y convertidos en necesidades de alto nivel en la plataforma *Trello*. A partir de ahí, dichas necesidades se refinaron para dar lugar a la batería de casos de uso incluida en 6.3.

6.2. Necesidades

La reunión presencial marcó un punto de inflexión en cuanto a objetivos del proyecto, lo que se traduce en un cambio de necesidades. Para reflejar la evolución completa, se dividirá su captura en dos fases detalladas a continuación:

6.2.1. Captura inicial

A continuación se resumen, en lenguaje natural, las necesidades identificadas durante la primera fase de desarrollo del proyecto:

1. Permitir a usuarios de la UNED de distintos colectivos etiquetar y generar sus propios cuadernos con información y metainformación semántica.
2. Permitir a dichos usuarios realizar consultas sobre sus cuadernos.
3. Desarrollar una interfaz web que permita al usuario gestionar tripletas RDF.
4. Desarrollar un módulo de generación de consultas SPARQL a partir de consultas de lectura y escritura en formato *JavaScript Object Notation* (JSON).
5. Desarrollar los correspondientes productos de interés para el usuario: consultas exportadas en forma diversa (*Comma Separated Values* (CSV), JSON) o embebidas en una plantilla HTML significativa para el usuario.
6. Ofrecer la posibilidad de variar interfaces de entrada y exportadores en función de los distintos colectivos de usuario utilizando los metadatos sobre cuadernos RDF previamente almacenados en una base de datos relacional.
7. Permitir mostrar en pantalla una serie de términos como punto de partida que el usuario pueda utilizar para construir ternas RDF y relaciones entre ellas.
8. Permitir mostrar en pantalla una serie de términos como punto de partida que el usuario pueda utilizar para construir ternas RDF y relaciones entre ellas.
9. Ofrecer al usuario una visualización sencilla y correcta de su modelo que proporciona una perspectiva adecuada sobre la que trabajar.

10. Presentar una interfaz de mantenimiento del grafo: vocabulario e instancias (conceptualización y poblamiento de una ontología).

11. Importar y exportar información estructurada en formatos semánticos estándar.

12. Extender con vocabularios tales como *Simple Knowledge Organization System* (SKOS) y *Web Ontology Language* (OWL).

6.2.2. Captura final

Una vez celebrada la reunión presencial, se decidió darle otro enfoque al proyecto. Si bien la idea inicial era desarrollar un sistema que permitiese generar cuadernos a través de la manipulación de grafos, una vez presentada una maqueta o prueba de concepto con funcionalidades básicas de modelado el tutor propuso convertir la herramienta en un *playground* o sistema de realización de actividades académicas con un enfoque docente orientado a facilitar el aprendizaje de las tecnologías de la Web Semántica (básicamente RDF y SPARQL) a personas con poco o ningún contacto con estas materias (por ejemplo, alumnos de los Grados de Ingeniería Informática o en Tecnologías de la Información de la UNED).

Este nuevo enfoque se tradujo en la siguiente instantánea de necesidades de alto nivel:

1. Permitir el modelado semántico con edición *Create Read Update Delete* (CRUD) de clases, subclases y propiedades de clases (anotaciones básicas).

2. Permitir la edición CRUD de relaciones o propiedades, subpropiedades, etc.

3. Ofrecer mecanismos de poblamiento del grafo.

4. Permitir el lanzamiento de consultas SPARQL sobre el grafo local y visualización de resultados.

5. Permitir el lanzamiento de consultas SPARQL sobre endpoints remotos y visualización de resultados.

6. Ofrecer funcionalidad de carga de consultas SPARQL predefinidas desde archivo de texto.

7. Incorporar módulo para añadir un texto con la definición de la actividad a realizar en formato *Markdown*.

8. Permitir la importación de tripletas desde archivos o *Uniform Resource Locator* (URL).

9. Permitir la incorporación (append) de tripletas al grafo local desde archivos o URL.
10. Ofrecer un panel para cargar en el grafo vocabularios comunes predefinidos.
11. Presentar una interfaz fácil de usar y responsiva para el usuario, con una gestión de errores adecuada y suficiente.

6.3. Casos de uso

6.3.1. Dar de alta un nuevo recurso (clase o propiedad)

6.3.1.1. Descripción breve

Este caso de uso permite a un usuario añadir un nuevo recurso a su grafo (clase o propiedad indistintamente, según la pestaña en la que el usuario se encuentre) a través de una tripleta que tendrá como sujeto el nombre del recurso, como objeto el tipo de recurso (*owl:Class*, *owl:DatatypeProperty* o *owl:ObjectProperty*, según el caso) y como predicado el recurso *rdf:type*. Se agrupan clases y propiedades en el mismo caso de uso porque son idénticos salvo por el tipo de recursos utilizados para dar de alta las ternas.

6.3.1.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.1.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el botón flotante + asociado a la tarjeta del listado de recursos.

6.3.1.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en el botón flotante + asociado a la tarjeta del listado de recursos.
   b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso.
   c) El usuario introduce el URI del recurso que quiere dar de alta como clase/propiedad y pulsa en guardar.
   d) El nuevo recurso de tipo clase/propiedad se crea y aparece en el listado de clases/propiedades.
2. Flujo alternativo 1
   a) El usuario pulsa en el botón flotante + asociado a la tarjeta del listado de recursos (clases/propiedades).
   b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso.
   c) El usuario introduce (o no) el URI del recurso que quiere dar de alta como clase/propiedad y pulsa fuera del cuadro de diálogo.
   d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
3. Flujo alternativo 2
   a) El usuario pulsa en el botón flotante + asociado a la tarjeta del listado de recursos.
   b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso.
   c) El usuario introduce el URI del recurso que quiere dar de alta como clase/propiedad y pulsa en guardar.
   d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.1.5. Requisitos especiales

Ninguno.

6.3.1.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral y haber seleccionado una de las pestañas existentes (clases o propiedades).

6.3.1.7. Post-condiciones

Ninguna.

6.3.1.8. Extensiones

Ninguna.

6.3.2. Editar un recurso (clase o propiedad) existente

6.3.2.1. Descripción breve

Este caso de uso permite a un usuario editar el recurso asociado a una clase o propiedad existente en su grafo y cambiar su URI.

6.3.2.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.2.3. Disparadores

El caso de uso comienza cuando el usuario pulsa en el icono de menú asociado al recurso (clase/propiedad) que quiere editar y selecciona la acción "Editar".

6.3.2.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en el icono de menú asociado al recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica (o no) el URI del recurso y pulsa en guardar.
   d) El recurso ha sido modificado y su cambio se ve reflejado en el listado.
2. Flujo alternativo 1
   a) El usuario pulsa en el icono de menú asociado al recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica (o no) el URI del recurso y pulsa fuera del cuadro de diálogo.
   d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
3. Flujo alternativo 2
   a) El usuario pulsa en el icono de menú asociado al recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica el URI del recurso y pulsa en guardar.
   d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.2.5. Requisitos especiales

Ninguno.

6.3.2.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien de la barra de navegación lateral y haber seleccionado una de las dos pestañas disponibles (clases o propiedades).

6.3.2.7. Post-condiciones

Ninguna.

6.3.2.8. Extensiones

Ninguna.

6.3.3. Eliminar un recurso (clase o propiedad) existente

6.3.3.1. Descripción breve

Este caso de uso permite a un usuario eliminar el recurso asociado a una clase/propiedad existente en su grafo.

6.3.3.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.3.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el icono de menú asociado al recurso (clase/propiedad) que quiere eliminar y selecciona la acción "Eliminar".

6.3.3.4. Flujo de eventos

1. Flujo básico

a) El usuario pulsa en el icono de menú asociado al recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario elige confirmar.

d) El recurso y su detalle asociado desaparecen del listado.

2. Flujo alternativo 1

a) El usuario pulsa en el icono de menú asociado al recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario pulsa fuera del cuadro de diálogo.

d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.

3. Flujo alternativo 2

a) El usuario pulsa en el icono de menú asociado al recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario elige cancelar.

d) El recurso y su detalle siguen apareciendo y no son eliminados.

4. Flujo alternativo 3

a) El usuario pulsa en el icono de menú asociado al recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario elige confirmar.

d) Ocurre un error en el borrado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.3.5. Requisitos especiales

Ninguno.

6.3.3.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien de la barra de navegación lateral y haber seleccionado una de las pestañas existentes (clases o propiedades).

6.3.3.7. Post-condiciones

Una vez eliminada un recurso de tipo clase, se eliminan también todas las tripletas que lo tengan como objeto o como sujeto (es decir, cualquier relación con dicha clase) con objeto de mantener la consistencia en el grafo.

6.3.3.8. Extensiones

Ninguna.

6.3.4. Dar de alta una nueva anotación de un recurso (clase o propiedad)

6.3.4.1. Descripción breve

Este caso de uso permite a un usuario añadir una nueva anotación (propiedades preseleccionadas) de un recurso (clase/propiedad) a su grafo.

6.3.4.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.4.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el signo + asociado al nombre de la anotación a la que quiere dar un valor.

6.3.4.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en el botón flotante + asociado al panel de anotaciones.
    b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre de la anotación.
    c) El usuario introduce el URI del recurso que quiere dar de alta como anotación y pulsa en guardar.
    d) El nuevo recurso se crea y aparece en el listado de anotaciones correspondiente.
2. Flujo alternativo 1
    a) El usuario pulsa en el botón flotante + asociado al panel de anotaciones.
    b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre de la anotación.
    c) El usuario introduce (o no) el URI del recurso que quiere dar de alta como anotación y pulsa fuera del cuadro de diálogo.
    d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
3. Flujo alternativo 2
    a) El usuario pulsa en el botón flotante + asociado al panel de anotaciones.
    b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre de la anotación.
    c) El usuario introduce el URI del recurso que quiere dar de alta como anotación y pulsa en guardar.
    d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.4.5. Requisitos especiales

Tan sólo se podrán añadir anotaciones o propiedades descriptivas de entre las previamente configuradas en el código fuente de la aplicación (es decir, han de fijarse de manera previa, si bien su extensión es sencilla). La configuración por defecto ofrece una lista de las propiedades más utilizadas.

6.3.4.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral, y haber seleccionado previamente una clase o propiedad de entre el listado de clases/propiedades (en la tarjeta de la izquierda) para poder añadirle una anotación.

6.3.4.7. Post-condiciones

Ninguna.

6.3.4.8. Extensiones

Ninguna.

6.3.5. Editar una anotación de un recurso (clase/propiedad)

6.3.5.1. Descripción breve

Este caso de uso permite a un usuario editar el recurso asociado a una anotación de un recurso (clase/propiedad) existente en su grafo y cambiar su URI.

6.3.5.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.5.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el icono de menú asociado a la anotación de clase/propiedad que quiere editar y selecciona la acción "Editar".

6.3.5.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica el URI del recurso y pulsa en guardar.
   d) El recurso ha sido modificado y su cambio se ve reflejado en el listado de detalle de la propiedad.
2. Flujo alternativo 1
   a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica (o no) el URI del recurso y pulsa fuera del cuadro de diálogo.
   d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
3. Flujo alternativo 2
   a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere editar y selecciona la acción "Editar".
   b) Aparece un cuadro de diálogo con el URI del recurso actual que el usuario puede modificar.
   c) El usuario modifica el URI del recurso y pulsa en guardar.

d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.5.5. Requisitos especiales

Tan sólo se podrá editar una anotación de entre las previamente configuradas en el código fuente de la aplicación (es decir, han de fijarse previamente, si bien su extensión es sencilla).

6.3.5.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien de la barra de navegación lateral, y haber seleccionado previamente una clase/propiedad de entre el listado de clases/propiedades (en la tarjeta de la izquierda) para poder editar una de sus anotaciones a través del menú.

6.3.5.7. Post-condiciones

Ninguna.

6.3.5.8. Extensiones

Ninguna.

6.3.6. Eliminar una anotación de una clase/propiedad

6.3.6.1. Descripción breve

Este caso de uso permite a un usuario eliminar el recurso asociado a una anotación de una clase existente en su grafo.

6.3.6.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.6.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere eliminar y selecciona la acción "Eliminar".

6.3.6.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere eliminar y selecciona la acción "Eliminar".
    b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.
    c) El usuario pulsa fuera del cuadro de diálogo.
    d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
2. Flujo alternativo 1
    a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere eliminar y selecciona la acción "Eliminar".
    b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.
    c) El usuario pulsa fuera del cuadro de diálogo.
    d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.
3. Flujo alternativo 2
    a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario elige cancelar.

d) El recurso y su detalle siguen apareciendo y no son eliminados.

4. Flujo alternativo 3

a) El usuario pulsa en el icono de menú asociado a la anotación del recurso que quiere eliminar y selecciona la acción "Eliminar".

b) Aparece un cuadro de diálogo advirtiendo al usuario de que si confirma el cambio, no podrá volver a recuperar el recurso del grafo.

c) El usuario elige confirmar.

d) Ocurre un error en el borrado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.6.5. Requisitos especiales

Tan sólo se podrá eliminar una anotación de entre las previamente configuradas en el código fuente de la aplicación (es decir, han de fijarse de manera previa, si bien su extensión es sencilla).

6.3.6.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien de la barra de navegación lateral, y haber seleccionado previamente una clase/propiedad de entre el listado de clases/propiedades (en la tarjeta de la izquierda) para poder editar una de sus propiedades a través del menú.

6.3.6.7. Post-condiciones

Ninguna.

6.3.6.8. Extensiones

Ninguna.

6.3.7. Dar de alta un nuevo sub-recurso (sub-clase o sub-propiedad)

6.3.7.1. Descripción breve

Este caso de uso permite a un usuario añadir un recurso derivado de otro (sub-clase o sub-propiedad, en función del tipo de recurso) a su grafo.

6.3.7.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.7.3. Disparadores

El caso de usa comienza cuando el usuario selecciona un recurso en la tarjeta que los lista y pulsa en el icono del menú para seleccionar "Añadir subclase/subpropiedad".

6.3.7.4. Flujo de eventos

1. Flujo básico
    a) El usuario selecciona un recurso en la tarjeta que los lista y pulsa en el icono del menú para seleccionar "Añadir subclase/subpropiedad".
    b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso derivado a crear.
    c) El usuario introduce el URI del recurso que quiere dar de alta como derivado y pulsa en guardar.

d) El nuevo recurso se crea y aparece en el listado de anotaciones correspondiente.

2. Flujo alternativo 1

   a) El usuario selecciona un recurso en la tarjeta que los lista y pulsa en el icono del menú para seleccionar "Añadir subclase/subpropiedad".

   b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso derivado a crear.

   c) El usuario introduce (o no) el URI del recurso que quiere dar de alta como derivado y pulsa fuera del cuadro de diálogo.

   d) El cuadro de diálogo se cierra y no hay ningún cambio en los recursos.

3. Flujo alternativo 2

   a) El usuario selecciona un recurso en la tarjeta que los lista y pulsa en el icono del menú para seleccionar "Añadir subclase/subpropiedad".

   b) Aparece un cuadro de diálogo donde el usuario debe introducir el nombre del recurso derivado a crear.

   c) El usuario introduce el URI del recurso que quiere dar de alta como derivado y pulsa en guardar.

   d) Ocurre un error en el guardado del recurso que le es comunicado al usuario a través de una notificación en pantalla.

