UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Pop Framework

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo Fin de Grado

Pop Framework

Autor: Juan Luis Fragoso del Rey

Tutor: Álvaro Prieto Ramos

ÍNDICE GENERAL DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

RESUMEN

En la sociedad de hoy en día, la filosofía Open Data está cobrando más fuerza que nunca. Esta filosofía, que cada vez cuenta con más apoyos, consiste en poner a disposición de la ciudadanía la información manejada por las administraciones públicas.

Esta información es publicada por las administraciones públicas en portales para que terceros puedan obtener el máximo beneficio mediante su reutilización (por ejemplo, con la construcción de aplicaciones que utilicen dichos datos). Por tanto, se puede afirmar que uno de los principales objetivos de la publicación de estos datos es generar beneficio económico.

Pero las administraciones públicas manejan una gran cantidad de datos, que pueden provenir del turismo, urbanismo, transporte, sanidad, … Ante tal cantidad de información, en muchas ocasiones los responsables de los portales no tienen medios ni herramientas para discernir qué datos tienen mayor probabilidad de ser reutilizados.

Otro problema es establecer los indicadores que permitan aclarar que datos son más utilizados por la comunidad. Para ello, podemos basarnos en los proyectos software de código abierto. Es difícil conocer el número de proyectos de código abierto que utilizan ciertos datos, pero estos proyectos tienen su código fuente fácilmente localizable en repositorios de código abierto. Estos repositorios ofrecen la posibilidad de buscar proyectos en función del contenido de su código fuente, por lo que realizando una búsqueda en dichos repositorios se podría establecer los indicadores necesarios.

Con el desarrollo de este trabajo, se pretende construir un sistema que solucione esto último, es decir, que permita a las administraciones públicas discernir qué tipo de datos son más propensos a su reutilización y, como consecuencia, con mayor capacidad para generar ingresos. Para poder conseguirlo, se construirá una aplicación que categorizará automáticamente una serie de registros obtenidos de repositorios de código abierto (con el fin de establecer los indicadores de utilización de los datos) y, a partir de ciertas

preferencias que el usuario podrá establecer en la aplicación, se realizará una estimación acerca de qué tipo de datos son más rentables.

Esta idea nace de un proyecto anterior desarrollado por el grupo de investigación *Quercus* de la *Universidad de Extremadura*, en el que se realiza esta misión dentro de un proyecto más amplio. Por tanto, el objetivo de este trabajo es mejorar en la medida de lo posible el proceso de toma de decisiones sobre qué datos publicar mediante la construcción de la aplicación mencionada.

1. INTRODUCCIÓN

En el siglo XXI, muchas ciudades del mundo están inmersas en un proceso de transformación a ciudad inteligente o Smart City en inglés. Para que una ciudad sea considerada como Smart City, debe "ser capaz de responder adecuadamente a las necesidades básicas de instituciones, empresas, y de los propios habitantes, tanto en el plano económico, como en los aspectos operativos, sociales y ambientales" [1]. Es decir, estas transformaciones van encaminadas a reducir costes y mejorar la eficiencia de las administraciones y al mismo tiempo mejorar la calidad de vida de los ciudadanos y la relación con las empresas que alberga.

Poniendo especial atención en la reducción de costes, ya que es un tema muy presente en nuestra vida diaria debido a la profunda crisis sufrida durante los últimos años y donde las ciudades han sido una de las entidades que más la han sufrido, los responsables de estas cuentan actualmente con un presupuesto normalmente limitado, por lo que antes de realizar cualquier inversión estudian su viabilidad. Por tanto, una ciudad que aspira a convertirse en inteligente debe encontrar el camino entre el equilibrio del presupuesto y que las inversiones realizadas retornen en el futuro algún tipo de ingreso.

Una de las principales características de las Smart Cities es que manejan una gran cantidad de datos de los distintos ámbitos de la ciudad, como puede ser transporte, urbanismo, salud, turismo, etc. Esta información, explotada de manera adecuada, puede suponer una vía muy importante de crecimiento económico. Pero la clave para que esta información genere ingresos es su reutilización, por lo que el CDO (Chief Data Officer en inglés, Director de datos en castellano), figura nacida con el auge de este tipo de ciudades y responsable de portales de datos abiertos, debe elegir cuidadosamente que datos se publicarán en el portal con vistas a que sean reutilizados lo máximo posible.

A la hora de elegir qué datos publicar en el portal, los CDOs deben ser capaces de discernir qué información podría llegar a suponer una mayor fuente de ingresos para aquellos que la utilicen en un momento determinado

y cuáles no, por lo que se hace esencial el uso de una herramienta que permita distinguir que datos son más adecuados para su publicación. Pero los CDOs no disponen de ninguna herramienta que solucione este problema, por lo que surge la necesidad de crear una.

Es en este punto donde nace el proyecto Pop Framework, desarrollado por el grupo de investigación *Quercus* de la Universidad de Extremadura. Inicialmente, este proyecto consiste en una herramienta disponible para los CDOs, de manera que puedan elegir apropiadamente que conjuntos de datos publicar en su portal buscando que la inversión realizada sea lo más óptima posible, lo que aportará un mayor beneficio económico a todo aquel que utilice estos datos. Este proyecto se divide inicialmente en las siguientes partes:

1. Una serie de procesos de búsqueda de conjuntos de datos en portales de datos abiertos.
2. Un proceso de categorización de datasets
3. Una herramienta de búsqueda de uso de los conjuntos de datos abiertos en repositorios de proyectos software de código abierto.
4. Unos procesos de estimación de indicadores del uso de los conjuntos de datos abiertos a partir de la información de los proyectos software que los utilizan.
5. Una herramienta de decisión multicriterio fundamentada en el Proceso Analítico Jerárquico[2] que recomendará a los CDOs de los portales de datos abiertos los conjuntos de datos más apropiados a publicar en función de los indicadores de uso de los conjuntos de datos abiertos en repositorios de proyectos software de código abierto y las valoraciones que hagan los CDOs de estos indicadores.

A modo de resumen, en la siguiente imagen se puede observar el conjunto de herramientas que componen el proyecto original:

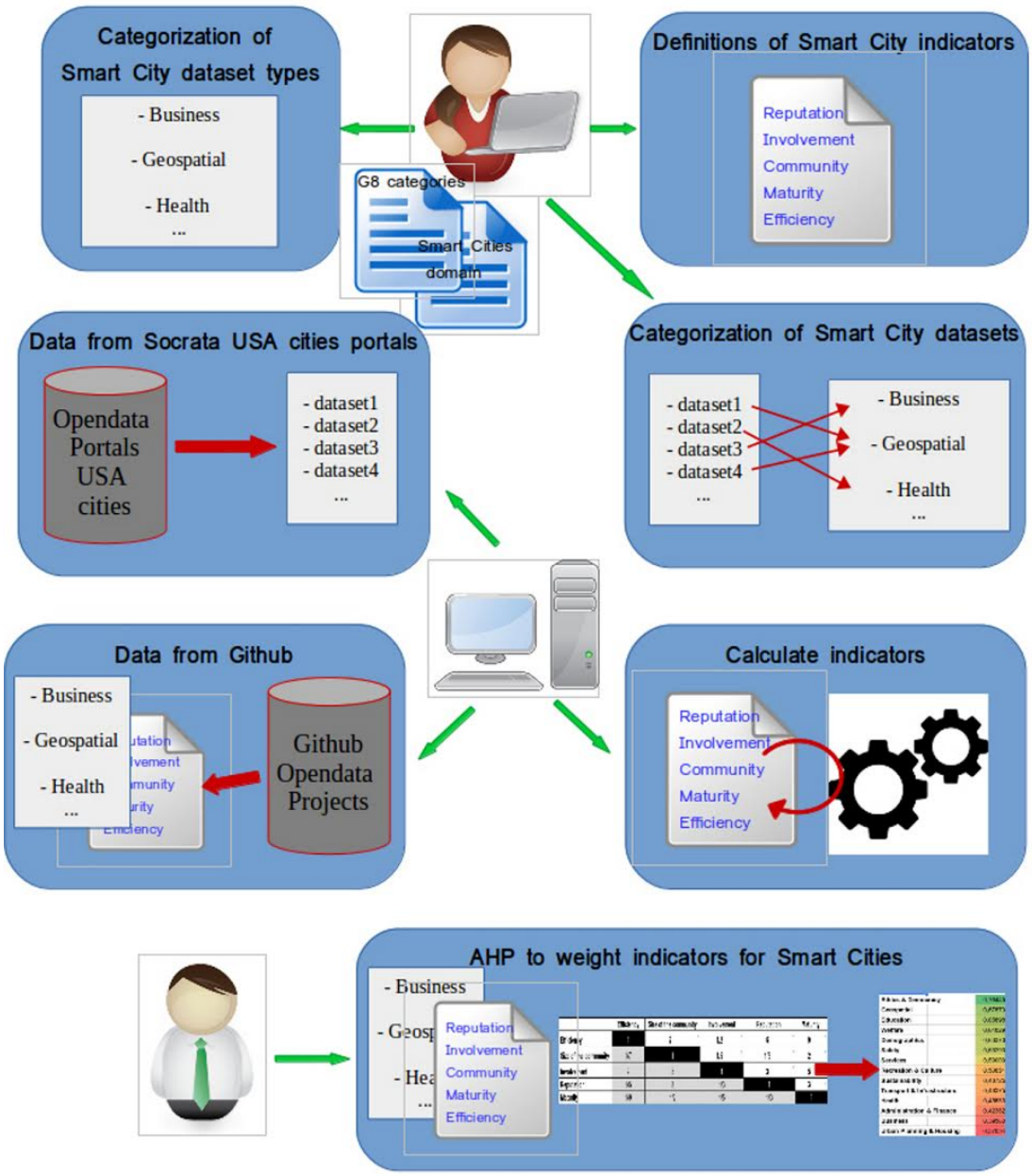

Ilustración 1: Resumen proyecto original

Partiendo de este punto, se encuentra susceptible de mejora las siguientes partes:

- Para estimar los indicadores del uso de conjuntos de datos abiertos, el proyecto previo a este trabajo necesita que un conjunto de expertos realice la categorización del conjunto de datasets registrados en la base de datos como resultado de los procesos de la parte 1 mencionada anteriormente. Por tanto, se busca realizar una

categorización automática de estos conjuntos de datos a partir de su significado semántico en diferentes categorías.

- La herramienta de decisión multicriterio en particular, y toda la versión previa en general se puede considerar poco amigable. De esta forma, se va a realizar una aplicación web con el objetivo de que resulte más sencilla de usar por parte de los CDOs y consiga una mejor experiencia de usuario.

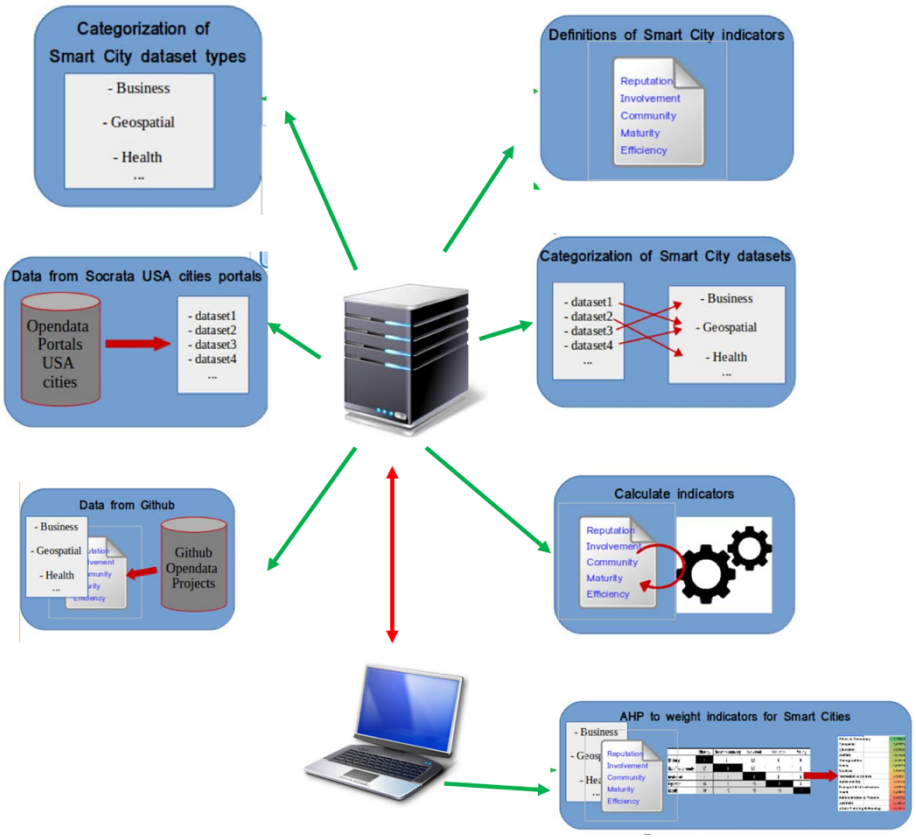

Ilustración 2: mejoras proyecto original

Por lo tanto, en este trabajo se desarrollará las herramientas necesarias para conseguir los objetivos mencionados anteriormente.

1.1. Objetivos

Partiendo de las mejoras a realizar explicadas en el apartado anterior, se establecen los siguientes objetivos:

- **Categorizar automáticamente** los diferentes conjuntos de datos almacenados utilizando algoritmos semánticos.

- **Desarrollar una aplicación web** que permita realizar la toma de decisiones multicriterio a través de una interfaz amigable y con una buena experiencia de usuario. De esto objetivo nacen los siguientes subobjetivos:
  - **Desarrollar el backend** de la aplicación web para que se pueda atender las peticiones de todos los usuarios desde el navegador web.
  - **Desarrollar el frontend** de la aplicación web para que los usuarios tengan una buena experiencia de usuario y puedan interactuar con la aplicación.

1.2. ¿Por qué una aplicación web?

Se conoce como aplicación o portal web al tipo de aplicaciones que son accesibles desde un servidor web a través de internet o una intranet a través de un navegador web [3].

Las ventajas que representan las aplicaciones web son:

- Ahorran costes de hardware y software.
- Son fáciles de distribuir y de usar.
- Son fácilmente escalables.
- Tienen una gran facilidad de uso.
- Sólo una versión de la aplicación localizada en el servidor, por lo que es menos propensa a fallos.

Y entre las desventajas encontramos [4]:

- Peor experiencia de usuario que aplicaciones híbridas o nativas.
- Ignora características del dispositivo.
- Ofrece menor seguridad, ya que depende de la seguridad que ofrezca el navegador.

Por tanto, teniendo en cuenta los pros y los contras de usar una aplicación web, y considerando que el público objetivo será personal cualificado con sobrado conocimiento para el despliegue de la aplicación en un navegador web, se opta por este tipo de aplicación.

1.3. Motivación

Los motivos que han propiciado el desarrollo de este trabajo han sido:

- En primer lugar, la posibilidad de trabajar con desarrollo web, tanto en la parte cliente como en la parte servidor, ya que es una parte de la Informática que llama bastante mi atención y, sin tener en cuenta la tecnología a emplear, considero que para mi futuro profesional es un buen aprendizaje.
- En segunda instancia, la posibilidad de trabajar con una temática muy de moda hoy en día como es Open Data y que, a su vez, permite que se puedan solucionar problemas que nuestra sociedad sufre a día de hoy, me ha parecido interesante.

2. PLANIFICACIÓN

La planificación de este trabajo se ha estructurado en las siguientes fases o etapas:

- Fase 1: estudio de las opciones disponibles previas a la realización de este trabajo para establecer las herramientas que permitan realizar la categorización automática de los registros contenidos sobre portales de datos abiertos. Este estudio se muestra en el capítulo 3.
- Fase 2: estudio de las diferentes herramientas y métodos para la construcción de un portal web que permita conseguir una interfaz amigable a la hora de utilizar la aplicación. Este estudio se detalla en el capítulo 3.
- Fase 3: una vez finalizadas las fases 1 y 2, se realiza un análisis de los casos de usos del sistema a construir y se establecen los diferentes requisitos que la aplicación deberá cumplir. El detalle de esta fase se encuentra en el capítulo 4.
- Fase 4: toma de decisiones sobre la arquitectura de la aplicación y herramientas a usar para llevar a cabo la construcción del sistema. Esta fase se encuentra detallada en el capítulo 4.
- Fase 5: construcción de la aplicación a raíz de los resultados obtenidos en la fase 4. Como en cualquier aplicación, a la hora de su construcción han surgido problemas que se han debido solventar y se detallan en el capítulo 6.
- Fase 6: documentación del trabajo. Debido a que esta fase se realiza durante todo el desarrollo del trabajo, ha convivido con el resto de fases y, por tanto, se ha realizado mientras se realizaban las demás.

Con todas las fases del trabajo establecidas, se define la tabla 1 en la que se indica el número de horas planificadas para cada fase y el número de horas necesitadas para llevarlas a cabo.