6.3.7.5. Requisitos especiales

Ninguno.

6.3.7.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de modelado, bien a través de la pantalla de bienvenida, bien a través de la barra de navegación lateral, y haber seleccionado previamente una clase o propiedad de entre el listado de clases/propiedades (en la tarjeta de la izquierda) para poder añadir un recurso derivado.

6.3.7.7. Post-condiciones

Se crearán dos tripletas en el almacén, una indicando que el nuevo recurso es derivado del padre y otro indicando que es del mismo tipo.

6.3.7.8. Extensiones

Ninguna.

6.3.8. Cargar una actividad

6.3.8.1. Descripción breve

Este caso de uso permite a un usuario cargar en la aplicación un fichero de texto en formato *Markdown* para su visualización. Su objetivo es presentar la actividad formativa a llevar a cabo en un formato legible y de forma persistente mientras se lleve a cabo la misma.

6.3.8.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.8.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el botón de "Importar archivo" presente en la tarjeta de actividad.

6.3.8.4. Flujo de eventos

1. Flujo básico

a) El usuario pulsa en el botón de "Importar archivo" presente en la tarjeta de actividad.

b) Aparece un cuadro de diálogo para seleccionar un archivo en formato *Markdown*.

c) El usuario selecciona un archivo en formato *Markdown* con extensión .md para cargar.

d) El cuadro de diálogo se cierra y aparece un mensaje informando del éxito en la carga, demás de poderse visualizar el contenido de la actividad en la tarjeta de contenido de la misma.

2. Flujo alternativo 1

a) El usuario pulsa en el botón de "Importar archivo" presente en la tarjeta de actividad.

b) Aparece un cuadro de diálogo para seleccionar un archivo en formato *Markdown*.

c) El usuario selecciona un archivo con una extensión distinta a .md para cargar.

d) El cuadro de diálogo se cierra y aparece un mensaje de error informando de que solo es posible cargar archivos en formato *Markdown* (con extensión .md).

3. Flujo alternativo 2

a) El usuario pulsa en el botón de "Importar archivo" presente en la tarjeta de actividad.

b) El usuario selecciona un archivo en formato *Markdown* con extensión *.md* para cargar que ocupa más del máximo permitido (500K, siendo este máximo configurable)

c) El cuadro de diálogo se cierra y aparece un mensaje de error informando de que solo es posible cargar archivos con tamaño inferior al máximo permitido, especificando cuál es dicho máximo.

6.3.8.5. Requisitos especiales

Ninguno.

6.3.8.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de Actividades a través de la barra de navegación lateral.

6.3.8.7. Post-condiciones

La actividad se visualizará con el formato *Markdown* interpretado en HTML en la tarjeta asociada al contenido de la misma.

6.3.8.8. Extensiones

Ninguna.

6.3.9. Cargar un vocabulario

6.3.9.1. Descripción breve

Este caso de uso permite a un usuario cargar en la aplicación los vocabularios más comunes de forma predefinida. Este conjunto de vocabularios es fácilmente extensible a través del código fuente y se carga mediante URLs externas.

6.3.9.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.9.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en uno de los interruptores asociado a cualquiera de los vocabularios más utilizados ofrecidos por la aplicación, siempre y cuando dichos interruptores estén en su posición “desactivado”.

6.3.9.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en un interruptor inactivo de un vocabulario.
   b) El interruptor pasa a estar activo y el vocabulario cargado.
2. Flujo alternativo 1
   a) El usuario pulsa en un interruptor inactivo de un vocabulario.
   b) Ocurre un error al recuperar o cargar el vocabulario y el interruptor vuelve a su posición de desactivado. Simultáneamente, aparece un mensaje de error.

6.3.9.5. Requisitos especiales

Es necesario que haya conexión a Internet para poder descargar los vocabularios de las URLs precargadas así como que los servidores remotos donde residen los vocabularios presenten una configuración correcta de las cabeceras *Cross-Origin Resource Sharing* (CORS)[1]

6.3.9.6. Pre-condiciones

El usuario debe haber navegado previamente hasta la sección de Vocabularios a través de la barra de navegación lateral.

6.3.9.7. Post-condiciones

Las tripletas de los vocabularios activados estarán cargadas en el grafo junto con las del modelo.

6.3.9.8. Extensiones

Ninguna.

[1] https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

6.3.10. Des-cargar un vocabulario

6.3.10.1. Descripción breve

Este caso de uso permite a un usuario des-cargar[2] en la aplicación los vocabularios que haya podido cargar previamente.

6.3.10.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.10.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en uno de los interruptores asociado a cualquiera de los vocabularios más utilizados ofrecidos por la aplicación, siempre y cuando su posición esté en "activado".

6.3.10.4. Flujo de eventos

1. Flujo básico
    a) El usuario pulsa en un interruptor activo de un vocabulario.
    b) El interruptor pasa a estar inactivo y el vocabulario des-cargado.
2. Flujo alternativo 1
    a) El usuario pulsa en un interruptor activo de un vocabulario.
    b) Ocurre un error al recuperar o cargar el vocabulario para su borrado del grafo y el interruptor vuelve a su posición de "activado", puesto que el vocabulario no se ha podido des-cargar. Simultáneamente, aparece un mensaje de error.

[2]Se utiliza el guión para distinguir el término del inglés *unload* y no *download*.

6.3.10.5. Requisitos especiales

Es necesario que haya conexión a Internet para poder descargar los vocabularios de las URLs precargadas y así eliminarlos del grafo local.

6.3.10.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de Vocabularios a través de la barra de navegación lateral.

6.3.10.7. Postcondiciones

Las tripletas de los vocabularios desactivados ya no estarán cargadas en el grafo junto con las del modelo.

6.3.10.8. Extensiones

Ninguna.

6.3.11. Poblar el grafo con una tripleta

6.3.11.1. Descripción breve

Este caso de uso permite a un usuario añadir una tripleta arbitraria al grafo, seleccionando su sujeto (campo libre), predicado (campo de selección que permite introducir nuevos términos) y objeto (campo de selección que permite introducir nuevos términos).

6.3.11.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.11.3. Disparadores

El caso de usa comienza cuando el usuario comienza a rellenar los campos del formulario de poblamiento y pulsa en aceptar cuando ha terminado.

6.3.11.4. Flujo de eventos

1. Flujo básico
    a) El usuario rellena los campos del formulario, bien con los selectores, bien con un texto libre.
    b) El usuario pulsa en añadir instancia.
    c) Aparece un mensaje indicando el éxito de la operación y la instancia se añade al grafo.
2. Flujo alternativo 1
    a) El usuario rellena los campos del formulario pero deja al menos un campo vacío (sin rellenar).
    b) El usuario pulsa en añadir instancia.
    c) Aparece un mensaje de error indicando que ningún elemento de la terna puede ser nulo.

6.3.11.5. Requisitos especiales

Para que las opciones del selector aparezcan precargadas es necesario que, además de los recursos por defecto, se haya cargado algún tipo de modelo en el grafo.

6.3.11.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de Poblamiento a través de la barra de navegación lateral.

6.3.11.7. Postcondiciones

La terna añadida en caso de éxito puede visualizarse utilizando el módulo de consultas o, en caso de ser una terna estructural de modelado, en el módulo correspondiente.

6.3.11.8. Extensiones

Ninguna.

6.3.12. Importar grafo

6.3.12.1. Descripción breve

Este caso de uso permite a un usuario importar un grafo completo en los formatos *N3, Turtle, TriG o N-Triples*. El grafo importado sustituirá al cargado por defecto en memoria por la aplicación.

6.3.12.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.12.3. Disparadores

El caso de usa comienza cuando el usuario importa un grafo desde archivo o desde URL.

6.3.12.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en importar desde archivo o rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) El usuario selecciona el archivo a importar en caso de ser importación desde archivo.
   c) Aparece un mensaje indicando el éxito de la operación y se carga el grafo en el almacén.
2. Flujo alternativo 1
   a) El usuario pulsa en importar desde archivo o rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) El usuario selecciona o apunta a un archivo que supera el límite de tamaño permitido (configurable en la aplicación.)
   c) Aparece un mensaje de error indicando que no es posible cargar el grafo.
3. Flujo alternativo 2
   a) El usuario rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) La URL no está disponible o devuelve un error de descarga (por ejemplo, CORS, si el servidor no está preparado).
   c) Aparece un mensaje de error indicando que no es posible cargar el grafo.

6.3.12.5. Requisitos especiales

Para cargar archivos desde una URL externa, el servidor remoto tiene que tener correctamente configuradas las cabeceras CORS[3].

6.3.12.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de Importar/Exportar a través de la barra de navegación lateral.

[3] https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

6.3.12.7. Postcondiciones

Una vez cargado el grafo, este podrá visualizarse en el módulo de modelado (si tiene ternas estructurales) o en el de consulta en todo caso.

6.3.12.8. Extensiones

Ninguna.

6.3.13. Añadir a grafo

6.3.13.1. Descripción breve

Este caso de uso es una leve modificación del anterior y permite a un usuario añadir un grafo externo al ya existente en el almacén en los formatos *N3, Turtle, TriG o N-Triples*. El grafo importado no sustituirá al cargado por defecto en memoria por la aplicación, sino que pasará a formar parte de él.

6.3.13.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.13.3. Disparadores

El caso de usa comienza cuando el usuario añade un grafo desde archivo o desde URL.

6.3.13.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en añadir desde archivo o rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) El usuario selecciona el archivo a añadir en caso de ser importación desde archivo.
   c) Aparece un mensaje indicando el éxito de la operación y se añade el grafo en el almacén.
2. Flujo alternativo 1
   a) El usuario pulsa en añadir a archivo o rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) El usuario selecciona o apunta a un archivo que supera el límite de tamaño permitido (configurable en la aplicación.)
   c) Aparece un mensaje de error indicando que no es posible cargar el grafo.
3. Flujo alternativo 2
   a) El usuario rellena la URL del mismo y pulsa en el icono con forma de lupa.
   b) La URL no está disponible o devuelve un error de descarga (por ejemplo, CORS, si el servidor no está preparado).
   c) Aparece un mensaje de error indicando que no es posible añadir el grafo.

6.3.13.5. Requisitos especiales

Para cargar archivos desde una URL externa, el servidor remoto tiene que tener correctamente configuradas las cabeceras CORS[4]

6.3.13.6. Precondiciones

El usuario debe haber navegado previamente hasta la sección de Importar/Exportar a través de la barra de navegación lateral.

[4] https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS

6.3.13.7. Postcondiciones

Una vez añadido el grafo, este podrá visualizarse junto con el existente previamente en el módulo de modelado (si tiene ternas estructurales) o en el de consulta en todo caso.

6.3.13.8. Extensiones

Ninguna.

6.3.14. Exportar grafo

6.3.14.1. Descripción breve

Este caso de uso permite a un usuario exportar a un archivo de texto en formato *Turtle* o *JavaScript Object Notation - Linked Data* (JSON-LD) el grafo de trabajo.

6.3.14.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.14.3. Disparadores

El caso de usa comienza cuando el usuario pulsa en el botón de "Exportar JSON-LD" o "Exportar Turtle".

6.3.14.4. Flujo de eventos

1. Flujo básico
   a) El usuario pulsa en el botón de "Exportar JSON-LD" o "Exportar Turtle".
   b) El grafo se exporta y aparece un cuadro de diálogo para guardar el archivo.
2. Flujo alternativo 1
   a) El usuario pulsa en el botón de "Exportar JSON-LD" o "Exportar Turtle".
   b) Ocurre un error inesperado que se muestra al usuario.

6.3.14.5. Requisitos especiales

Ninguno.

6.3.14.6. Precondiciones

Debe existir un grafo cargado en memoria.

6.3.14.7. Postcondiciones

Ninguna.

6.3.14.8. Extensiones

Ninguna.

6.3.15. Lanzar consulta a grafo local

6.3.15.1. Descripción breve

Este caso de uso permite a un usuario lanzar una consulta SPARQL al grafo local cargado en memoria, bien desde un archivo con extensión *".rq"* o URL, bien desde un cuadro de texto.

6.3.15.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.15.3. Disparadores

El caso de usa comienza cuando el usuario carga la consulta pulsa en el botón de "Lanzar Consulta".

6.3.15.4. Flujo de eventos

1. Flujo básico
    a) El usuario carga la consulta a lanzar, bien escribiéndola en el cuadro de texto, bien cargándola desde un archivo *".rq"* o URL donde se encuentre.
    b) El usuario pulsa en el botón de "Lanzar Consulta".
    c) Los resultados de la consulta aparecen en la tabla de "Resultados".
2. Flujo alternativo 1
    a) El usuario carga la consulta a lanzar, bien escribiéndola en el cuadro de texto, bien cargándola desde un archivo *".rq"* o URL donde se encuentre.
    b) El usuario pulsa en el botón de "Lanzar Consulta".
    c) Ocurre un error inesperado o de sintaxis al lanzar la consulta y aparece un mensaje de error para el usuario.

6.3.15.5. Requisitos especiales

Ninguno.

6.3.15.6. Precondiciones

Debe existir un grafo cargado en memoria.

6.3.15.7. Postcondiciones

Ninguna.

6.3.15.8. Extensiones

Ninguna.

6.3.16. Lanzar consulta a grafo remoto

6.3.16.1. Descripción breve

Este caso de uso permite a un usuario lanzar una consulta SPARQL a un grafo ubicado en un *endpoint* remoto, bien desde un archivo con extensión *".rq"* o URL, bien desde un cuadro de texto.

6.3.16.2. Actores

El actor principal es el usuario de la aplicación (fundamentalmente dirigida a un alumno, pero podría ser un profesor o cualquier otro usuario).

6.3.16.3. Disparadores

El caso de usa comienza cuando el usuario activa la opción "Usar *endpoint* SPARQL remoto".

6.3.16.4. Flujo de eventos

1. Flujo básico
    a) El usuario activa la opción "Usar *endpoint* SPARQL remoto".
    b) El usuario introduce la URL del *endpoint* remoto.
    c) El usuario carga la consulta a lanzar, bien escribiéndola en el cuadro de texto, bien cargándola desde un archivo *".rq"* o URL donde se encuentre.
    d) El usuario pulsa en el botón de "Lanzar Consulta".
    e) Los resultados de la consulta al grafo remoto aparecen en la tabla de "Resultados".
2. Flujo alternativo 1
    a) El usuario activa la opción "Usar *endpoint* SPARQL remoto".
    b) El usuario introduce la URL del *endpoint* remoto.
    c) El usuario carga la consulta a lanzar, bien escribiéndola en el cuadro de texto, bien cargándola desde un archivo *".rq"* o URL donde se encuentre.
    d) El usuario pulsa en el botón de "Lanzar Consulta".
    e) Ocurre un error inesperado, de URL o de sintaxis al lanzar la consulta y aparece un mensaje de error para el usuario.

6.3.16.5. Requisitos especiales

Ninguno.