Tabla 1: Horas de cada fase del proyecto

Fase	Horas planificadas	Horas reales
Fase 1	15	40
Fase 2	15	30
Fase 3	30	50
Fase 4	40	50
Fase 5	160	250
Fase 6	30	90

3. ESTADO DEL ARTE

En este capítulo se muestra el análisis de las distintas herramientas existentes previo al desarrollo de este trabajo y en las cuales se ha basado la implementación del mismo. Para el estudio de estas herramientas, se ha dividido el análisis en 2 partes. Por un lado, la categorización automática de los datasets y, por otro lado, la creación del portal web.

3.1. Categorización automática

Para la categorización de los registros de datos abiertos de forma automática, existen unas herramientas conocidas como tesauros. Un tesauro es una herramienta documentalista la cual se puede definir desde el punto de vista de su función o desde el punto de vista de su estructura:

- Por su función, es una herramienta de control terminológico utilizada para transponer a un lenguaje más estricto el idioma natural utilizado por los documentos y por los indizadores.
- Por su estructura, es un vocabulario controlado y dinámico de términos que tienen entre ellos relaciones semánticas y genéricas y que se aplica a un dominio particular del conocimiento.

El tesauro probablemente más conocido desde el punto de vista semántico es WordNet [5]. Wordnet es una extensa base de datos léxica en inglés, en la que los sustantivos, verbos, adjetivos y adverbios se agrupan en conjuntos de sinónimos denominados *synsets*. Cada *synset* representa un concepto distinto. Los *synsets* están relacionados entre sí por medio de un número de correlación que indica el parecido semántico que tienen dichos términos, de manera que a través de estos indicadores de correlación podemos saber si los términos son equivalentes semánticamente o no. En definitiva, la unión de todos los *synsets* representa un grafo en el que el camino entre los nodos contiene la medida de similitud entre los mismos.

Para realizar el cálculo de similitud entre distintos términos existen varios algoritmos, pero sin duda dos de los más utilizados y conocidos son Wu & Palmer y Leacock-Chodorow:

- **Wu & Palmer [6]:** este algoritmo devuelve un valor que indica la similitud entre dos términos en función de la profundidad de los términos en el grafo de synsets que compone Wordnet. A continuación se muestra el pseudocódigo el algoritmo WUP para el cálculo de similitud entre dos términos:

```
Algorythm similarityWithWUP (s1, s2)
Begin
        Init WUP Instance
        Synsets_s1 = set of synsets s1
        Synsets_s2 = set of synsets s2
        For each Synsets_s1
                For each Synsets_s2
                        Score = WUP.getSimilarity(Synsets_s1, Synsets_s2)
                        If score is greather tan maxScore
                                maxScore = score
        if maxScore is not valid
                maxScore = 0
end
```

Ilustración 3: algoritmo Wu & Palmer

Como se puede apreciar, por cada término es necesario obtener cada uno de los synsets que lo componen y, posteriormente, realizar el cálculo de similitud entre los synsets de cada término. La medida de similitud definitiva será la mayor de entre todas las comparaciones realizadas.

- **Leacock-Chodorow [7]:** devuelve un valor que establece la similitud de ambos términos en función del camino más corto que une los synsets en el grafo de Wordnet. En la siguiente imagen se puede apreciar el pseudocódigo del algoritmo Leacock-Chodorow, en el que se especifica cómo se realiza el cálculo de similitud entre dos términos:

```
Algorythm similarityWithLeC (s1, s2)
Begin
        Init LeC Instance
        Synsets_s1 = set of synsets s1
        Synsets_s2 = set of synsets s2
        For each Synsets_s1
                For each Synsets_s2
                        Score = LeC.getSimilarity(Synsets_s1, Synsets_s2)
                        If score is greather tan maxScore
                                maxScore = score
        if maxScore is not valid
                maxScore = 0
end
```

Ilustración 4: algoritmo Leacock-Chodorow

La metodología seguida es la misma que con el algoritmo Wu & Palmer (WUP). La diferencia radica en que el cálculo de similitud entre los diferentes synsets se realiza con el algoritmo Leacock-Chodorow.

Para utilizar Wordnet con dichos algoritmos se encuentra la siguiente librería de Google [8] que proporciona la implementación de los algoritmos indicados anteriormente.

Los dos algoritmos mostrados (tanto el de Wu & Palmer como el Leacock-Chodorow) realizan la categorización semántica entre dos palabras. A la hora de categorizar frases enteras, como es el caso que ocupará en este trabajo, este proceso debe repetirse para cada una de las palabras que componen las frases, es decir, se debería comparar cada palabra de primera frase con cada palabra de la segunda frase. Todo esto, teniendo en cuenta que por cada par de palabras comparadas se obtienen todos los synsets que la componen y, a su vez, se comparan cada uno de estos synsets entre sí para obtener el comprobar cuales tienen mayor grado de similitud, supondría un impacto enorme en el rendimiento de la aplicación, ya que debería almacenar en memoria mucha información (a primera vista, los resultados de las comparaciones parciales de cada synset, así como los resultados parciales de cada palabra) además de requerir un tiempo de ejecución el elevado, ya que el número de comparaciones a realizar sería bastante grande.

Fuera del ámbito del Wordnet, encontramos el algoritmo de **Jaro-Winkler [9]**, muy utilizado en informática y estadística, que permite medir la distancia entre dos strings de manera que indica su grado de similitud. Este algoritmo arroja un valor entre 0 y 1, siendo 0 el valor que indica que son completamente diferentes y 1 semánticamente iguales. El pseudocódigo del algoritmo de Jaro Winkler es:

```
Algorythm similarityJaroWinkler (Sting s1, String s2)
Begin
        Init JaroWinkler instance
        Score = JaroWinkler.calculateSimiliarty(s1,s2)
        If score is not valid
                Score = 0
        return Score
End
```

Ilustración 5: Algoritmo Jaro-Winkler

Como se puede apreciar, para calcular la similitud entre dos palabras con este algoritmo únicamente es necesario comparar las dos palabras entre sí, no es necesario realizar ningún proceso extra como era necesario con los algoritmos de Wu & Palmer y Leacock-Chodorow. Esto permite que el algoritmo de Jaro-Winkler proporcione mejores resultados en cuanto a manejo de memoria y tiempo de ejecución.

Al igual que con los demás algoritmos, para calcular la similitud entre dos frases, serían necesario comparar cada palabra de una frase con cada palabra de la otra y, posteriormente, se podría realizar la media aritmética de todas las comparaciones.

Para utilizar el algoritmo de Jaro-Winkler se encuentra el siguiente repositorio de Github [10] que proporciona la implementación del mismo.

En la siguiente tabla se muestra una comparativa entre los algoritmos mencionados anteriormente:

Tabla 2: comparativa algoritmos categorización

Algoritmo	Grado de categorización	Complejidad	Inclusión en backend	Tiempo de ejecución	Uso de memoria
Wu & Palmer	Alto	Media	Fácil	Alto	Allto
Leacock-Chodorow	Alto	Media	Fácil	Alto	Alto
Jaro-Winkler	Alto	Baja	Fácil	Bajo	Medio

Como se puede apreciar en la tabla, los algoritmos Wu & Palmer y Leacock-Chodorow presentan los mismos valores en los parámetros escogidos para la selección del algoritmo de categorización. El algoritmo de Jaro-Winkler presenta un grado de categorización alto y una fácil inclusión en el backend de la aplicación, al igual que los otros dos algoritmos. En cambio, la complejidad algorítmica que implica su inclusión en dicho backend en baja, comparada con los algoritmos de Wu & Palmer y Leacock-Chodorow, que requieren de gran optimización para conseguir un buen grado de categorización. Esto último adquiere mayor importancia si tenemos en cuenta los tiempos de ejecución, donde el algoritmo de Jaro-Winkler tiene una rápida respuesta (tiempo de respuesta bajo), mientras que los algoritmos Wu & Palmer y Leacock-Chodorow tardan mucho más en categorizar la misma cantidad de datos (tiempo de respuesta alto). Por último, teniendo en cuenta el uso de memoria por parte de cada algoritmo, los algoritmos Wu & Palmer hacen un alto uso de memoria, mientras que el algorimot Jaro-Winkler podríamos decir que realiza un uso medio de la misma, por los motivos explicados anteriormente.