6.3.16.6. Precondiciones

Los endpoints remotos deben soportar el protocolo SPARQL y tener correctamente configuradas las cabeceras CORS.

6.3.16.7. Postcondiciones

Ninguna.

6.3.16.8. Extensiones

Ninguna.

Capítulo 7

Diseño

Un diseño cuidadoso puede suponer un gran ahorro a la hora de llevar a cabo el mantenimiento de una aplicación software. Son, por tanto, labores que requieren de un estudio exhaustivo y que pueden concretarse de lo general a lo particular, a medida que se avanza en el desarrollo. En las siguientes secciones se describe la arquitectura del sistema desde el punto de vista semántico y de *frontend* web.

7.1. Arquitectura de una aplicación semántica

Independientemente de la tecnología elegida para su implementación, una aplicación semántica genérica se caracteriza por una arquitectura concreta que se va a proceder a detallar. Dicha arquitectura está basada en componentes que pueden ser proporcionados por uno o varios productos de mercado. Es común encontrar los siguientes:

- **Almacén de ternas** (*store*): no es más que una base de datos que permite el almacenamiento y extracción de ternas RDF, así como consolidar información procedente de distintas fuentes de datos.
- **Procesador/Serializador** (*parser/serializer*): un procesador RDF lee ficheros de texto y los interpreta como ternas RDF; un serializador realiza el proceso inverso.
- **Motor de consultas** (*query engine*): provee a la aplicación de la funcionalidad de recuperación de información en base a consultas estructuradas.

A continuación se estudian con más detalle, ofreciendo ejemplos de los más utilizados y el caso concreto del proyecto que atañe a esta memoria.

7.1.1. Almacén RDF (*store*)

El almacén de ternas es un componente esencial para salvaguardar conjuntos de datos en ternas independientemente de su origen (un fichero serializado, el resultado de una consulta, etc.). Conceptualmente, el equivalente en una base de datos relacional sería una tabla con tres columnas (*Sujeto* - *Predicado* - *Objeto*). Sin embargo, parece evidente que un almacén RDF esté optimizado para el almacenamiento y recuperación de ternas RDF.

Comparados con una base de datos relacional, sin embargo, son más flexibles y requieren de menor coste de uso y mantenimiento. Más concretamente:

- **Flexibilidad:** su esquema flexible permite realizar cambios sin paradas ni rediseños, cosa que no ocurre con una base de datos relacional.
- **Estandarización:** el nivel de estandarización de RDF y SPARQL es mucho mayor que el de SQL. Es prácticamente inmediato sustituir un almacén de tripletas por otro, mientras que en el caso de las bases de datos relacionales es necesario tener en cuenta los distintos dialectos e implementaciones de SQL según el fabricante.
- **Expresividad:** es mucho más sencillo modelar datos complejos y con elevado número de relaciones entre ellos en RDF que en SQL. Ocurre al contrario si los datos son de naturaleza tabular, naturalmente.

No todo son ventajas en la comparación:

- **Madurez:** las bases de datos relacionales son mucho más maduras y presentan más funcionalidades que los almacenes de ternas.
- **Coste de almacenamiento:** El coste por unidad de información almacenada en un almacén de ternas es mucho mayor que el de una base de datos relacional, lo cual puede ser crítico si se están tratando grandes volúmenes de datos.

Entre los almacenes de ternas más utilizados, se pueden citar *OpenLink Virtuoso*[1], *GraphDB de*

[1] https://virtuoso.openlinksw.com/

Ontotext[2], *Apache Jena TDB*[3] o *AllegroGraph*[4].

7.1.1.1. Almacén en el proyecto

En el caso del presente proyecto, se valoraron dos alternativas:

- ***RDF-Ext Dataset***[5] un simple conjunto de datos cargado en memoria y conforme con la especificación estándar. Este almacén presentaba limitaciones en cuanto a rendimiento y número de tripletas a almacenar, además de estar pensado para operaciones síncronas.
- ***N3.js:***[6] La bilioteca *N3.js* ofrece, además de almacenamiento de tripletas en memoria en *Javascript* nativo, facilidades de serialización y procesado con formatos estándar como *Turtle, N3, N-Triples y TriG*. Si bien su conformidad con la especificación de interfaces *RDF.js* no es completa, sí está presente en gran parte de sus módulos (*DataFactory, StreamParser, StreamWriter y Store*).

Confirmadas las limitaciones de *RDF-Ext Dataset* y consultado con el *W3C RDFJS Community Group*, se opta finalmente por utilizar ***N3.js*** (lo que supuso adaptar ligeramente la implementación del proyecto al nuevo *storage*).

7.1.2. Procesador y Serializador RDF (*parser/serializer*)

En muchos casos, los módulos para importación y exportación de datos en RDF son proporcionados por el propio almacén. Sin entrar a valorar su sintaxis dada su complejidad específica y por considerarse fuera del interés de este proyecto, sí se va a proceder a identificar y caracterizar los formatos de serialización más comunes.

[2] https://ontotext.com/products/graphdb/
[3] https://jena.apache.org/documentation/tdb/
[4] https://franz.com/agraph/allegrograph/
[5] https://github.com/rdf-ext/rdf-store-dataset
[6] https://github.com/rdfjs/N3.js/

7.1.2.1. RDF/XML

Se trata de una representación en XML, comúnmente criticada por su dificultad de lectura por parte de personas. Entre sus otras críticas están las limitaciones impuestas por sus reglas de nomenclatura o los problemas encontrados para procesar este formato con herramientas populares.[32]. Para poder codificar un grafo RDF en XML, tanto los nodos como los predicados han de ser representados en términos de XML (elementos, atributos, contenido de elementos y valores de atributos).

Conceptualmente, se construye a partir de un conjunto de descripciones más pequeñas, cada una de las cuales traza un camino a lo largo de un grafo RDF. Estos caminos se describen en términos de los nodos (sujetos) y enlaces (predicados), que los conectan con otros nodos (objetos).

A continuación se muestra un ejemplo de serialización en RDF/XML[7]:

```
<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<ex:editor>
<rdf:Description>
<ex:homePage>
<rdf:Description rdf:about="http://purl.org/net/dajobe/">
</rdf:Description>
</ex:homePage>
</rdf:Description>
</ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<ex:editor>
<rdf:Description>
<ex:fullName>Dave Beckett</ex:fullName>
</rdf:Description>
</ex:editor>
</rdf:Description>

<rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
<dc:title>RDF 1.1 XML Syntax</dc:title>
</rdf:Description>
```

Listado de código 7.1: Ejemplo de RDF/XML del W3C.

7.1.2.2. *N-Triples*

Se trata de una notación muy simple pero verbosa, donde cada línea representa una única declaración conteniendo sujeto, predicado y objeto seguidos por un punto.

[7] https://www.w3.org/TR/rdf-syntax-grammar/#example3

Ejemplo[8]:

```
<http://example.org/#spiderman> <http://www.perceive.net/schemas/relationship/enemyOf> <http://example.org/#green-goblin> .
```

Listado de código 7.2: Ejemplo de *N-Triples* del W3C.

7.1.2.3. N3

N3 fue un proyecto personal de Tim Berners-Lee[32]. Es muy similar a *N-Triples*, pero añadiendo características adicionales como atajos, un formato más claro, o la condensación de muchas de las repeticiones de este formato.

A pesar de todo, *N3* nunca se convirtió en un estándar y sus características adicionales frente a *N-Triples* no tuvieron mucha aceptación.

7.1.2.4. *Turtle*

El formato Turtle permite escribir un grafo RDF en texto de una forma más compacta que RDF/XML y más sencilla de leer que *N-Triples*. Su gramática es un subconjunto de la especificación del lenguaje de consulta SPARQL 1.1, compartiendo ambas nombres de terminales y producciones en la medida de lo posible. En estos momentos, es el formato de serialización más popular entre las comunidades de desarrolladores[33].

A continuación se muestra un ejemplo de serialización en *Turtle*[9]:

```
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://www.perceive.net/schemas/relationship/> .

<#green-goblin>
rel:enemyOf <#spiderman> ;
a foaf:Person ;    # in the context of the Marvel universe
foaf:name "Green Goblin" .

<#spiderman>
```

[8] https://www.w3.org/TR/n-triples/
[9] https://www.w3.org/TR/turtle/

```
rel:enemyOf <#green-goblin> ;
a foaf:Person ;
foaf:name "Spiderman" .
```

Listado de código 7.3: Ejemplo de Turtle del W3C.

7.1.2.5. Procesadores y serializadores en el proyecto

El almacén seleccionado, ***N3.js*, proporciona procesadores y serializadores** para los principales formatos citados: *N3, N-Triples, Turtle y TriG* (una extensión de este último).

No se ha considerado necesario diseñar ningún serializador o procesador personalizado, al estar ya incluidos los formatos más comunes utilizados por desarrolladores (nótese la ausencia deliberada de RDF/XML).

7.1.3. Motor de consulta RDF (*query engine*)

El motor de consulta es un componente íntimamente ligado al almacén de ternas. El W3C fijó un estándar para consultas: SPARQL (presentado en 2.4), cuya especificación está en su versión 1.1 desde 2013.

Desde el punto de vista de la arquitectura de una aplicación semántica, es deseable que el motor de consultas, además de cumplir con la especificación, esté íntimamente integrado con el almacén de ternas para ofrecer un rendimiento aceptable.

Los principales proveedores de almacenes de ternas ofrecen también motores de consulta integrados, como *Apache Jena, Virtuoso, AllegroGraph...*

7.1.3.1. Motor de consultas en el proyecto

En el caso del presente proyecto, integrar un motor de consultas no fue sencillo. El componente *N3.js* no ofrece dichas funcionalidades, con lo que fue necesario estudiar la viabilidad de incorporar un motor local desarrollado completamente en *Javascript*.

Tras el estudio de los trabajos previos desarrollados (4.1.1) y la situación actual (4.2.3.2), se llegó a la conclusión de que la plataforma *Comunica* se posicionaba como el producto abierto más prometedor para satisfacer las necesidades de consulta SPARQL.

Comunica proporciona las herramientas necesarias para crear una aplicación combinando múltiples bloques de construcción independientes. Su propósito es ofrecer una implementación modular de un cliente *Triple Pattern Fragment* o *Linked Data Fragment*[10], un tipo de fragmento que consiste en:

- **Datos** que corresponden a un patrón de tripleta.
- **Metadatos** que consisten en el total aproximado de tripletas.
- **Controles** que llevan al resto de fragmentos del mismo conjunto de datos.

De forma resumida, un cliente que soporta este tipo de fragmentos puede resolver múltiples consultas SPARQL de forma eficiente.

Comunica consta de los siguientes componentes:

- **Actores:** definen el formato de la entrada que aceptan y la salida correspondiente que producen.
- **Buses:** combinan los actores que soportan el mismo formato de entrada y salida y permite enviar mensajes a todos los actores registrados en un bus dado.
- **Mediadores:** envuelven los buses y se aseguran de que cada petición recibe tan sólo una respuesta.

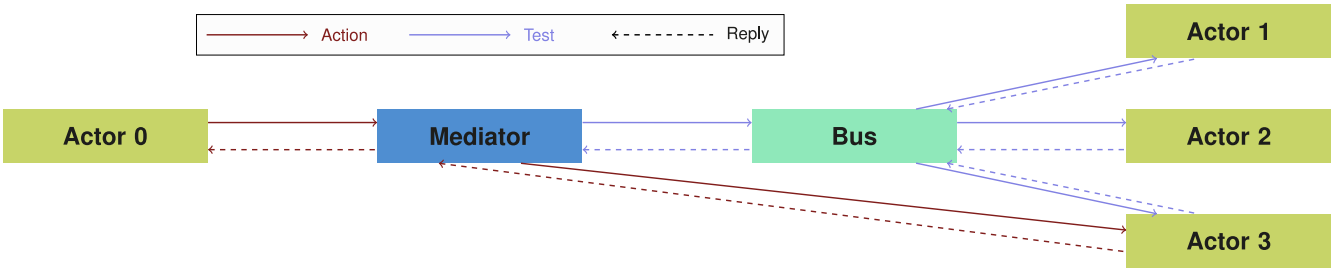


Figura 7.1: Arquitectura de *Comunica*. Fuente: Web de documentación de *Comunica*.

Para el presente proyecto, han sido necesarios dos actores:

[10] http://linkeddatafragments.org/in-depth/#tpf

- **actor-init-sparql:** se trata de un cliente que permite resolver consultas sobre interfaces heterogéneos. En concreto, este módulo inicializa un motor *Comunica* con actores que evalúan consultas SPARQL.
- **actor-init-sparql-rdfjs:** permite lanzar consultas SPARQL a fuentes que implementan la interfaz *Source*[11]. Una de esas fuentes es *N3.js*, el *store* utilizado para almacenar ternas en el proyecto.

Es decir, se utiliza un módulo para lanzar consultas a *endpoints* SPARQL remotos y otro para la fuente o almacén local. Estos módulos **aún están en desarrollo y no han alcanzado una versión estable** a fecha de redacción de la presente memoria, con lo que se van adquiriendo nuevas funcionalidades y corrigiendo defectos a lo largo del tiempo de desarrollo de este proyecto. Por ejemplo:

- En Julio de 2018 el motor de consulta no soportaba la palabra clave SERVICE, característica implementada en Agosto
- En Septiembre de 2018 el motor de consulta no soportaba agregados como COUNT, si bien su incorporación estaba prevista para un par de meses después.

7.2. Arquitectura de la aplicación Vue

Al margen de su naturaleza semántica, la herramienta desarrollada se implementa en *Javascript ES6+* y *Vue.js*. Es necesario, por tanto, proponer un diseño de alto nivel de sus componentes.

7.2.1. Almacenamiento de estado

Para almacenar el estado de la aplicación se ha optado por usar *Vuex*[12].

Vuex es una biblioteca que implementa un patrón de gestión de estado (ver figura 7.2[34]) basado

[11] http://rdf.js.org/#source-interface
[12] https://vuex.vuejs.org/

en *Redux*[13], *Flux*[14] y la arquitectura *Elm*[15]. Se utiliza como un almacén centralizado para todos los componentes de la aplicación, y permite asegurar:

- Que todos los componentes comparten ese estado.
- Que el estado sólo podrá ser modificado de forma controlada, bien síncrona o asíncrona.

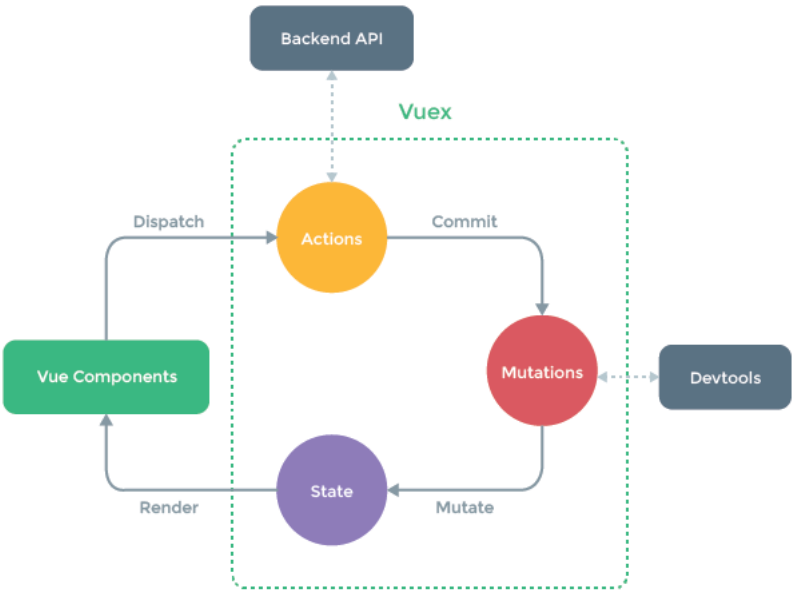


Figura 7.2: *Vuex*: gestión de estado centralizado.

Sin entrar en detalles, el funcionamiento de *Vuex* se ciñe a las siguientes reglas:

- Existe un único estado compartido por todos los componentes de la aplicación, independientemente de que cada componente tenga su propio estado individual.
- El acceso al estado centralizado se realiza a través de **getters**.
- El estado sólo puede ser modificado mediante de **mutaciones síncronas** (*mutations*).
- Las mutaciones pueden ser ejecutadas de manera asíncrona a través de **acciones** (*actions*).

[13] *Redux* es un contenedor predecible del estado de aplicaciones *Javascript*. Ver https://es.redux.js.org/
[14] Una arquitectura de aplicación para *React* que utiliza un flujo de datos unidireccional. Ver https://github.com/facebook/flux
[15] Un patrón sencillo de arquitectura de aplicaciones web. Ver https://guide.elm-lang.org/architecture/

Estado	Descripción
Bloques de construcción RDF	Contiene los bloques más comunes utilizados para modelar un dominio de información en RDF (ej: clases, propiedades, etc.)
Almacén de ternas	Estructura de datos de tipo almacén que albergará las ternas que componen el grafo.
Actividad	Contiene el contenido de la actividad académica a realizar.
Prefijos	Contiene los prefijos más comunes utilizados para precargar en las consultas SPARQL.
Vocabularios	Contiene las definiciones y URLs de los vocabularios más comunes para posibilitar su descarga al grafo en el módulo correspondiente.

Cuadro 7.1: Estado de la aplicación.

ID	*Getter*
GET-1	Recuperar listado de sujetos por predicado y objeto.
GET-2	Recuperar listado de sujetos por predicado.
GET-3	Recuperar listado de objetos por predicado y sujeto.
GET-4	Recuperar listado de ternas cuyo sujeto coincide con uno dado.
GET-5	Recuperar listado de ternas cuyo objeto coincide con uno dado.
GET-6	Recuperar todas las ternas del almacén.
GET-7	Recuperar un listado de las propiedades cargadas por defecto para el modelado.

Cuadro 7.2: Listado de *getters* que recuperan el estado de la aplicación.

El estado de la aplicación puede resumirse con las entidades mostradas en el cuadro 7.1. Por otra parte, se han definido las acciones y mutaciones necesarias para modificar dicho estado. En el cuadro 7.3 se resumen las acciones a implementar con la funcionalidad requerida (no se incluyen las mutaciones por considerarse que no aportan valor añadido en cuanto a diseño que no aporten ya las acciones).

Es interesante también enumerar los *getters* (ver cuadro 7.2), o métodos de consulta que permiten extraer información del estado de la aplicación sin acceder directamente a sus estructuras de datos.

ID	Acción	Responsabilidad
ACC-1	Añadir terna	Añade una terna al grafo pasando como parámetros su sujeto, predicado y objeto.
ACC-2	Eliminar recurso	Elimina todas las ternas que tengan un determinado recurso en cualquiera de sus posibles posiciones (sujeto, predicado, objeto).
ACC-3	Editar recurso	Modifica la definición de un recurso en todas las ternas que lo tengan en cualquiera de sus posibles posiciones (sujeto, predicado, objeto).
ACC-4	Eliminar terna	Elimina una terna del grafo pasando como parámetros su sujeto, predicado y objeto.
ACC-5	Editar literal	Modifica la definición de un literal o cadena de texto utilizado como objeto en una propiedad.
ACC-6	Añadir literal	Añade una terna con un literal o cadena de texto como objeto.
ACC-7	Importar *N3*	Importa un grafo en formato *N3, Turtle, TriG o N-Triples*.
ACC-8	Añadir *N3*	Añade al grafo existente otro en formato *N3, Turtle, TriG o N-Triples*.
ACC-9	Eliminar *N3*	Elimina del grafo local las ternas existentes en otro grafo en formato *N3, Turtle, TriG o N-Triples*
ACC-10	Cargar actividad	Carga el contenido de una actividad en formato Markdown.
ACC-11	Exportar a *JSON-LD*	Exporta el grafo local a un archivo en formato *JSON-LD*.
ACC-12	Exportar a Turtle	Exporta el grafo local a un archivo en formato *Turtle*.
ACC-13	Modificar estado de un vocabulario	Activa o desactiva uno de los vocabularios precargados.

Cuadro 7.3: Listado de *actions* que modifican el estado de la aplicación.

7.2.2. Arquitectura de módulos

En el cuadro 7.4 se plantea el desglose de la aplicación en módulos independientes que dan respuesta a las necesidades expuestas y a los casos de uso concretos.

ID	Módulo	Necesidades	Casos de uso
MOD-1	Actividades	7	6.3.8
MOD-2	Vocabularios	10	6.3.9, 6.3.10
MOD-3	Importar/exportar	8, 9	6.3.12, 6.3.13, 6.3.14
MOD-4	Modelado	1, 2	6.3.1, 6.3.2, 6.3.3, 6.3.4, 6.3.5, 6.3.6, 6.3.7
MOD-5	Poblamiento	3	6.3.11
MOD-6	Consulta	4, 5, 6	6.3.15, 6.3.16

Cuadro 7.4: Matriz de trazabilidad de módulos, requisitos y necesidades.

7.2.3. Arquitectura de componentes

Para poder dar cumplimiento a los requisitos de la aplicación, se definen en el cuadro 7.5 los componentes necesarios con sus ámbitos de responsabilidad.

En la figura 7.3 se muestra un diagrama UML 2.0[29] con la jerarquía de componentes de la aplicación, que muestra convenientemente las relaciones entre los mismos distinguiéndose tres niveles:

- Componente raíz de la aplicación, del que dependen todos los demás (actúa de enrutador).
- Componentes intermedios o contenedores, no reutilizables. Se encargan de gestionar las entradas y salidas de los componentes reutilizables.
- Componentes finales reutilizables. Son los últimos niveles de la jerarquía y realizan tareas muy concretas e independientes.

ID	Componente	Responsabilidad
COMP-1	Contenedor de aplicación	Estructura básica de la aplicación (cabecera, pie, barra lateral).
COMP-2	Bienvenida	Entrada a la aplicación con ayuda y descripción general.
COMP-3	Contenedor de modelo	Contiene los componentes de edición del modelo.
COMP-4	Contenedor de importación/exportación	Contiene los componentes de importación y exportación de datos.
COMP-5	Contenedor de consultas SPARQL	Contiene los componentes de ejecución y visualización de resultados de consultas SPARQL.
COMP-6	Contenedor de actividades	Contiene los componentes de carga y visualización de actividades.
COMP-7	Listado de recursos	Componente reutilizable para listar recursos en una tabla de datos.
COMP-8	Detalle de recursos	Componente reutilizable para visualizar detalles de recursos.
COMP-9	Lanzador de consultas SPARQL	Componente reutilizable para lanzar consultas SPARQL
COMP-10	Listado de resultados de consultas SPARQL	Componente reutilizable para mostrar resultados de una consulta SPARQL.
COMP-11	Cargador de archivos	Componente reutilizable para importar datos desde archivo.
COMP-12	Cargador de URLs	Componente reutilizable para importar datos desde URL.
COMP-13	Cargador de Vocabularios	Componente reutilizable para habilitar o deshabilitar vocabularios.
COMP-14	Visor de *markdown*	Componente reutilizable para visualizar *markdown* renderizado en HTML.
COMP-15	Poblamiento	Componente a modo de formulario para poblar el grafo.
COMP-16	Acerca de	Componente con información relevante sobre la aplicación.
COMP-18	Ayuda	Componente contenedor del manual de usuario.

Cuadro 7.5: Listado de componentes de la aplicación.

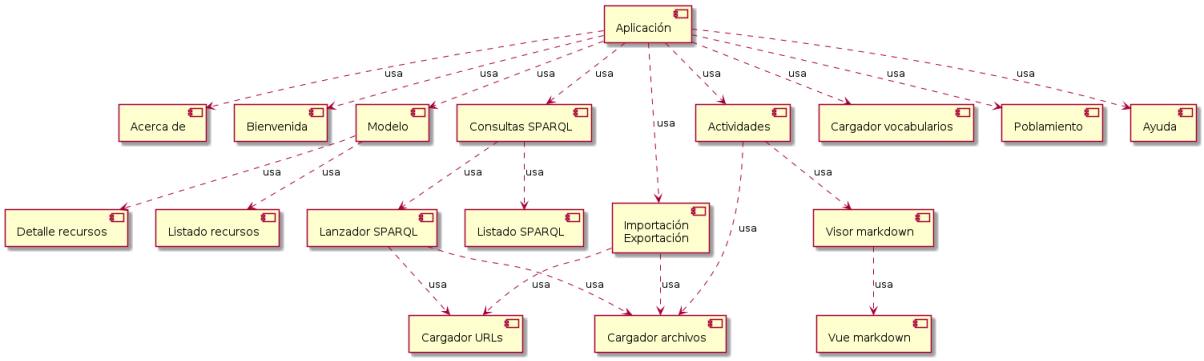


Figura 7.3: Jerarquía de componentes de la aplicación.

Capítulo 8

Implementación y pruebas

La implementación del diseño del software se compara habitualmente con la construcción de un edificio basándose en sus planos[35], pero lo cierto es que el desarrollo de software es una disciplina eminentemente creativa que hace que dicha comparación no termine de ser realmente representativa. Por otra parte, en la filosofía del *Test Driven Development* (TDD) [36], primero se codifica la prueba unitaria y después se implementa la lógica a probar. Este enfoque permite garantizar niveles razonables de calidad en el desarrollo de software. En el presente proyecto no se ha seguido estrictamente debido a la naturaleza del desarrollo, muy orientado a la interfaz gráfica; no obstante, sí se han incluido las pruebas unitarias consideradas como críticas para garantizar un correcto funcionamiento general de la aplicación. El presente capítulo expone detalles relevantes de implementación así como una relación de las pruebas unitarias y extremo a extremo codificadas para garantizar la calidad del producto.

8.1. Implementación

8.1.1. Estándar Javascript

Desde su nacimiento en 1995 de la mano de *Netscape* como lenguaje de *scripting* para mejorar la experiencia final del usuario de Internet a través de un navegador, *Javascript* ha evolucionado hasta convertirse en la tecnología dominante en el desarrollo de *frontend* web.

Es importante describir dicha evolución a través de sus distintos estándares (European Computer Manufacturers Association *(ECMA)Script*) para contextualizar el presente desarrollo:

- ES5, publicado en 2009 y soportado por la práctica totalidad de los navegadores (incluidos los antiguos).
- ES2015, también conocido como ES6,fue la primera de una serie de revisiones del estándar con carácter anual, después de varios años transcurridos desde el estándar anterior. Se trata de una versión rompedora con los estándares anteriores, que incorpora importantes novedades de sintaxis.
- ES2016 y posteriores, que van incorporando más mejoras como la gestión de eventos asíncronos con *async/await*.

Aunque los navegadores están haciendo grandes progresos con su compatibilidad total con ES6, aún no es recomendable utilizarlo en forma nativa. Para resolver este problema surgen los transpiladores o *transpilers* como *Babel*[1] (utilizado en el presente proyecto), que traducen el lenguaje programado en un estándar más moderno como *ES2015+* a *"Vanilla ES5"*, de tal manera que el código pueda ser ejecutado en cualquier navegador.

A la vista de lo expuesto, se plantea la implementación de la presente aplicación en ES6+ (es decir, ES6 y posteriores), pero efectuando un *transpiling* a ES5 para minimizar cualquier problema de compatibilidad con los distintos navegadores.

8.1.2. Entorno

Hoy en día, no es posible hablar de desarrollo profesional y moderno en *Javascript* sin hablar de *Node.js*[2]. *Node.js* es un entorno de ejecución para *Javascript* construido con el motor V8 de *Chrome*[3] y orientado a eventos asíncronos. Está diseñado para construir aplicaciones de red escalables. Introdujo un sistema de módulos que ha sido la base de la reutilización de bibliotecas de código en *Javascript* a través del repositorio *npm*[4], con en torno a 600.000 paquetes de código abierto disponibles para la comunidad de desarrolladores.

[1] https://babeljs.io/
[2] https://nodejs.org
[3] El motor de alto rendimiento de Google para *Javascript* escrito en C++. Ver https://developers.google.com/v8/
[4] https://docs.npmjs.com/getting-started/what-is-npm

En la implementación de esta aplicación se han utilizado las siguientes versiones de estos componentes, cuya instalación previa es necesaria para la construcción y ejecución de la misma:

- *Node.js 10.7.0*
- *npm 6.1.0*

Tanto *Node* como *npm* pueden descargarse e instalarse en múltiples sistemas operativos desde sus respectivas páginas web.

8.1.3. Estructura del proyecto

El proyecto presenta la siguiente estructura jerárquica de directorios:

- **build**: contiene los archivos de configuración de *Webpack*[5], tanto para el servidor de desarrollo como para el entorno de producción.
- **config**: contiene los archivos de configuración del proyecto por entorno (*dev, test, pro*). Si bien no es conveniente modificarlo, para la presente aplicación tuvo que añadirse el soporte para la biblioteca *babel-polyfill* para permitir la correcta ejecución de determinados paquetes en todos los navegadores.
- **dist**: contiene los archivos que deberán ser desplegados en la plataforma de producción (por ejemplo, *Firebase*).
- **docs**: notas del proyecto en *Markdown* y código fuente en LaTeXde esta memoria.
- **node_modules**: contiene todos los paquetes y dependencias de *npm* necesarios para ejecutar el proyecto.
- **src**: contiene el código fuente de la aplicación y está estructurado, a su vez, de la siguiente forma:
  - **components**: contiene los distintos componentes de la aplicación, siendo *App.vue* el principal.
  - **fonts**: contiene las fuentes necesarias para visualizar correctamente *Material Design*.
  - **router**: contiene la configuración de *vue-router*, un enrutador para *Vue.js*.

[5] Un empaquetador de módulos JS.

  - **services**: contiene la configuración única de servicios para recuperar información de APIs REST a través de bibliotecas como *axios*[6].
  - **store**: contiene la lógica necesaria para mantener el estado único de la aplicación mediante *Vuex* (ficheros de acciones, mutaciones, *getters* y el propio almacén).
  - **utils**: contiene utilidades o información por defecto necesaria para el correcto funcionamiento de la aplicación.
- **static**: recursos estáticos que son copiados directamente.
- **test**: contiene la configuración y especificaciones de las pruebas unitarias y extremo a extremo.