3.2. Aplicación Web

En primer lugar, es necesario aclarar los tipos de aplicaciones que existen para el fin que se busca en este trabajo. Según encontramos en [4], podemos distinguir los siguientes tipos de aplicaciones:

- **Aplicaciones nativas:** es una aplicación desarrollada y optimizada para el sistema operativo del fabricante. Se adapta al 100% a las características del dispositivo, ofreciendo una mejor experiencia de usuario. El desarrollo de este tipo de aplicaciones supone un mayor coste, puesto que realizar una aplicación multiplataforma supone desarrollos distintos.
- **Aplicaciones web:** Consiste en el tipo de aplicaciones más económico. Se desarrolla una única aplicación la cual es posible ejecutar en varios dispositivos. Utilizando la técnica *"responsive web design"* se consigue que la aplicación pueda ser abierta en todos los dispositivos, adaptándose a su tamaño de pantalla ofreciendo una correcta visualización.
- **Aplicaciones híbridas:** Es una mezcla de las dos anteriores, es decir, extrae los beneficios de una aplicación web adaptándose al dispositivo como una aplicación nativa. Permite desarrollar la aplicación con los estándares utilizados en desarrollo web(HTML5, CSS3, Javascript, …) pero utilizando las funciones del dispositivo en el que se ejecuta (cámara, GPS, contactos,…). Para poder compilar la aplicación en distintos sistemas operativos se puede hacer uso de plugins como Cordova o Phonegap, Xamarin, …

Dentro de las aplicaciones web, podemos encontrar los siguientes tipos de aplicaciones según se establece en [11]:

- **Aplicación web estática:** aplicaciones enfocadas a mostrar una información permanente, donde el navegante se limita a obtener dicha información sin que pueda interactuar con el sitio web. Están construidas principalmente con hipervínculos o enlaces entre las páginas que conformas la aplicación web. Este tipo de aplicaciones

son incapaces de soportar gestores de base de datos e información cambiante (foros, consultas online, …).

- **Aplicación web dinámica:** son aplicaciones en las que la información presentada se genera a partir de una petición del usuario. En las aplicaciones dinámicas la información aparece tras una solicitud realizada por el usuario. Permiten almacenar y hacer actualizaciones de la información contenida, así como modificaciones dinámicas de la estructura y del diseño de dicha información por parte de su propietario.
- **Tienda virtual o comercio electrónico:** es el tipo de aplicaciones pensada para el e-commerce (tienda online). El desarrollo de este tipo de aplicaciones es más complejo que el de las aplicaciones web estáticas o dinámicas, ya que aparte de tener que dar soporte para base de datos y servidor, se debe construir una pasarela de pagos electrónicos a través de tarjetas de créditos, PayPal u otro método de pago. Además, este tipo de aplicaciones también cuentan con un gestor de contenidos o CMS que permite subir los productos, actualizarlos o eliminarlos.
- **Portal Web App:** aplicación web optimizada y adaptable a cualquier dispositivo, es decir, una aplicación que se puede tanto desde un navegador en un ordenador como desde un smartphone, independientemente del sistema operativo que utilice. Esta optimización es posible gracias a que utilizan HTML5 y CSS3.
- **Aplicación web animada:** aplicaciones que utilizan la tecnología FLASH para presentar contenidos de forma animada. También permiten diseños más creativos y modernos. El inconveniente de este tipo de aplicaciones es que para temas de posicionamiento web y SEO, la tecnología FLASH no es la más idónea debido a que los buscadores no son capaces de leer correctamente la información.
- **Aplicación web con gestor de contenidos:** en este tipo de aplicaciones web la información se actualiza constantemente, por lo tanto, es necesario instalar un gestor de contenidos o CMS, que

permite a los administradores y a los editores realizar los cambios y actualizaciones de dicha información.

3.3. Conclusión

Después de realizar el estudio tanto de las posibles herramientas a utilizar como de los tipos de aplicaciones web (mostrado en la tabla 2), se aprecia que los tres algoritmos son fáciles de incluir en el backend y tienen un grado de categorización alto. La diferencia radica a la hora de realizar la categorización, como en el tiempo de respuesta de la misma. El algoritmo de Jaro-Winkler tiene un tiempo de respuesta mínimo en comparación con los algoritmos de Wu & Palmer y Leacock-Chodorow. A la hora de realizar el algoritmo de categorización, resulta más complejo para estos dos últimos algoritmos que para el de Jaro-Winkler. Por tanto, se utilizará este algoritmo para realizar la categorización de datasets.

Por otro lado, para la construcción de la aplicación se opta por una aplicación web de tipo portal web, ya que el uso de varias secciones es factible según las características de la aplicación a construir. De igual forma, en un principio de descarte el uso de algún CMS, ya que en un principio los contenidos de la web permanecerán estáticos con excepción de la herramienta de la herramienta para cálculos multicriterio. Así mismo, se descarte realizar un diseño *responsive*, ya que en un principio la aplicación será destinada a equipos de escritorio debido a los cálculos complejos que se deben realizar. La decisión de incluir un CMS, así como un diseño *responsive* será tenida en cuenta en versiones posteriores.

4. ANÁLISIS Y DISEÑO

En esta sección se definirán en primer lugar los requisitos funcionales y los requisitos no funcionales. A continuación, se explicarán los casos de uso de la aplicación y posteriormente se comentará las decisiones tomadas en cuanto a la solución propuesta. En última instancia, se detalla el motivo de uso de una API REST y la solución utilizada para su implementación.

4.1. Requisitos

Para asegurar que se cumplen los objetivos propuestos con el desarrollo de este trabajo, se han propuestos una serie de requisitos que definen el comportamiento del sistema. Los requisitos software definen el comportamiento del sistema y en este apartado se dividen en **requisitos funcionales**, que son declaraciones de los servicios que proveerá la aplicación y de la manera que esta reaccionará a entradas particulares (a veces definen lo que el sistema no debe hacer), y en **requisitos no funcionales**, que especifican restricciones en el diseño o la implementación, es decir, son cualidades que el sistema debe cumplir.

4.1.1. Requisitos funcionales

En la siguiente tabla se especifican los requisitos funcionales que debe cumplir el sistema y que, como se ha explicado anteriormente, define su comportamiento:

Tabla 3: Requisitos funcionales

Identificador	Descripción
RF1	Permitir al usuario establecer sus preferencias en base a una serie de características predefinidas.
RF2	Permitir al usuario consultar que datos serían deseables en base a sus preferencias.
RF3	Permitir al usuario recategorizar los registros de la base de datos.
RF4	El sistema guardará la recategorización lanzada por el usuario en base de datos, estando disponible para las sucesivas ejecuciones.

RF5	Ofrecer información detallada sobre el funcionamiento de la aplicación.

4.1.2. Requisitos no funcionales

En la tabla mostrada a continuación se presentan los requisitos no funcionales, que como bien se ha indicado antes, especifican restricciones sobre la implementación o el diseño del sistema.

Tabla 4: Requisitos No Funcioanles

Identificador	Descripción
RNF1	El sistema presentará una interfaz gráfica amigable.
RNF2	El sistema realizará las llamadas a la API REST asíncronamente.
RNF3	El sistema debe estar completamente construido en en inglés para facilitar su uso a nivel internacional.

4.2. Casos de uso

En el contexto de la ingeniería del software, , un caso de uso es una secuencia de acciones que se desarrollarán entre un sistema y sus actores en respuesta a un evento que es lanzado por un actor principal sobre el propio sistema. Por tanto, podemos definir como actor a la persona o entidad que interacciona con el sistema.

4.2.1. Definición de actores

En el caso de la aplicación que se propone desarrollar, nos encontramos con un único rol o actor, que será el CDO que ejecutará la aplicación. Dado que se trata de una herramienta que no requiere ningún tipo de sistema de autenticación ni datos previos para acceder a la misma, no se distingue entre varios tipos de usuarios, únicamente existirá un usuario que lanza la aplicación, por tanto, este será el actor principal y único.

4.2.2. Diagrama y descripción de los casos de uso

En esta sección se expondrá el diagrama de casos de uso de la aplicación junto con sus respectivas tablas explicativas de los mismos.

Los diagramas de casos de uso representan de forma gráfica la funcionalidad del sistema, a través del comportamiento y la interacción de los usuarios con el mismo.