Además, en el directorio raíz de la estructura del proyecto se pueden encontrar, entre otros, los siguientes archivos de configuración relevantes:

- **.babelrc**: configuración del *transpiler Babel*.
- **.gitignore**: archivos de proyecto que no deberán subirse al repositorio (binarios, temporales, etc.)
- **.eslintrc.js**: configuración del analizador estático de estilo y sintaxis *ESlint*.
- **package.json**: contiene la configuración de la tarea de construcción y la especificación de dependencias del proyecto.
- **.firebaserc**: configuración de proyecto de *Firebase*.
- **firebase.json**: configuración de despliegue en *Firebase* (ver subsección 8.1.6).

8.1.4. Construcción del proyecto

El proyecto de trabajo se ha generado usando la herramienta *vue-cli*[7] en su versión 2 (durante el tiempo de desarrollo de este proyecto se publicó la versión 3). Esta herramienta permite generar proyectos configurados a partir de una plantilla y selección de parámetros. En concreto, se utilizó una plantilla propuesta por *Vuetify*[8], una biblioteca de componentes de interfaz de usuario basada en

[6] Un cliente *Hypertext Transfer Protocol* (HTTP) basado en promesas para Node.js. Ver https://github.com/axios/axios
[7] https://cli.vuejs.org/
[8] https://vuetifyjs.com

Material Design[9] para *Vue*. Se eligió esta biblioteca por ser una de las más usadas entre la comunidad de usuarios, disponer de un amplio catálogo de componentes implementados y presentar líneas de trabajo prometedoras para finales de este año (con la previsión de publicación de nuevos componentes y de mejora de los actuales).

La propia web de *Vuetify* presenta un asistente para elegir la plantilla más adecuada basándose en diversos parámetros introducidos por el usuario, tales como: experiencia con *Vue*, tipo de interfaz (web, móvil, escritorio), necesidad o no de generar estructuras de pruebas unitarias, etc. Dadas las necesidades de este proyecto, se optó por elegir una plantilla para programadores con poca experiencia en *Vue*, para la web y con estructuras de pruebas unitarias (y no con posicionamiento web).

El resultado obtenido fue la plantilla *webpack*[10], que presenta las siguientes características:

- Desarrollo:
  - Incluye *Webpack* y *vue-loader* para *Single File Component* (SFC).
  - Recarga en caliente con preservación de estado.
  - Análisis estático de estilo con *ESlint*[11] al guardar.
- Construcción:
  - Minificación de JS con *UglifyJS*[12].
  - Minificación de HTML con *html-minifier*.
  - Minificación de *Cascade Style Sheets* (CSS) con *cssnano*.
  - Compilación de recursos estáticos con hashes de versiones para cacheo eficiente.
- Pruebas:
  - Soporte de *ES2015+* en archivos de test.
  - *Jest*[13] como plataforma de testing.
  - *End-to-end*[14] con *Nightwatch*[15].

[9] Un sistema de diseño para desarrollar interfaces de usuario impulsado por *Google*. Ver https://material.io/

[10] Webpack es un empaquetador de módulos para JS. Ver https://github.com/vuetifyjs/webpack y https://webpack.js.org/

[11] https://eslint.org/

[12] https://github.com/mishoo/UglifyJS

[13] https://jestjs.io/

[14] Las pruebas *end-to-end* o extremo a extremo permiten verificar el flujo de trabajo completo de una aplicación web. Son más lentos que los unitarios y difíciles de depurar, pero comprueban la funcionalidad completa de la aplicación.

[15] http://nightwatchjs.org/

8.1.5. Ejecución del proyecto

De manera predeterminada, es posible ejecutar los siguientes comandos:

- **npm install**: instala todas las dependencias del proyecto.
- **npm run dev**: sirve el contenido con recarga en caliente a través del puerto 8080.
- **npm run build**: construye el proyecto para desplegar en producción (con compresión).
- **npm run build –report**: construye el proyecto para producción y visualiza el informe del analizador de empaquetado.
- **npm run unit**: ejecuta las pruebas unitarias.
- **npm run e2e**: ejecuta las pruebas *end-to-end*.
- **npm test**: ejecuta todas las pruebas.

8.1.6. Despliegue del proyecto

Para desplegar la aplicación a modo de demostración se ha optado por utilizar *Firebase*[16], una plataforma de desarrollo de *Google*, a modo de *Platform as a Service* (PaaS)[17]. Si bien *Firebase* provee de muchos servicios (como base de datos, autenticación, almacenamiento, etc.), para este proyecto tan solo ha sido necesario utilizar el servicio de alojamiento. Para ello, basta con incluir un par de sencillos ficheros de configuración con el siguiente contenido:

[16] https://firebase.google.com/
[17] Entorno de desarrollo e implementación en la nube.

```
{
        "hosting": {
                "public": "./dist",
                "ignore": [
                        "firebase.json",
                        "**/.*",
                        "**/node_modules/**"
                ]
        }
}
```

Listado de código 8.1: Configuración de Firebase.

```
{
        "projects": {
                "default": "unedtfg-198720"
        }
}
```

Listado de código 8.2: Configuración de proyecto en Firebase.

El proyecto puede desplegarse en cualquier servidor web o HTTP (como *Apache*[18], por ejemplo). Para ello, será necesario seguir los siguientes pasos:

- **npm run build:** Construcción del proyecto en /dist.
- Copia de la carpeta */dist* al servidor web donde se vaya a desplegar.
- Configuración del servidor web para servir el proyecto construido y empaquetado.

[18] *Apache Web Server* es el servidor HTTP con mayor cuota de mercado. Ver https://news.netcraft.com/archives/2018/07/19/july-2018-web-server-survey.html

El entorno de demostración del proyecto es accesible con conexión a Internet a través de la siguiente URL: https://unedtfg-198720.firebaseapp.com/#/

8.1.7. Estructuras soportadas para el modelado

Dado el abrumador número de ontologías disponibles[19], para la implementación de la funcionalidad de modelado en el presente proyecto se ha optado por incluir en memoria un subconjunto de ternas pertenecientes a los vocabularios más utilizados. Usando como referencia [10], las sugerencias del Director del proyecto y criterios relacionados con el alcance de la funcionalidad implementable para esta primera versión, se ha decidido dar soporte a las estructuras listadas en el cuadro 8.1

8.1.8. Detalles y restricciones de implementación relevantes

Sin pretender entrar a fondo en los detalles de la implementación, que puede consultarse en el repositorio de código fuente del proyecto en: https://github.com/predicador37/rdf-editor, sí se ha considerado conveniente documentar determinadas decisiones:

- En el módulo de **modelado**, el listado de recursos se ha implementado mediante una tabla para permitir su paginado, debido a que el número de clases o propiedades fDOMpuede ser elevado en un modelo más o menos complejo. Por otra parte, los detalles de anotaciones (propiedades de clase, por ejemplo) se implementaron inicialmente con listas expandibles, pero finalmente se optó por paneles expandibles y tablas para permitir una mejor experiencia de usuario en dispositivos móviles.
- En este mismo módulo, el soporte para literales se ha reducido a las propiedades *rdfs:label* y *rdfs:comments*. Asimismo, el resto de propiedades tan solo soportan recursos en forma de URI (igualmente que en el módulo de poblamiento). Esta decisión viene motivada por la necesidad de reducir el alcance del proyecto y simplificar la herramienta de modelado, si bien una mejora en este sentido se incluye como futura línea de trabajo en la sección 10.2.
- En el módulo de **vocabularios**, la selección de estos se ha implementado como un conjunto de interruptores o *switches on/off*. Esto le permite al usuario cargar o descargar los vocabularios de forma sencilla, como si de una configuración se tratase (tal es la idea). No obstante, es

[19]Ejemplo: lista de ontologías de Protégé en https://protegewiki.stanford.edu/wiki/Protege_Ontology_Library

Recurso	Descripción
rdf:type	Permite definir una relación entre una instancia y su clase correspondiente.
rdf:Class	Categoría para clasificar recursos.
rdfs:subClassOf	Indica que todas las instancias de una clase lo son también de la clase padre.
rdfs:subPropertyOf	Indica que un recurso asociado a otro con una propiedad también lo está por su propiedad padre.
rdfs:domain	Colección de tipos que usan una propiedad.
rdfs:range	Tipos de valores que puede tomar una propiedad.
owl:Thing	El conjunto de **todos** los individuos.
owl:Class	Categoría para clasificar recursos o individuos. Es subclase de *rdf:Class*.
owl:DatatypeProperty	Relaciona individuos con literales.
owl:ObjectProperty	Relaciona individuos con otros individuos.
rdf:XMLLiteral	Clase de todos los valores literales de XML.
owl:equivalentClass	Enlaza dos clases que contienen exactamente el mismo conjunto de individuos.
owl:equivalentProperty	Indica que dos propiedades tienen la misma extensión (conjunto de instancias asociadas).
owl:disjointWith	Implica que los conjuntos de individuos de las clases que relaciona no tienen ninguna instancia en común.
owl:propertyDisjointWith	Implica que los conjuntos de individuos de las propiedades que relaciona no tienen ninguna instancia en común.
rdfs:label	Proporciona una versión legible del nombre de un recurso.
rdfs:comment	Proporciona una versión legible de la descripción de un recurso.
rdfs:seeAlso	Indica que el recurso relacionado podría proporcionar información adicional sobre el sujeto.

Cuadro 8.1: Estructuras soportadas para el modelado.

necesario disponer de red y de servicios con CORS correctamente configurado para que su carga sea posible.

- En el módulo de **poblamiento**, se tiene lo siguiente:
  - El campo "sujeto" se implementa como una entrada de texto libre, puesto que lo más probable es que se trate de un individuo nuevo.
  - Los campos "predicado" y "objeto" ofrecen un selector con autocompletado que permite la introducción de nuevos términos. De esta forma, el usuario puede aprovechar los términos ya introducidos en la aplicación (bien por defecto bien a través de otros medios) o añadir los suyos propios, si bien ha de tener prevista su definición en el grafo. Los URIs de los elementos mostrados se han recortado para permitir una mejor visualización en dispositivos móviles.
- En el módulo de actividades, la importación a *Markdown* se realiza mediante la biblioteca *vue-markdown*[20], que a su vez utiliza *markdown-it*[21]. Esta última biblioteca implementa un procesador del formato *Markdown* con buen rendimiento y conforme a la especificación *CommonMark Spec* 0.28[22], si bien añade extensiones de sintaxis (como tablas y tachados de *Github Flavored Markdown*[23]) y otros aspectos.

8.2. Pruebas

8.2.1. Pruebas unitarias

La codificación de pruebas unitarias ayuda a resolver diversos problemas inherentes al desarrollo de software[36]:

- Extensión innecesaria del alcance: muchas veces se tiende a desarrollar funcionalidad extra como previsión de uso en el futuro. Esto lleva a generar productos software con un alto porcentaje de funcionalidades que no se utilicen. Las pruebas unitarias ayudan al desarrollador a centrarse en lo que realmente debe hacer el sistema y a conseguir un ritmo adecuado de trabajo.

[20] https://github.com/miaolz123/vue-markdown

[21] https://github.com/markdown-it/markdown-it

[22] https://spec.commonmark.org/

[23] La sintaxis de Markdown propuesta por Github, que añade características como tablas, estilo de texto, etc. Ver https://help.github.com/articles/basic-writing-and-formatting-syntax/#styling-text

- Cohesión y acoplamiento: si una prueba unitaria resulta difícil de escribir, lo más probable es que se trate de un problema del diseño y no de la prueba en sí.
- Confianza: una *suite* de pruebas unitarias funcionando generan confianza en que el código desarrollado hace lo que debe hacer de forma correcta.

Con objeto de aportar el máximo valor posible con un esfuerzo razonable, se ha optado por codificar pruebas unitarias de los siguientes elementos de la aplicación:

- **Actions**: métodos que permiten modificar el estado general de la aplicación de forma asíncrona. Al llamar a su vez síncronamente a las mutaciones (*mutations*), no se ha considerado necesario codificar pruebas unitarias para estas últimas.
- **Getters**: métodos que permiten recuperar diferentes variables de estado de la aplicación, o subconjuntos de estas a través de un procesado adicional.

En los cuadros 8.2 y 8.3 se muestran sendos listados de las pruebas codificadas (indicando qué se está probando) y los *actions* y *getters* relacionados. No se ha considerado necesario probar la exportación a distintos formatos por tratarse de una característica intrínseca de la biblioteca utilizada (y por lo tanto, probada en dicha biblioteca) cuyo tratamiento no ha sido modificado en esta aplicación.

La propia biblioteca *jest* incluye herramientas para calcular la cobertura de pruebas unitarias, que en este caso, según dichas herramientas, resulta ser de en torno a un 40 % para el código relacionado con el estado de la aplicación. Sin embargo, **este indicador falsea la cobertura real**, puesto que carencias en la integración de *jest* con *Vuex* impiden considerar como válidas las pruebas de las llamadas a las mutaciones, por ejemplo, que sí están siendo probadas pero no aparecen reflejadas en este indicador.

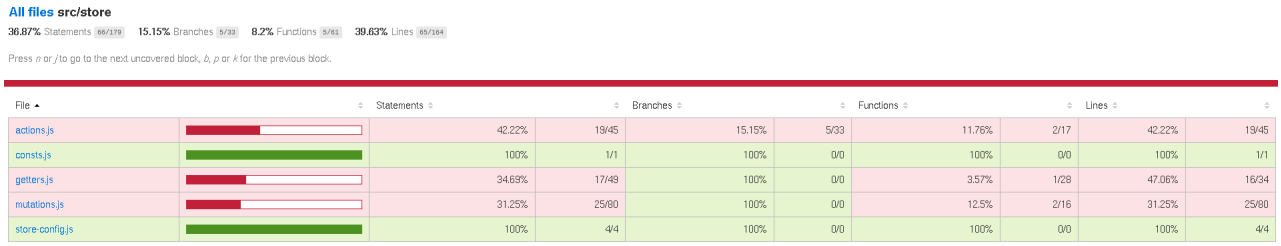
No obstante lo anterior, y con un interés meramente informativo, se incluyen en la figura 8.1 los resultados mostrados por *jest*.