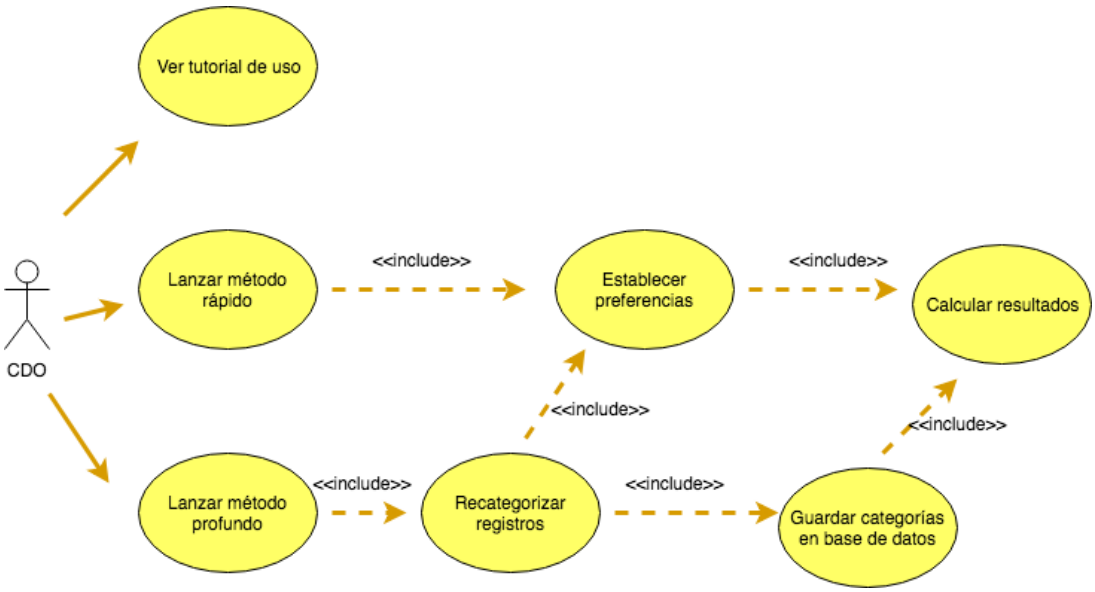

Ilustración 6: Casos de uso del sistema

Tabla 5: Caso de uso "Ver tutorial aplicación

Nombre	**Ver tutorial de uso**	
Descripción	Permite al usuario ver información detallada sobre el funcionamiento de la aplicación en caso de no ser experto en la misma y en especial en AHP como sistema de toma de decisiones	
Precondición	La información debe estar accesible para el usuario	
Secuencia normal	Paso 1	Acceder a la página de inicio de la aplicación
	Paso 2	El usuario pulsa sobre el botón de información
Postcondición	La información queda disponible para que el usuario pueda consultarla	
Excepciones		

Tabla 6: Caso de uso "Lanzar método rápido"

Nombre	**Lanzar método rápido**	
Descripción	Permite al usuario ejecutar la herramienta de consulta para saber sobre qué datos publicar	
Precondición	La herramienta debe estar disponible para el usuario. Se parte de un conjunto de datasets categorizados previamente.	
Secuencia normal	Paso 1	Acceder a la página de inicio de la aplicación
	Paso 2	El usuario pulsa sobre el botón de herramienta
	Paso 3	El usuario pulsa sobre el botón “método rápido”
	Paso 4	La aplicación lee los indicadores de la base de datos.
	Paso 5	Cuando los indicadores se han leído, se muestra al usuario la tabla para que indique sus preferencias.
	Paso 6	El usuario indica sus preferencias
	Paso 7	Cuando el usuario ha establecido sus preferencias, pulsa sobre el botón calcular resultados.
	Paso 8	Se calculan los resultados
	Paso 9	Se muestran los resultados al usuario
Postcondición	El usuario puede ver los resultados según sus preferencias	
Excepciones		

Tabla 7: Caso de uso "Lanzar método profundo

Nombre	**Lanzar método profundo**	
Descripción	Permite al usuario ejecutar la herramienta de consulta para saber sobre qué datos publicar	
Precondición	La herramienta debe estar disponible para el usuario	
Secuencia normal	Paso 1	Acceder a la página de inicio de la aplicación
	Paso 2	El usuario pulsa sobre el botón de herramienta
	Paso 3	El usuario pulsa sobre el botón “método profundo”
	Paso 4	La aplicación lanza la recategorización de los registros de la base de datos
	Paso 5	Cuando la recategorización ha terminado, la aplicación la guarda en la base de datos.
	Paso 6	La aplicación lee los indicadores de la base de datos.
	Paso 7	Cuando la aplicación ha leído los indicadores de base de datos, muestra al usuario una tabla para que establezca sus preferencias.
	Paso 8	El usuario establece sus preferencias.
	Paso 9	Cuando el usuario ha establecido sus preferencias, pulsa sobre el botón calcular resultados.
	Paso 10	La aplicación realiza el cálculo de resultados.
	Paso 11	Cuando la aplicación ha terminado con los cálculos, los muestra al usuario.
Postcondición	El usuario puede ver los resultados según sus preferencias	
Excepciones		

Tabla 8: Caso de uso "Recategorizar registros"

Nombre	**Recategorizar registros**	
Descripción	Permite al usuario categorizar nuevamente todos los datasets de la base de datos en diferentes categorías	
Precondición	El usuario debe haber lanzado previamente la funcionalidad “método profundo”	
Secuencia normal	Paso 1	Se accede a base de datos
	Paso 2	Se leen todos los registros de base de dato
	Paso 3	Por cada registro de base de datos se compara con las diferentes categorías su parecido semántico
	Paso 4	La categoría que contenga mayor parecido semántico el tema del registro en cuestión se establece como su categoría
Postcondición	Se obtiene la nueva categoría para el registro	
Excepciones		

Tabla 9: Caso del uso "Guardar categorías en base de datos"

Nombre	**Guardar categorías en base de datos**	
Descripción	Tras ejecutar la recategorización, se guardan las categorías en base de datos.	
Precondición	Se debe haber categorizado previamente un registro de la tabla de datasets de base de datos.	
Secuencia normal	Paso 1	Se obtiene la categoría resultante tras realizar la categorización del registro.
	Paso 2	Se guarda en base de datos la categoría resultante tras la categorización
	Paso 3	Se repite el Paso 1 y 2 hasta categorizar todos los registros
Postcondición	Los registros quedan actualizados con las nuevas categorías en base de datos	
Excepciones		

Tabla 10: Caso de uso "Establecer preferencias"

Nombre	**Establecer preferencias**	
Descripción	Permite al usuario establecer preferencias sobre los datos que desea publicar en su portal	
Precondición	El usuario debe haber lanzado previamente la funcionalidad “método profundo” o “método rápido	
Secuencia normal	Paso 1	Se accede a base de datos.
	Paso 2	Se leen los indicadores desde base de datos.
	Paso 3	Se calculan los datos necesarios a partir de los indicadores para poder realizar los cálculos para decisiones multicriterio.
	Paso 4	Cuando los cálculos están listos, se muestra al usuario una tabla para que pueda establecer sus preferencias.
	Paso 5	Cada vez que el usuario establece una preferencia, se comprueba que los datos tienen una consistencia válida.
Postcondición	El usuario ha establecido sus preferencias respecto a los datos que desea publicar	
Excepciones		

Tabla 11: Caso del uso "Calcular resultados"

Nombre	**Calcular resultados**	
Descripción	Permite al usuario visualizar los resultados para publicar datos en base a sus preferencias	
Precondición	El usuario deberá establecer sus preferencias previamente	
Secuencia normal	Paso 1	Se comprueba la consistencia de las prefencias.
	Paso 2	El usuario pulsa el botón para calcular resultados.
	Paso 3	En base a las preferencias, se realiza los cálculos multicriterios mediante el método AHP
	Paso 4	Se muestran al usuario los resultados a través de una tabla
Postcondición	El usuario puede visualizar los resultados	

Excepciones	

4.3. Solución propuesta

En esta sección se detalla algunos tipos de aplicaciones web existentes en la actualidad, así como arquitecturas para este tipo de aplicaciones. Este análisis se tomará como referencia para el desarrollo del sistema.

4.3.1. Tipos de aplicaciones web

Podemos encontrar distintos tipos de aplicaciones web, donde las más importantes y destacables a día de hoy son:

Partiendo de los tipos de aplicaciones webs analizados en el apartado 3.2, podemos extraer las siguientes conclusiones:

- **Aplicación web estática:** este tipo de aplicaciones no son compatibles con el sistema que se pretende construir, ya que no soporta información cambiante, es decir, la información mostrada siempre es la misma y no varía tras alguna acción del usuario. Además, no tiene soporte para sistema gestores de base de datos. Por tanto, no es una opción a tener en cuenta.
- **Aplicaciones web dinámicas**: en estas aplicaciones el contenido es generado automáticamente tras una acción del usuario. Para la creación de este tipo de páginas, además de la existencia de una base de datos, por lo que este tipo de aplicaciones puede cumplir los requisitos necesarios para la aplicación que se pretende construir.
- **Tienda online para aplicaciones web:** el punto fuerte de los e-commerce es la implementación de una pasarela de pagos que permita compras online, así como una correcta gestión del stock en base de datos. Las aplicaciones de este tipo no son válidas para el sistema que se desea construir, ya que no es necesario una pasarela de pago ni gestión alguna de stock. Por tanto, tiene más requisitos de los necesarios.
- **Portal web app:** los portales web cuentan con diversos apartados categorías o secciones como pueden ser chats, foros, correo

electrónico, un buscador, zona de acceso con registro, etc. Requieren del uso de base de datos y pueden utilizar un sistema CMS, pero su uso no es requisito indispensable. Dadas estas característica, esta opción puede llegar a ser viable para la construcción del sistema.

- **Aplicación web animada:** la principal característica de una aplicación web animada es la animación de sus elementos. Debido a que el sistema que se pretende construir no necesita mostrar información animada, este tipo de aplicaciones no son válidas para la construcción del sistema.
- **Aplicación web con "Gestor de contenidos":** este tipo de aplicaciones no es apropiada para nuestro caso ya que la información, a pesar de ser cambiante, se hace en base a unos cálculos, no a datos obtenidos de una plataforma como puede ser un CMS.

A continuación, y a modo de resumen, en la tabla 12 se muestra la comparación de cada tipo de aplicación aplicada a los requisitos del sistema que se pretende construir.

Tabla 12: Comparativa aplicaciones web

Tipo de aplicación web	Cumple los requisitos
Aplicación web estática	NO
Aplicación web dinámica	SI
Tienda online	NO
Portal web	SI
Aplicación web animada	NO
Aplicación web con CMS	NO

Como se puede apreciar, únicamente 2 tipos de aplicaciones cumple los requisitos exigidos, que son las aplicaciones web dinámicas y los portales web. La diferencia entre estos dos tipos de aplicaciones radica en que los portales web tienen un diseño "*responsive*", es decir, permiten adaptarse a todos los dispositivos independientemente del sistema operativo utilizado. Dado que no es objetivo de este trabajo realizar este tipo de diseño para el sistema, se optará por implementar una aplicación web dinámica, dejando su conversión a un portal web para una versión posterior.

4.3.2. Arquitectura

En este apartado se explicará la arquitectura que tiene una aplicación web y, posteriormente, las diversas arquitecturas que se pueden aplicar en la construcción de una aplicación web.

4.3.2.1. Arquitectura de una aplicación web

Una aplicación web [12] es proporcionada por un servidor web y utilizada por usuarios que se conectan desde cualquier punto vía clientes web (navegadores). La arquitectura de un sitio web tiene tres componentes principales:

- Un servidor web.
- Una conexión de red.
- Uno o más clientes.

El servidor web distribuye páginas de información formateada a los clientes que las solicitan. Las peticiones son hechas a través de una conexión de red, y para ello se usa el protocolo HTTP. Una vez que se solicita esta petición mediante el protocolo HTTP y la recibe el servidor web, éste localiza la página web en su sistema de archivos y la envía de vuelta al navegador que la solicitó.

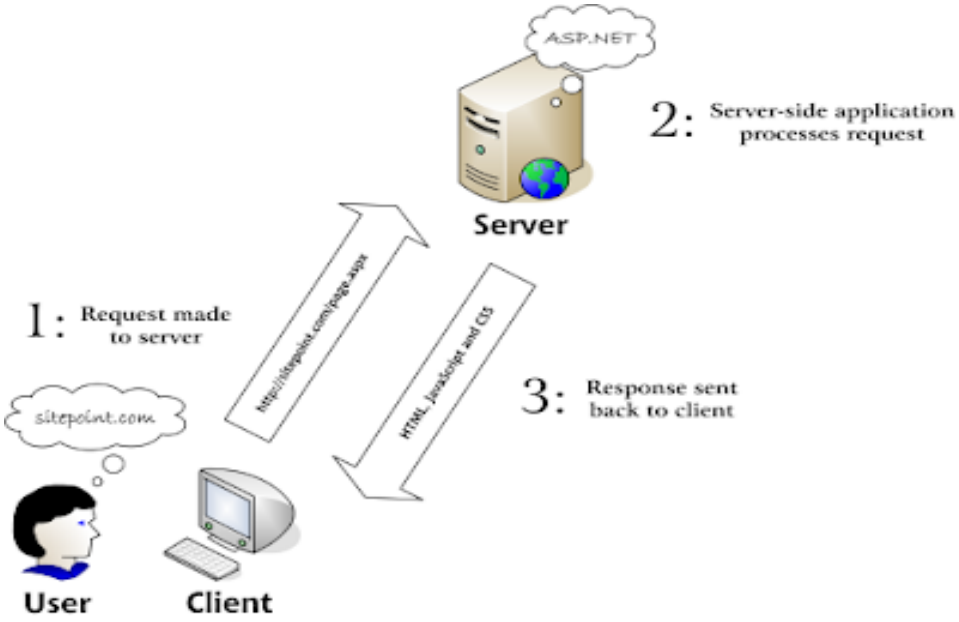

Ilustración 7: Arquitectura aplicación web

Fuente : http://henry-e-estrada.blogspot.com.es/2010/11/client-side-vs-server-side.html

Las aplicaciones web están basadas en el modelo cliente/servidor, que gestionan servidores web y que utilizan como interfaz páginas web.

Las páginas web son el componente principal de una aplicación web. Los navegadores piden páginas (almacenas o creadas dinámicamente) con información a los servidores web. En algunos ambientes de desarrollo de aplicaciones web las páginas contienen código HTML y scripts dinámicos que son ejecutados por el servidor antes de entregar la página.

Una vez es entregada la página, la conexión entre el navegador y el servidor web se rompe, es decir, la lógica de negocio en el servidor solamente se activa por la ejecución de scripts de las páginas solicitadas por el navegador (en el servidor, no en el cliente). Cuando el navegador ejecuta un script en el cliente, éste no tiene acceso directo a los recursos del servidor.

En la arquitectura que se pretende desarrollar en Pop Framework se busca algo diferente, ejecutando parte de la lógica de negocio en el cliente, descargando de trabajo al servidor, y como se explicará en el punto 4.3.2.3.

4.3.2.2. Arquitectura SPA

Dentro del desarrollo de aplicaciones web hay una tendencia importante a las denominadas SPAs (Single Page Applications). Uno de los principales objetivos es conseguir una mejor experiencia de usuario, mejorando los tiempos de espera y la latencia entre vistas.

Con esta arquitectura, la aplicación se envía una vez al navegador y la página no se recarga durante su uso, es decir, no vuelve a realizar una petición al servidor. Este tipo de arquitectura permite realizar cualquier aplicación tradicional de escritorio vía web ya que el tiempo de respuesta es mucho más rápido que el de una aplicación web tradicional, como se ha mencionado anteriormente.

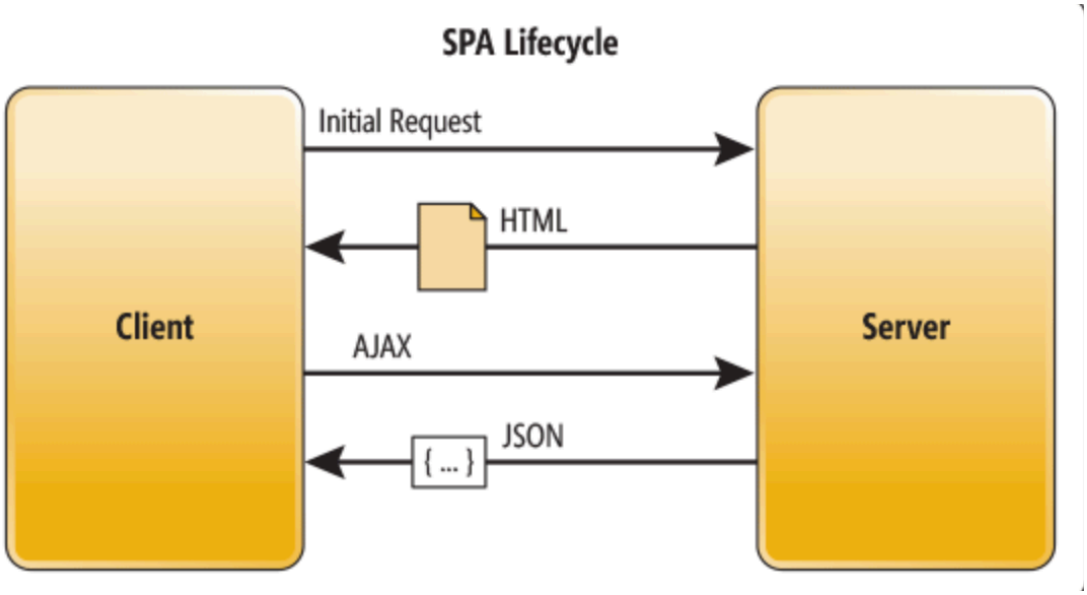

Ilustración 8: Arquitectura SPA

Fuente : https://itblogsogeti.com/2014/06/10/single-page-applications-roberto-bermejo-sogeti/

A continuación, se detallan las principales características de este tipo de arquitectura:

- La mayor parte de la funcionalidad se lleva al cliente. Se podría considerar como un cliente enriquecido que carga desde un servidor web.
- El código en el servidor se usa básicamente para proveer de una API REST a nuestro código cliente usando AJAX.