ID	Prueba	Descripción	*Action*
PU-1	Añadir terna	Añade una terna al grafo y comprueba que el número total de ternas de este se ha incrementado en uno.	ACC-1
PU-2	Eliminar ternas según recurso	Elimina todas las ternas que tengan el recurso "Analista" en cualquiera de sus posiciones; se cuentan las ternas eliminadas y se comprueba que el tamaño final del grafo es correcto.	ACC-2
PU-3	Editar recurso	Se edita el recurso "Analista" cambiándolo por otro nombre y se recuperan las ternas asociadas al antiguo y nuevo recurso antes y después de su edición, comprobando que el número de ternas asociadas es el correcto en cada caso.	ACC-3
PU-4	Editar literal	Se comprueba que, tras editar el literal asociado a un objeto de una terna recuperada, el contenido de este cambia en consecuencia.	ACC-5
PU-5	Añadir literal	Añade un literal como propiedad de un recurso y se comprueba que el número de propiedades de tipo literal para ese recurso se incrementa en uno.	ACC-6
PU-6	Importar *N3*	Se incluye cadena de texto simulando un fichero en formato *Turtle*, se importa y se comprueba que el número de ternas del grafo coincide con las importadas.	ACC-7
PU-7	Añadir *N3*	Se incluye cadena de texto simulando un fichero en formato *Turtle*, se añade al grafo actual y se comprueba que el número de ternas del grafo resultante es la suma de las que tenía anteriormente más las importadas.	ACC-8
PU-8	Eliminar *N3*	Se incluye cadena de texto simulando un fichero en formato *Turtle*, se añaden las ternas al grafo, se cuenta el número total de ternas, se eliminan de nuevo del grafo actual y se comprueba que el número de ternas del grafo resultante es el original.	ACC-9
PU-9	Importar actividad	Se importa una actividad en formato *Markdown* y se comprueba que la variable de estado que aloja su contenido deja de estar vacía.	ACC-10
PU-10	Cargar vocabulario	Se verifica que el cambio de estado de un vocabulario de inactivo a activo ocurre, consultando el mismo tras la operación.	ACC-13

Cuadro 8.2: Listado de pruebas unitarias de acciones (*actions*).

ID	Prueba	Descripción	*Getter*
PU-11	Recuperar listado de sujetos por predicado y objeto	Se recuperan todos los sujetos de tipo owl:Class y se comprueba que el número devuelto es correcto.	GET-1
PU-12	Recuperar listado de sujetos por predicado	Se recupera una lista de sujetos de tipo anotación, por ejemplo, y se comprueba que se devuelve el número de correcto de resultados.	GET-2
PU-13	Recuperar listado de objetos por predicado y sujeto	Se recuperan todos los objetos para el sujeto "Analista" y el predicado etiqueta, y se comprueba que el número devuelvo es el correcto.	GET-3
PU-14	Recuperar listado de ternas para un sujeto dado	Se recupera un listado de ternas para el sujeto "Analista" y se comprueba que el número devuelto es el correcto.	GET-4
PU-15	Recuperar listado de ternas para un objeto dado	Se recupera un listado de ternas cuyo objeto es owl:Class y se comprueba que el número resultante es el correcto.	GET-5
PU-16	Recuperar listado de todas las ternas del almacén	Se recuperan todas las ternas y se comprueba que el resultado es el número total de ternas cargadas en el grafo.	GET-6
PU-17	Recuperar listado de todas las propiedades cargadas por defecto	Se recupera el listado de todas las propiedades cargadas por defecto en la aplicación y se comprueba que el número total resultante es el correcto.	GET-7

Cuadro 8.3: Listado de pruebas unitarias de *getters*.


All files src/store

36.87% Statements 66/179 15.15% Branches 5/33 8.2% Functions 5/61 39.63% Lines 65/164

Press n or j to go to the next uncovered block, b, p or k for the previous block.

File	Statements		Branches		Functions		Lines	
actions.js	42.22%	19/45	15.15%	5/33	11.76%	2/17	42.22%	19/45
consts.js	100%	1/1	100%	0/0	100%	0/0	100%	1/1
getters.js	34.69%	17/49	100%	0/0	3.57%	1/28	47.06%	16/34
mutations.js	31.25%	25/80	100%	0/0	12.5%	2/16	31.25%	25/80
store-config.js	100%	4/4	100%	0/0	100%	0/0	100%	4/4

Figura 8.1: Cobertura calculada por *jest* (no real).

8.2.2. Pruebas extremo a extremo (*end-to-end*)

Las pruebas extremo a extremo (o *end-to-end*) permiten probar la interfaz de usuario a través de un navegador automatizado. En su forma más básica, este tipo de pruebas ayuda a comprobar la existencia de determinados elementos en el DOM de la aplicación web, así como verificar que su contenido es correcto.

Para el presente proyecto se ha optado por codificar una prueba extremo a extremo por cada módulo definido en la sección 7.2.2, a modo de *sanity check* o prueba de cordura. En el cuadro 8.4 se detalla un listado de los mismos relacionados con sus correspondientes módulos:

ID	Prueba	Descripción	Módulo
E2E-1	Actividad	Se comprueba número de *inputs* y el contenido de párrafo.	MOD-1
E2E-2	Vocabulario	Se comprueba número de *inputs* y el contenido de una cabecera.	MOD-2
E2E-3	Importar/exportar	Se comprueba número de botones y el contenido de una cabecera.	MOD-3
E2E-4	Modelado	Se comprueba el número de enlaces y el contenido de una pestaña.	MOD-4
E2E-5	Poblamiento	Se comprueba el número de *inputs* y una cabecera.	MOD-5
E2E-6	Consulta	Se comprueba el contenido de una etiqueta y el número de *textarea*.	MOD-6

Cuadro 8.4: Listado de pruebas extremo a extremo (*e2e*).

Capítulo 9

Resultados

El presente proyecto ha generado los siguientes entregables:

- Código fuente de la aplicación desarrollada en *Javascript* / *Vue* y sus baterías de pruebas.
- Código fuente de la presente memoria en LaTeXy binario en *Portable Document Format* (PDF).
- Código fuente de diagramas UML 2.0 con *PlantUML*[1].
- Aplicación desplegada en *Firebase* en https://unedtfg-198720.firebaseapp.com/#/
- Tablero *Trello* con notas del proyecto en https://trello.com/b/OXm3Sa2R/tfg-tecnolog%C3%ADas-sem%C3%A1nticas-uned (en el momento de redacción de la presente memoria, de acceso privado).

El repositorio público en *Github* que alberga todo el código fuente citado es:
https://github.com/predicador37/rdf-editor

En general, el producto obtenido presenta una interfaz sobria, ordenada y efectiva que permite un rápido acceso a todas sus funcionalidades de consulta, carga de datos, modelado, etc. Es perfectamente válido para su trabajo con datos reales extraídos de otros conjuntos de datos de mayor tamaño y utilizable desde el primer momento sin requerir de más herramientas que un sencillo navegador compatible con *Javascript 2015* (cualquiera de los navegadores más utilizados en sus últimas versiones lo es). En el cuadro 9.1 se analiza en qué medida y de qué manera se ha tratado de dar cumplimiento a los objetivos marcados.

[1]Herramienta de diagramado basada en texto plano y de fuentes abiertas. Ver http://plantuml.com/

id	Objetivo	Respuesta
1	Facilitar el aprendizaje y la **familiarización con tecnologías de la Web Semántica** a usuarios inexpertos.	Para verificar el cumplimiento de este objetivo será necesario utilizar la aplicación en producción con actividades reales de alumnos.
2	Reducir al máximo los requisitos de uso de la herramienta para alcanzar el mayor público objetivo posible.	La herramienta no requiere de instalación para su uso; una vez desplegada en un servidor web, cuenta con ejemplos cargados por defecto con los que el alumno puede empezar a trabajar inmediatamente.
3	Ofrecer a los Equipos Docentes la posibilidad de **distribuir actividades formativas** auto-contenidas para que los alumnos trabajen sobre ellas en la herramienta.	La aplicación incorpora un módulo para cargar actividades manualmente desde texto en formato *Markdown* o a través de un servicio web remoto. Ver MOD-1 en 7.2.2
4	Agrupar en un solo producto **funcionalidades básicas de edición y consulta sobre grafos RDF**, de cara a cubrir las expectativas educativas en este ámbito.	El sistema desarrollado consta de varios módulos, entre ellos uno de modelado básico y otro de consultas SPARQL, que el alumno puede utilizar de forma razonablemente autónoma. Ver MOD-4, MOD-6 en 7.2.2
5	Permitir a los alumnos comprobar *in situ* los efectos de sus acciones sobre un grafo RDF y ofrecerles la **flexibilidad suficiente** como para permitirles dar rienda suelta a su creatividad e inquietudes.	El grafo por defecto que proporciona la aplicación vuelve a recargarse cada vez que se refresca el navegador. El alumno es libre de realizar las operaciones que desee, incluso de índole destructiva, y el estado inicial de la aplicación se restablecerá fácilmente. Ver MOD-4, MOD-5 en 7.2.2
6	Obtener un MVP que pueda ser evolucionado y poner su código a disposición de la comunidad educativa para revertir sobre ella su valor aportado.	El propio sistema es un MVP preparado para ser evolucionado con distintas características y mejoras sobre las desarrolladas inicialmente.

Cuadro 9.1: Verificación del cumplimiento de los objetivos iniciales.

9.1. Accesibilidad

La accesibilidad web tiene como objetivo extender el uso de la web al mayor número posible de personas, incluyendo a aquellos con cualquier tipo de limitación, ya sea geográfica, socio-económica, funcional, etc.

Actualmente, la legislación española (*Real Decreto 1494/2007 de 12 de noviembre, por el que se aprueba el Reglamento sobre las condiciones básicas para el acceso de las personas con discapacidad a las tecnologías, productos y servicios relacionados con la sociedad de la información y medios de comunicación social*[2]) establece que la "*Norma UNE 139803:2012: Requisitos de accesibilidad para contenidos en la Web*" define los requisitos de accesibilidad para los contenidos web. Esta norma es equivalente a los criterios *Web Content Accesibility Guidelines 2.0* (WCAG) 2.0[3].

Para verificar el grado de accesibilidad de la aplicación se ha utilizado el complemento Axe[4] para el navegador *Firefox*. Es una de las herramientas más utilizadas por la comunidad de desarrolladores, abierta y flexible, que permite validar si el HTML generado es conforme a las pautas WCAG 2.0 (y, por tanto, a la norma obligada en España).

Validar la accesibilidad de una aplicación 100 % *Javascript* es complejo, dado que su contenido y el árbol del documento (DOM[5]), una API para documentos HTML y XML se generan dinámicamente. No obstante, el funcionamiento a modo de complemento de navegador de la herramienta utilizada ha permitido facilitar esta tarea y su aplicación en los distintos módulos de la aplicación (ver ejemplo en figura 9.1)

[2] https://www.boe.es/buscar/act.php?id=BOE-A-2007-19968
[3] https://www.w3.org/TR/WCAG20/
[4] https://www.deque.com/axe/
[5] Ver https://www.w3.org/TR/WD-DOM/introduction.html

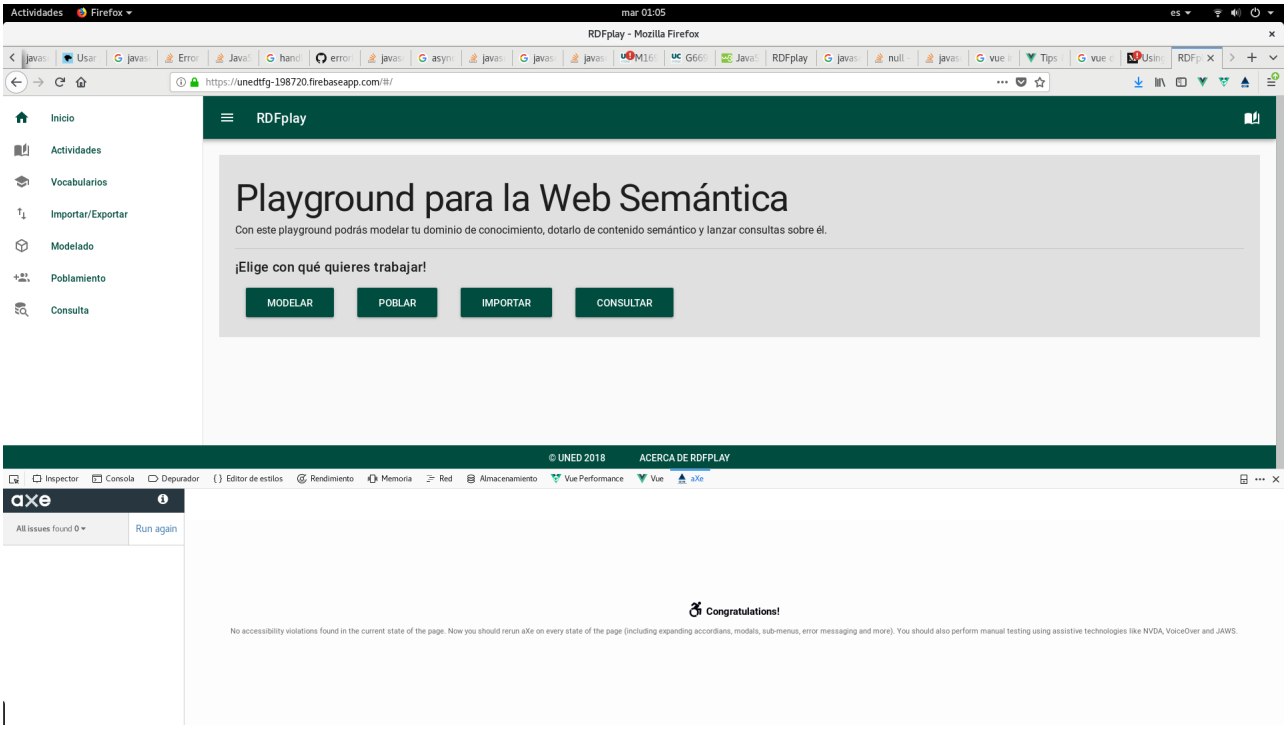


Figura 9.1: Validación de accesibilidad con *Axe* para *Mozilla Firefox*.

Otra herramienta comúnmente utilizada para la validación de la accesibilidad en aplicaciones web es *Lighthouse*[6], instalable como un plugin del navegador *Chrome*. Según su esquema de reglas, *RDFplay* es 100 % accesible, tal y como se puede comprobar en la figura 9.2

[6] https://developers.google.com/web/tools/lighthouse/#devtools

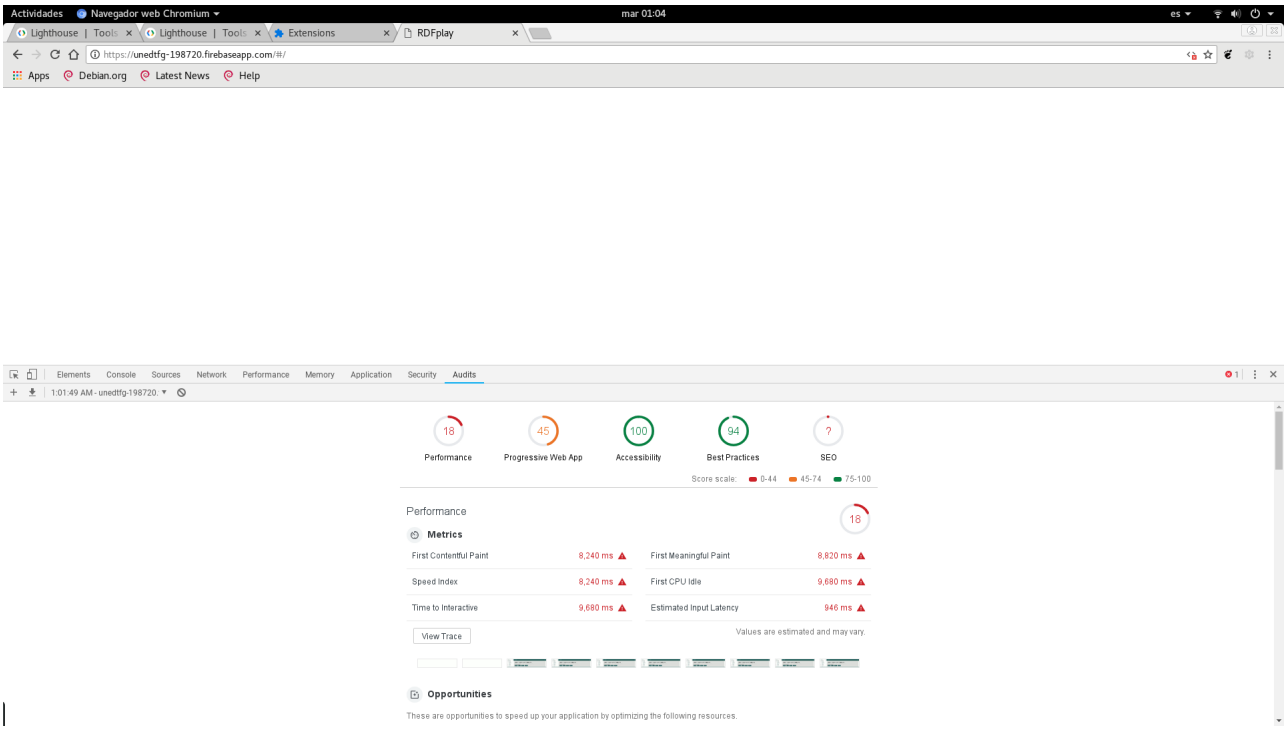


Figura 9.2: Validación de accesibilidad con *Axe* para *Mozilla Firefox*.

Lighthouse proporciona otros indicadores que no son relevantes aquí, puesto que no era el propósito de este proyecto desarrollar la aplicación resultante como una *Progressive Web Application* (PWA). En cuanto al pobre rendimiento mostrado en el indicador correspondiente, este depende de muchos factores: los factores principales son el alojamiento de la aplicación en una plataforma gratuita y el elevado tamaño a descargar debido a las bibliotecas de *Comunica*, que han de incluirse en su totalidad mientras se encuentren en versiones inestables.

9.2. Visualización en dispositivos móviles

La biblioteca de componentes utilizada, *Vuetify*, es *"mobile friendly"*, lo que facilita el desarrollo de una aplicación visualizable en dispositivos móviles. Sin embargo, no basta tan solo con utilizar la biblioteca; determinados componentes como listas tienen implementadas características de elipsis de contenidos que hacen que su visualización en un dispositivo móvil no sea la más adecuada.

Dado que durante todo el proceso de desarrollo la aplicación ha estado desplegada en *Firebase* y por tanto accesible desde Internet, se ha podido revisar y mejorar la visualización en dispositivos móviles de forma incremental e iterativa. El resultado final consiste en una aplicación perfectamente

visualizable en un dispositivo móvil, tal y como se puede comprobar manualmente o a través de herramientas de validación como *Google Mobile Friendly Test*[7] (ver figura 9.3).

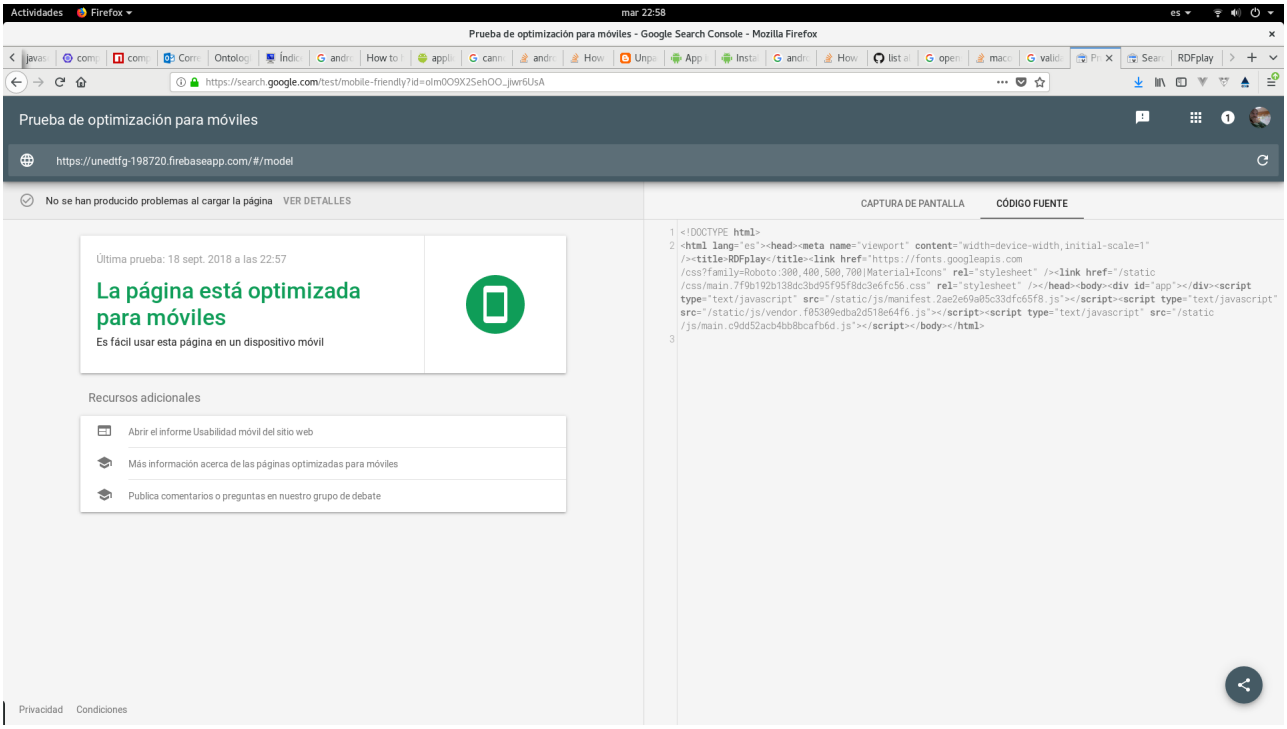


Figura 9.3: Resultados de *Google Mobile Friendly Test*.

9.3. Calidad

Como medida fundamental de calidad del código fuente se ha elegido la complejidad ciclomática, un indicador que arroja datos sobre la futura facilidad de mantenimiento del mismo.

La biblioteca *Eslint*, que realiza análisis estático de código en cada compilación y es utilizada en el proyecto, permite configurar distintas reglas. Además de las incorporadas por defecto[8], se ha configurado la regla de complejidad con un valor máximo de 10. Como se puede comprobar en la figura 9.4), no hay ninguna regla invalidante a la hora de compilar y ejecutar la aplicación en el momento de finalización de su desarrollo (lo que implica que el valor máximo de complejidad no se sobrepasa en ningún método o función).

[7] https://search.google.com/test/mobile-friendly

[8] Ver reglas por defecto en https://eslint.org/docs/rules/

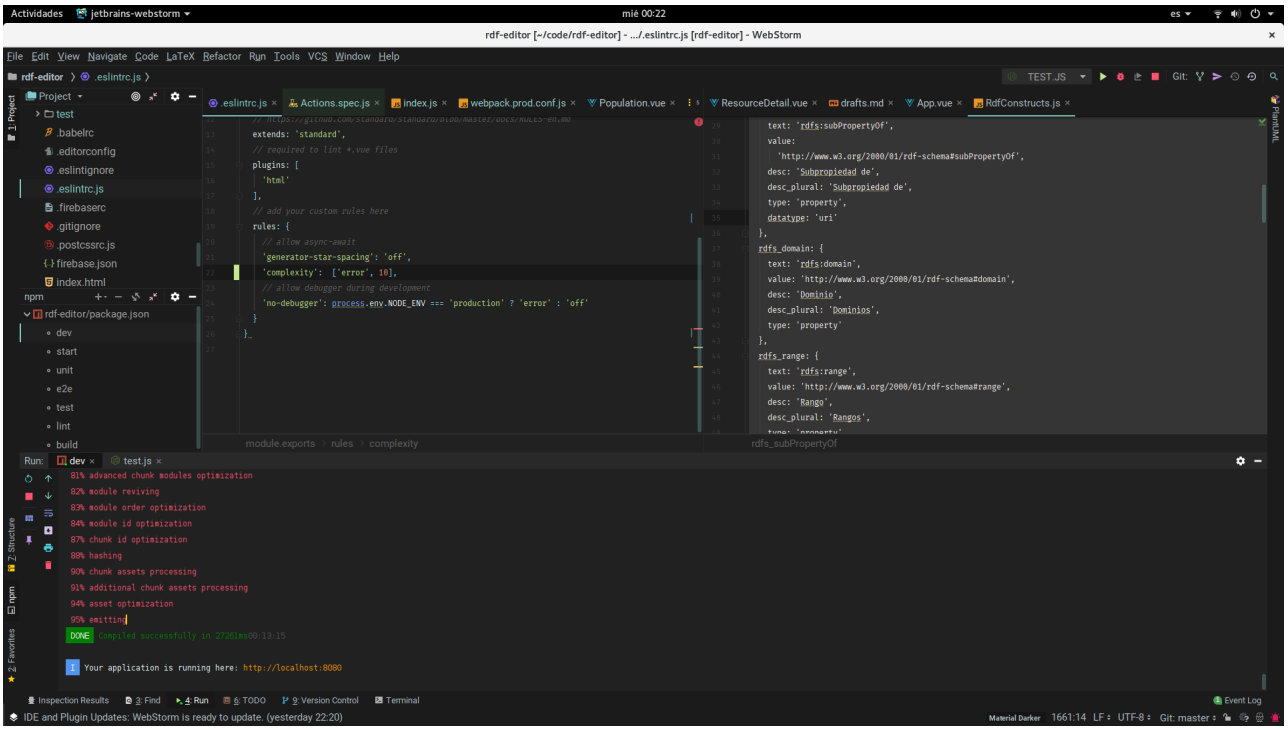


Figura 9.4: Compilación y análisis estático con *Eslint*

9.4. Rendimiento

Es importante destacar que la aplicación desarrollada, al ejecutarse completamente en el navegador del usuario, puede presentar importantes limitaciones de rendimiento debido fundamentalmente al uso de memoria *Random Access Memory* (RAM).

Conjuntos de datos del orden de *Kilo bytes* son manejables, pero las pruebas de carga de un fichero TTL de 22 *Mega bytes* revelan un consumo final de RAM de en torno a 500 MB, así como la **inviabilidad de lanzar consultas sin limitar los resultados a dicho grafo**.

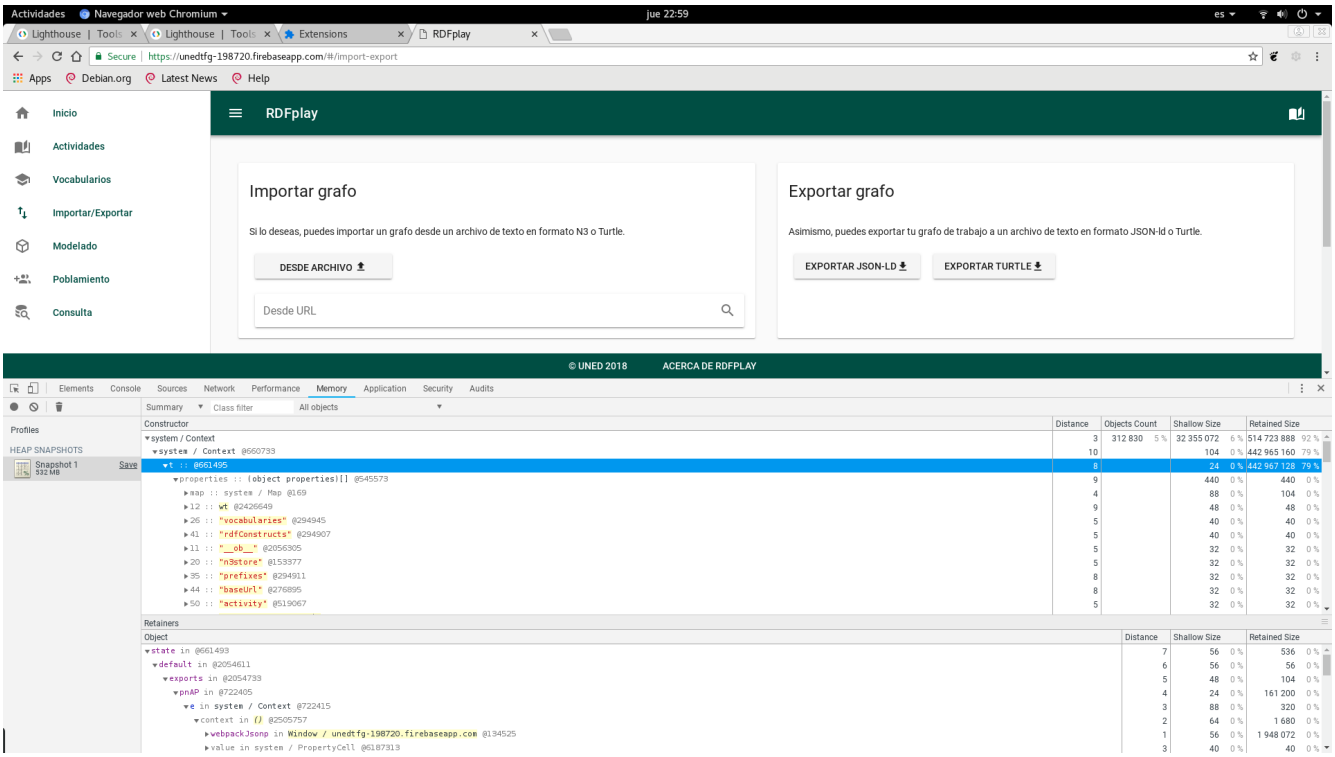


Figura 9.5: Memoria utilizada para un conjunto de datos de 22 MB.

Sin embargo, dado que el objetivo del presente proyecto no es competir con bases de datos RDF y *backends* SPARQL que contengan millones de tripletas sino facilitar el aprendizaje sobre grafos sencillos, estas limitaciones no se consideran problemáticas siempre y cuando se tengan en cuenta a la hora de trabajar con distintos conjuntos de datos. Además, siempre es posible explorar otros caminos de mejora del rendimiento, como el lanzamiento de un ***web worker***[9] distinto para las bibliotecas de consulta SPARQL (ver sección 10.2).

[9]Proceso *Javascript* corriendo en *background*.

Capítulo 10

Conclusiones y líneas de trabajo futuras

10.1. Conclusiones

Alimentar la Web con datos estructurados y contextualizados con metadatos no ha de ser una labor exclusiva de entornos académicos y universitarios. Si bien son estas organizaciones las que han modelado dominios de información y liberado conjuntos de datos de forma masiva a lo largo de los últimos años, es necesario extender esta responsabilidad a otros organismos y entidades y, por qué no, a potenciales usuarios o clientes particulares. Esta es la forma de conseguir que el relato inicial de Berners-Lee[2] en su primera definición de la Web Semántica sea una realidad en nuestra Sociedad.