Esta arquitectura se adapta fácilmente a lo que se pretende construir en Pop Framework, ya que reduce los tiempos de respuesta (que se intuyen inicialmente altos al realizar recategorizaciones que se suponen costosas computacionalmente) y permite delegar parte de la lógica de negocio en el cliente.

4.3.2.3. Arquitectura de Pop Framework

La arquitectura de la aplicación a desarrollar será una SPA donde podemos distinguir las siguientes partes:

- Base de datos: sistema donde se almacena toda la información persistente de la aplicación, concretamente la información correspondiente a los datos obtenidos de los repositorios de Github, así como la vistas creadas (todo esto será explicado en el apartado 4.4). Se encontrará alojada en un servidor que permitirá intercambiar información con el cliente a través de una API REST.
- API REST: servicio web encargado de traspasar información entre la base de datos y el cliente, es decir, proporcionará información al cliente cuando este la solicite y modificará la información en base de datos cuando el cliente realice algún cambio en la misma. Por tanto, permitirá lanzar desde el cliente web el cálculo de indicadores o la categorización automática, actuando de comunicador entre el cliente web y el servidor web. La API REST se encontrará alojada en un servidor que tendrá comunicación directa con el cliente.
- Servidor web: contendrá la aplicación cliente que será devuelta al navegador cuando el usuario la solicite. Puede tratarse del mismo servidor en el que se encuentre alojada la API REST, pero ambas partes están completamente desacopladas. Este servidor web realizará tareas como la categorización automática o el cálculo de indicadores, siempre por petición del cliente web a través de la API REST.
- Cliente: navegador web que realiza la petición de la aplicación web al servidor web y solicita información a la base de datos a través de la API REST o la modifica a través de la misma. Se trata de una SPA

(comentada anteriormente) que proporciona al usuario las herramientas para interactuar con todo el sistema.

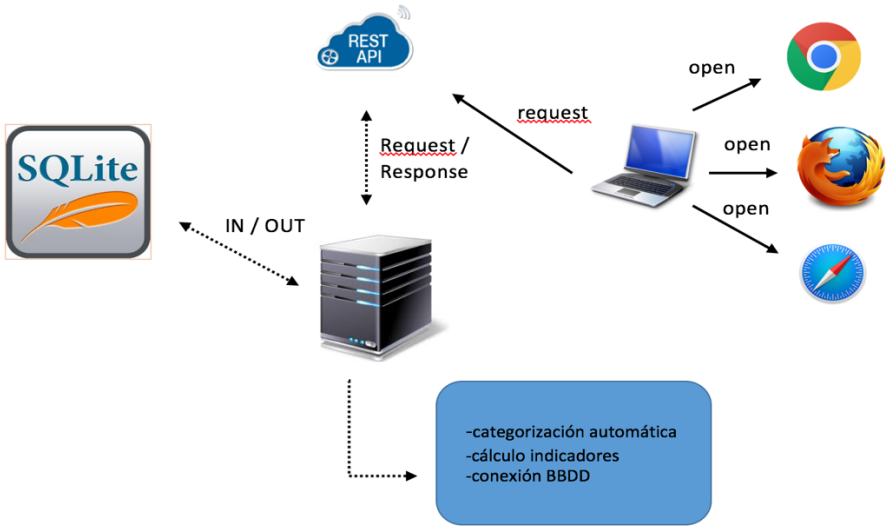

Ilustración 9: arquitectura Pop Framework

4.4. Diseño de la base de datos

La base de datos es heredada del sistema anterior. Se encuentra implementada en SQLite, sistema gestor de base de datos muy versátil y que tiene un alto grado de integración con la mayoría de lenguajes de programación que se usan para desarrollos de API RESTs. Esta base de datos no es una base de datos al uso, sin que es el resultado de llevar a cabo los diferentes procesos de la aplicación antigua, ejecutando una serie de scripts, mediante los cuales extrae información de Github y los almacena en tablas distintas. Cada una de estas tablas por sí mismas carecen de sentido. Su objetivo es la creación de una tabla final, que es la suma de todas las tablas anteriores, en la que se junta la información necesaria de cada una de ellas para poder realizar la categorización y estimaciones que implementará el sistema final.

4.4.1. Measures_repository

La tabla ***"measures_repository"*** almacena una fila para cada repositorio de Github que hace referencia a conjuntos de datos de ciudades de los EE.UU.

con información útil sobre el repositorio. Esta tabla tiene los siguientes campos:

Tabla 13: tabla "measures_repository"

CAMPO	TIPO
repository_id	TEXT
user_id	TEXT
stargazers_count	TEXT
watchers_count	TEXT
language	TEXT
forks_count	TEXT
subscribers_count	TEXT
network_count	TEXT
created_at	TEXT
updated_at	TEXT
pushed_at	TEXT
total_contributors	TEXT
total_contributions	TEXT

4.4.2. Result_search_open_data

Esta tabla, llamada ***"result_search_open_data"***, almacena la información recuperada de Github con una fila para cada referencia desde un repositorio de Github a un conjunto de datos de ciudades de EE.UU. Tiene los siguientes campos:

Tabla 14: tabla "result_search_open_data"

CAMPO	TIPO
identifier	TEXT
theme	TEXT
keyword	TEXT
file_name	TEXT
file_path	TEXT
file_url	TEXT
repository_id	TEXT
repository_name	TEXT
repository_url	TEXT
user_id	TEXT
user_name	TEXT
user_url	TEXT
score	TEXT

4.4.3. Socrata_customers

Esta tabla, llamada ***"socrata_customers"***, almacena datos sobre las organizaciones que usan tecnología Open Data de Socrata y tiene las siguientes características:

Tabla 15: tabla "socrata_customers"

CAMPO	TIPO
id_customer	INTEGER
customer_name	TEXT
type	TEXT
location_name	TEXT

country	TEXT
open_data_site_url	TEXT
api_dcat_url	TEXT
api_data_url	TEXT
longitude	TEXT
latitude	TEXT

4.4.4. Socrata_data

La tabla denominada ***"socrata_data"*** almacena información de cada conjunto de datos que contiene el data.json de los portales de Socrata. Esta tabla tiene las siguientes características:

Tabla 16: tabla "socrata_data"

CAMPO	TIPO
identifier	TEXT
api_url	TEXT
landingPage	TEXT
publisher	TEXT
keyword	TEXT
theme	TEXT
title	TEXT
issued	TEXT
modified	TEXT
description	TEXT
id_customer	INTEGER

4.4.5. Socrata _dcat

La tabla ***"socrata_dcat"***, al igual que la tabla socrata_data, también almacena información de cada conjunto de datos que contiene el data.json de los portales de Socrata. Ambas tablas se complementan. Sus características son:

Tabla 17: tabla "socrata_dcat"

CAMPO	TIPO
identifier	TEXT
webService	TEXT
accessURL	TEXT
feed	TEXT
keyword	TEXT
theme	TEXT
title	TEXT
created	TEXT
releaseDate	TEXT
modified	TEXT
description	TEXT
id_customer	INTEGER

4.4.6. Usa_city_datasets_categorized

La tabla ***"usa_city_datasets_categorized"*** es la tabla final utilizada para realizar el proceso de categorización de los conjuntos de datos. Esta compuesta a partir de datos de las tablas anteriores. Inicialmente la categoría es nula para cada fila. A partir del tema de cada fila se realiza la categorización, pero también contiene el título y la descripción de cada fila para ayudar en aquellos casos en los que se pueda dudar entre una o más

categorías cuando un conjunto de datos no contiene un tema. Los campos de esta tabla son:

Tabla 18: tabla "usa_city_datasets_categorized"

CAMPO	TIPO
IDENTIFIER	TEXT
THEME	TEXT
KEYWORD	TEXT
TITLE	TEXT
DESCRIPTION	TEXT
CATEGORY	TEXT

4.4.7. Vistas

La base de datos contiene 15 vistas creadas a partir de las tablas explicadas anteriormente. Estas vistas son necesarias para poder calcular las estimaciones que realizará el sistema final. A continuación se explica la utilidad de cada una de ellas:

- **Socrata_DATA_CITY_USA:** contiene información útil de cada conjunto de datos que contiene el data.json de los portales Socrata de las ciudades de EE.UU.
- **Socrata_DCAT_CITY_USA:** muetra información útil que contiene el archivo dcat.json de los portales Socrata de las ciudades de EE.UU.
- **USA_CITY_IDS_WITH_THEME_OR_KEYWORD:** muestra el identificador, el tema y la palabra clave de los conjuntos de datos de las ciudades de EE.UU. cuando al menos existe un tema o palabra clave, ya que sin ellos la tarea de categorizar resulta más difícil.