Aún queda mucho trabajo por hacer. Organizaciones públicas y privadas han de concienciarse de la enorme importancia de poner a disposición de la Sociedad en general datos estructurados en formatos reutilizables. Solo de esta forma crecerá el interés en su tratamiento y explotación, así como en el desarrollo de agentes inteligentes que sean capaces de procesarlos para generar nuevos resultados de análisis. No hay más que recurrir a los datos del *Estudio de Caracterización del Sector Infomediario en España*[37] proporcionados por el *Observatorio Nacional de las Telecomunicaciones y de la Sociedad de la Información* (ONTSI), que recogen que el 70 % de las empresas encuestadas accede a datos no estructurados, frente al 69 % que lo hace a datos en formatos abiertos (si bien no se especifica si se trata de datos a nivel de datos abiertos enlazados o *linked open data*).

La comunidad de tecnólogos y desarrolladores de aplicaciones para la Web Semántica debe trabajar por la democratización de esta tecnología. Con los niveles de complejidad conceptual y de uso actuales, su difusión entre el público general es todo un reto. En este sentido, las grandes revoluciones surgidas en torno al mundo del desarrollo de *frontend* web en general (y *Javascript* en particular) y el nivel de madurez alcanzado en estos momentos por las tecnologías de la Web Semántica, favorecen la llegada del momento más propicio en los últimos veinte años para lograr un impulso real. Aún son pocas las comunidades de desarrolladores trabajando en ofrecer servicios semánticos a través de

Javascript, pero la existencia de un *"community group"*[5] que aglutina esfuerzos para aunar a dichas comunidades en un sólo proyecto es una noticia prometedora de cara al futuro de esta tecnología.

Como pequeña contribución a estos esfuerzos de difusión y democratización, se ha desarrollado el presente proyecto, que representa la culminación de meses de trabajo pero también el inicio de la vida de un producto. Se sientan las bases para obtener una herramienta académica que permita acercar las tecnologías de la Web Semántica a cualquier no iniciado. En esta primera versión ya se cuenta con una aplicación funcional, y el estado de las bibliotecas que integra y las posibilidades de RDF y SPARQL sugieren unas expectativas de evolución lo suficientemente optimistas como para continuar su desarrollo y no abandonar su mantenimiento.

A partir de este punto, la hoja de ruta que se propone para el producto desarrollado en el presente proyecto es clara: en un primer curso académico, comenzar a utilizarlo a modo de prueba piloto en asignaturas técnicas relacionadas con la Web Semántica a través de actividades opcionales (para mejorar la nota total del alumno, por ejemplo) e ir evaluando su idoneidad de cara a su manejo en el entorno para el que fue pensado. Paralelamente, la aplicación puede seguir siendo mejorada y perfeccionada, en caso de aparición de defectos inesperados.

Un claro compromiso por parte del autor de este proyecto debe ser para con la Comunidad Educativa, y especialmente para con los grupos de trabajo como el *RDF Javascript Libraries Community Group* por su inestimable ayuda prestada. Por tanto, parece razonable trabajar en una futura traducción al idioma inglés para permitir su publicación como repositorio de código abierto y revertir el desarrollo a la Comunidad.

Este capítulo no estaría completo sin valorar las competencias tecnológicas y organizativas que el desarrollo del proyecto han permitido adquirir a su autor: profundización en el modelado RDF, las arquitecturas de aplicaciones de la Web Semántica, los motores de consultas SPARQL y el desarrollo en marcos de trabajo *Javascript* completos y bien estructurados como *Vue.js* (una inversión, esta última, más que rentable en términos de utilización de la tecnología).

Tomar decisiones entre altas dosis de incertidumbre no es sencillo, y menos aún cuando la oferta tecnológica es numerosa y variada (como es el caso, por ejemplo, de las bibliotecas de componentes gráficos para *Vue*). Además, la complejidad inicial del desarrollo en *Javascript* no presenta una curva de aprendizaje suave, especialmente en la configuración del entorno necesario: empaquetadores y *"transpilers"* con sus módulos asociados, existencia de diversos estándares conviviendo entre distintos módulos que complican su integración, etc.

Por otra parte, los trabajos previos de implementación de RDF en *Javascript* también son numerosos y confusos (prueba de ello es la reciente noticia de Junio de este mismo año, 2018[5], indicando la unificación de las diferentes implementaciones existentes en una única implementación). Afortunadamente, la ayuda y los consejos de los propios desarrolladores, ya con experiencia, ha permitido enfocar el proyecto con las estrategias adecuadas.

En resumidas cuentas, se ha tratado de obtener un producto apto para la formación académica en tecnologías de la Web Semántica, cumpliendo razonablemente con los objetivos iniciales y abriendo un camino al desarrollo de una herramienta mucho más completa que pueda aprovecharse de las tecnologías emergentes en el área del *frontend* web. Se han utilizado bibliotecas y módulos estándares dentro de la situación actual, lo que permitirá una mayor facilidad de mantenimiento en el futuro y pretende abrir el camino a evoluciones más ambiciosas. A fecha de finalización de la presente memoria, ya se están manteniendo conversaciones con el Director del proyecto para el futuro uso de la aplicación en la actividad docente, lo que permitirá, sin duda, obtener *feedback* de cara a su futura mejora. El tablero *Kanban* del proyecto y el propio repositorio en *Github* están disponibles desde este momento para incorporar nuevas tarjetas y peticiones de evolución, lo que convierte a *RDFplay* en un proyecto con expectativas de futuro razonablemente prometedoras. Parece apropiado finalizar esta memoria como se empezó, citando al padre de la Web:

> *La web es más una creación social que técnica. La diseñé para un efecto social -para ayudar a las personas que trabajan juntas- y no como un juguete técnico. El objetivo último de la web es apoyar y mejorar nuestra existencia en la telaraña mundial. Nos agrupamos en familias, asociaciones y empresas. Desarrollamos la confianza a grandes distancias, y la desconfianza a la vuelta de la esquina. ([38])*
>
> - Tim Berners-Lee.

10.2. Líneas de trabajo futuras

A modo casi de epílogo del presente proyecto y a su vez de prólogo para futuras mejoras y nuevas funcionalidades, se proponen las siguientes líneas de trabajo:

- Integrar la primera versión estable (una vez publicada) de los distintos módulos de la plataforma *Comunica* y optimizar el uso de dependencias.
- Integrar las futuras versiones del marco de trabajo de componentes *Vuetify* y utilizar sus novedades, como por ejemplo una estructura jerárquica (en árbol) para representar clases y propiedades.
- Integrar o desarrollar consultas SPARQL con *Comunica* a grafos en RDF/XML[a].
- Incrementar la cobertura de pruebas unitarias y extremo a extremo.
- Reorientar las operaciones de consulta a través del módulo *actor-init-sparql-rdfjs* (ver sección 7.1.3.1).
- Funcionalidad de importar y exportar en formato *JSON-raw* (una lista de objetos con ternas).
- Facilitar gestión de prefijos en toda la aplicación.
- Dar soporte a literales y URIs de recursos en todas las facilidades de modelado así como a tipos de datos XML, manteniendo la consistencia con rangos y dominios.
- Enriquecer y refactorizar la funcionalidad de modelado.
- Ofrecer soporte para múltiples grafos.
- Revisar la carga de términos básicos y estudiar la viabilidad de carga en línea desde una ontología estándar.
- Actualizar las bibliotecas más importantes de la aplicación y refactorizar en caso de haber sufrido cambios estructurales, como es el caso de *Babel*[b].
- Lanzar todas las operaciones relacionadas con las bibliotecas de comunica desde un *web worker* independiente de la interfaz y dedicado exclusivamente a estas tareas.
- Traducir de la aplicación al idioma inglés y puesta a disposición de las distintas comunidades de desarrolladores y usuarios.

[a] https://github.com/comunica/comunica/issues/140
[b] Ver https://babeljs.io/docs/en/v7-migration

Bibliografía

[1] Communication from the Comission to the European Parliament, the Council, the European Economic and Social Committee and the Committee of the Regions on the Digital Education Action Plan, 2018.

[2] Tim Berners-Lee. The Semantic Web. *Scientific American, Inc.*, 2001.

[3] Tim Berners-Lee. Linked Data. *W3C*, 2006.

[4] Data Never Sleeps 5.0, 2017. https://www.domo.com/learn/data-never-sleeps-5.

[5] Ruben Verborgh and Thomas Bergwinkl. Rdf Javascript Libraries Community Group, 2018. https://www.w3.org/community/rdfjs/.

[6] Ora Lassila. Resource Description Framework (RDF) Model and Syntax Specification. W3C recommendation, W3C, February 1999. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[7] Andy Seaborne and Eric Prud'hommeaux. SPARQL query language for RDF. W3C recommendation, W3C, January 2008. http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

[8] Semstats. 2018.

[9] Apache Software Foundation. Apache Jena. https://jena.apache.org/.

[10] Colin Evans and Jamie Taylor. *Programming the Semantic Web*. 2009.

[11] Varios. Comparison of RDFJS libraries.

[12] Raphaël Benitte, Sacha Greif, and Michael Rambeau. StateOfJS, 2017. https://2017.stateofjs.com/.

[13] TechMagic. Reactjs vs Angular4 vs Vue.js, What to choose in 2018, 2018. https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d.

[14] JD Isaacks. *Get Programming With Javascript Next*. 2018.

[15] Stephen Grider. ES6 Javascript: The Complete Developer's Guide. https://www.udemy.com/javascript-es6-tutorial/learn/v4/overview.

[16] Stone River eLearning. Working with Javascript Streams, 2018. https://www.udemy.com/working-with-javascript-streams/learn/v4/overview.

[17] Maximilian Schwarzmüller. Vue JS 2 - The Complete Guide, 2018. https://www.udemy.com/vuejs-2-the-complete-guide/learn/v4/overview.

[18] Anthony Gore. The Ultimate Vue JS 2 Developers Course, 2018. https://www.udemy.com/vuejs-2-essentials/learn/v4/overview.

[19] Paul. Halliday. *Vue.js 2 Design Patterns and Best Practices*. 2018.

[20] President Draws Planning Moral: Recalls Army Days to Show Value of Preparedness in Time of Crisis by William M. Blair, Quote Page 4, Column 3, New York. (ProQuest), 1957.

[21] Winston W. Royce. Managing the development of large software systems. *IEEE WESCON*, 1970.

[22] Craig Larman. *Applying UML and patterns*. 2005.

[23] The Standish Group. Chaos Report, 2015.

[24] Scott W. Ambler. The Non-Existent Software Crisis: Debunking the Chaos Report. *Dr. Dobb's*, 2014.

[25] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change*. 2005.

[26] Thomas Bergwinkl, Michael Luggen, elf Pavlik, Blake Regalia, Piero Savastano, and Ruben Verborgh. Interface Specification: RDF Representation, 2017. http://rdf.js.org/.

[27] Ruben Taelman and Joachim Van Herwegen. Comunica query engine platform, 2018. https://github.com/comunica/comunica.

[28] Marcus Hammamberg and Juakim Sunden. *Kanban in Action*. 2014.

[29] Martin Fowler. *UML Distilled*. 2004.

[30] Alistair Cockburn. *Writing Effective Use Cases*. 2001.

[31] Rational. Rational Unified Process: Best Practices for Software Development Teams. 1998.

[32] Bob DuCharme. *Learning SPARQL*. 2013.

[33] Liyang Yu. *A Developer's Guide to the Semantic Web*. 2015.

[34] What is Vuex?, 2018. https://vuex.vuejs.org/.

[35] Javier Garzás. Dos razones por las que desarrollar software no es lo mismo que fabricar coches o construir casas, 2011. http://www.javiergarzas.com/2011/02/diferencias-software-fabricacion-tradicional-1.html.

[36] Kent Beck. *Test-Driven Development*. 2003.

[37] Ricardo Vázquez Martínez and María Martínez López. Caracterización del Sector Infomediario 2016. Resumen ejecutivo. Technical report, Observatorio Nacional de las Telecomunicaciones y de la Sociedad de la Información, 2017.

[38] Tim Berners-Lee. Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web, 2000. https://thisiswhatgoodlookslike.com/2012/06/10/gartner-survey-shows-why-projects-fail/.

Acrónimos

API Application Programming Interface. 7, 16, 19, 20, 28, 96, 109

CORS Cross-Origin Resource Sharing. 64, 69, 71, 76, 102

CRUD Create Read Update Delete. 45

CSS Cascade Style Sheets. 97

CSV Comma Separated Values. 44

DOM Document Object Model. 24, 25, 106, 109

ECMA European Computer Manufacturers Association. 93

HTML HyperText Markup Language. 5, 17, 24, 25, 44, 62, 91, 97, 109

HTTP Hypertext Transfer Protocol. 96, 99

IDE Integrated Development Environment. 29

JSF Java Server Faces. 23

JSON JavaScript Object Notation. 44, 118

JSON-LD JavaScript Object Notation - Linked Data. 72, 73, 89

JSP Java Server Pages. 23

JSX Javascript Syntax eXtension. 24

MIT Massachusetts Institute of Technology. 21, 24, 25

MVC Model View Controller. 24

MVP Minimum Viable Project. 14, 108

MVVM Model View View-Model. 25

N3 Notation 3. 19, 20, 68, 70, 81, 83, 84, 89

ONTSI Observatorio Nacional de las Telecomunicaciones y de la Sociedad de la Información. 115

OWL Web Ontology Language. 45

PaaS Platform as a Service. 98

PDF Portable Document Format. 107

POC Proof Of Concept. 42

PWA Progressive Web Application. 111

RAM Random Access Memory. 113

RDF Resource Description Framework. 2–9, 12, 14, 16, 17, 19–23, 28, 35, 44, 45, 79–84, 88, 108, 114, 116–118

REST Representational State Transfer. 19, 96

RUP Rational Unified Process. 43

SFC Single File Component. 97

SKOS Simple Knowledge Organization System. 45

SPARQL SPARQL Protocol And Query Language. 2, 3, 8, 9, 12, 13, 15, 16, 18–23, 28, 37, 42, 44, 45, 74–76, 80, 83–86, 88, 91, 108, 114, 116, 118

SQL Structured Query Language. 8, 12, 80

TDD Test Driven Development. 93

TIC Tecnologías de la Información y las Comunicaciones. 1

UML Unified Modeling Language. 42, 90, 107

UNED Universidad Nacional de Educación a Distancia. 1, 44, 45

URI Uniform Resource Identifier. 6, 7, 47–49, 53–55, 59, 60, 100, 118

URL Uniform Resource Locator. 45, 46, 63, 64, 66, 68–71, 74–76, 88, 91

W3C World Wide Web Consortium. 1, 4, 5, 20, 27, 28, 81–84

WCAG Web Content Accesibility Guidelines 2.0. 109

XML eXtensible Markup Language. 3, 19, 20, 81–84, 101, 109, 118

Anexo A

Manual de usuario

El manual de usuario forma parte de la propia aplicación web, pudiendo ser accedido a través del enlace "Ayuda" ubicado en el pie de página de la misma.