- **Categories_contributions:** muestra, para cada categoría, la suma de total de contribuciones de los distintos que hacen referencia a cada categoría.

- **Categories_contributors:** muestra, para cada categoría, la suma total de contribuyentes de los distintos repositorios que hacen referencia a cada categoría.
- **Categories_madurity_total:** muestra, para cada categoría, la madurez total de los distintos repositorios que hacen referencia a cada categoría.
- **Categories_subscribers:** muestra, para cada categoría, la suma total de suscriptores de los distintos repositorios que hacen referencia a cada categoría.
- **Categories_total_datasets_in:** muestra, para cada categoría, el número de datasets de la categoría, estén referenciados o no en Github.
- **Categories_total_datasets_no_referenced:** muestra, para cada categoría, la suma total de repositorios a los que no se hace referencia en Github.
- **Categories_total_datasets_referenced:** muestra, para cada categoría, la suma total de repositorios a los que se hace referencia en Github.
- **Categories_total_references:** muestra, para cada categoría, el número total de referencias en Github a conjuntos de datos de la categoría.
- **Categories_total_respositories_referencing:** muestra, para cada categoría, el número de repositorios distintos de Github de la categoría.
- **repository_useful_data_for_indicators:** muestra los campos *"repository_id"*, *"total_contributors"*, *"total_contributions"*, *"subscribers_count"*, *"created_at"* y *"update_at"* de los repositorios contenidos en la tabla *measures_repository* cuando existe información sobre la fecha de creación (*"created_at"* no sea ""). Esta vista se usa para crear la mayoría de los indicadores utilizados en esta aplicación.

- **datasets_categorized_referenced_and_distinct_repository_id_referencing:** muestra una fila con el identificador, la categoría y el *"repository_id"* de cada conjunto de datos de las ciudades de EE.UU. a los que hace referencia un repositorio de Github. Aunque un repositorio haga referencia varias veces a un conjunto de datos, solo aparece 1 fila en la vista.
- **Datasets_categorized_not_referenced:** muestra una fila con el identificador, la categoría y el *"repository_id"* con valor NULL de todos los conjuntos de datos de ciudades de EE.UU. a los que no hace referencia un repositorio de Github.

4.5. API REST

Como se ha comentado en el apartado 4.3.2, la aplicación implementará una API REST. En esta sección detallará cuál es su función y los beneficios que se obtiene a la hora de usar este tipo de tecnología.

4.5.1. Definición

Podemos definir un servicio REST como cualquier interfaz entre sistemas que use http para obtener datos o generar operaciones sobre esos datos en todos los los formatos posibles, como XML y JSON, Es una alternativa en auge a otros protocolos estándar de intercambio de intercambio de datos como SOAP (Simple Object Access Protocol), que disponen de una gran capacidad, pero también mucha complejidad. En ocasiones es preferible una solución más sencilla de manipulación de datos como REST.

4.5.2. Características

Los servicios REST [13] ofrecen las siguientes características:

- **Protocolo cliente/servidor sin estado:** cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla. Aunque esto es así, algunas aplicaciones HTTP incorporan memoria caché. Se configura lo que se conoce como protocolo cliente-caché-servidor sin estado: existe la posibilidad de definir algunas respuestas a peticiones HTTP concretas como cacheables, con el objetivo de que el cliente pueda ejecutar en un

futuro la misma respuesta para peticiones idénticas. De todas formas, que exista la posibilidad no significa que sea lo más recomendable.

- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST y la especificación HTTP son cuatro: POST (crear), GET (leer y consultar), PUT(editar) y DELETE (eliminar).
- **Los objetos en REST siempre se manipulan a partir de la URI:** Es la URI y ningún otro elemento el identificador único de cada recurso de ese sistema REST. La URI nos facilita acceder a la información para su modificación o borrado, o, por ejemplo, para compartir su ubicación exacta con terceros.
- **Interfaz uniforme:** para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URI. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información
- **Uso de hipermedios:** hipermedio es un término acuñado por Ted Nelson en 1965 y que es una extensión del concepto de hipertexto. Ese concepto llevado al desarrollo de páginas web es lo que permite que el usuario poder navegar por el conjunto de objetos a través de enlaces HTML. En el caso de una API REST, el concepto de hipermedio explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

4.5.3. Beneficios

1) **Separación entre el cliente y el servidor:** el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
2) **Visibilidad, fiabilidad y escalabilidad:** la separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de

desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.

3) **La API REST siempre es independiente del tipo de plataformas o lenguajes:** la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores PHP, Java, Python o Node.js. Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

4.6. Patrones de diseño

En esta sección se explicará los diferentes patrones de diseño analizados y los motivos por los que se elige un patrón en concreto.

4.6.1. Modelo Vista Presentador (MVP)

El patrón modelo vista presentador (MVP) [14] deriva del patrón modelo vista controlador, y su principal uso es para interfaces de usuario. En este patrón, el presentador asume el papel de intermediario entre la vista y el modelo y realiza toda la lógica de presentación.

Los componentes de este patrón arquitectónico son:

- Modelo: interfaz que define los datos que se mostrarán o no actuando con la interfaz de usuario.
- Presentador: actúa sobre el modelo y sobre la vista. Extrae datos del modelo y se los pasa a la vista para que esta pueda mostrarlos al usuario. En otras palabras, comunica la vista y el modelo.
- Vista: es una interfaz pasiva que muestra datos (del modelo) y se comunica con el presentador para interactuar sobre los datos. Es la representación visual del modelo.

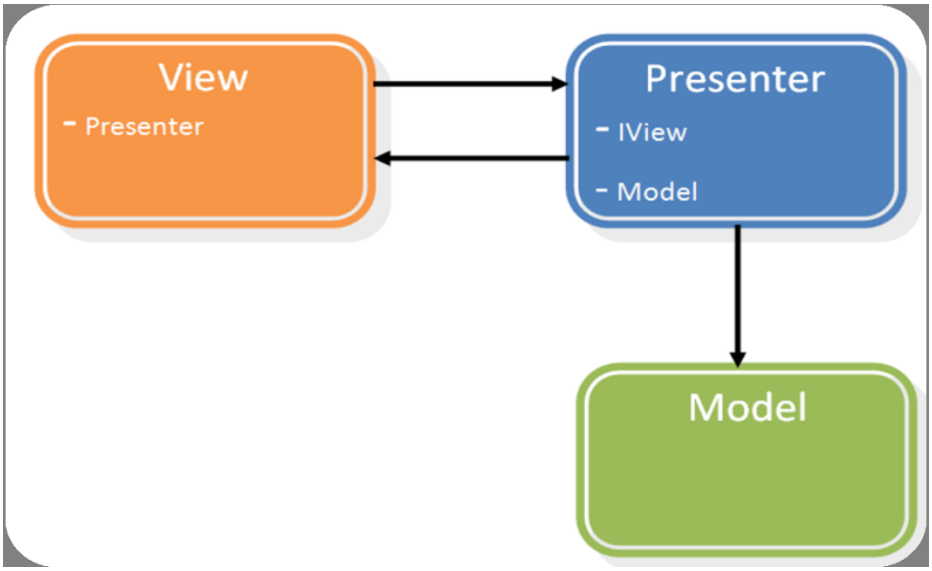

Ilustración 10: modelo vista presentador

Fuente: http://mohamedradwan.com/2011/01/15/mvp-design-pattern-for-web-application-vs-mvc-with-example/

En un extremo, la vista es totalmente pasiva, realiza peticiones al presentador para interactuar con el modelo pero ella por si misma no decide nada, es decir, no contiene lógica de negocio. El presentador recupera los datos del modelo y realiza la lógica de negocio correspondiente. Además, formatea los datos para que puedan ser consumidos por la vista de nuevo.

Este tipo de arquitectura es aplicable a aplicaciones web, por lo que es una opción a tener en cuenta.

4.6.2. Modelo vista vista-modelo (MVVM)

El patrón modelo vista vista-modelo [15] se caracteriza por tratar de desacoplar lo máximo posible la interfaz de usuario de la lógica de negocio. En este tipo de patrones tenemos los siguientes elementos:

- Modelo: representa la capa de datos y/o la lógica de negocio. El modelo contiene la información, pero nunca las acciones o servicios que la manipulan. En ningún caso tiene dependencia alguna con la vista.

- Vista: es la representación gráfica del modelo. Está incomunicado con el modelo, y para interactuar con el utiliza el modelo de la vista.
- Modelo de la vista: es un actor intermediario entre el modelo y la vista, contiene toda la lógica de presentación y se comporta como una abstracción de la interfaz.

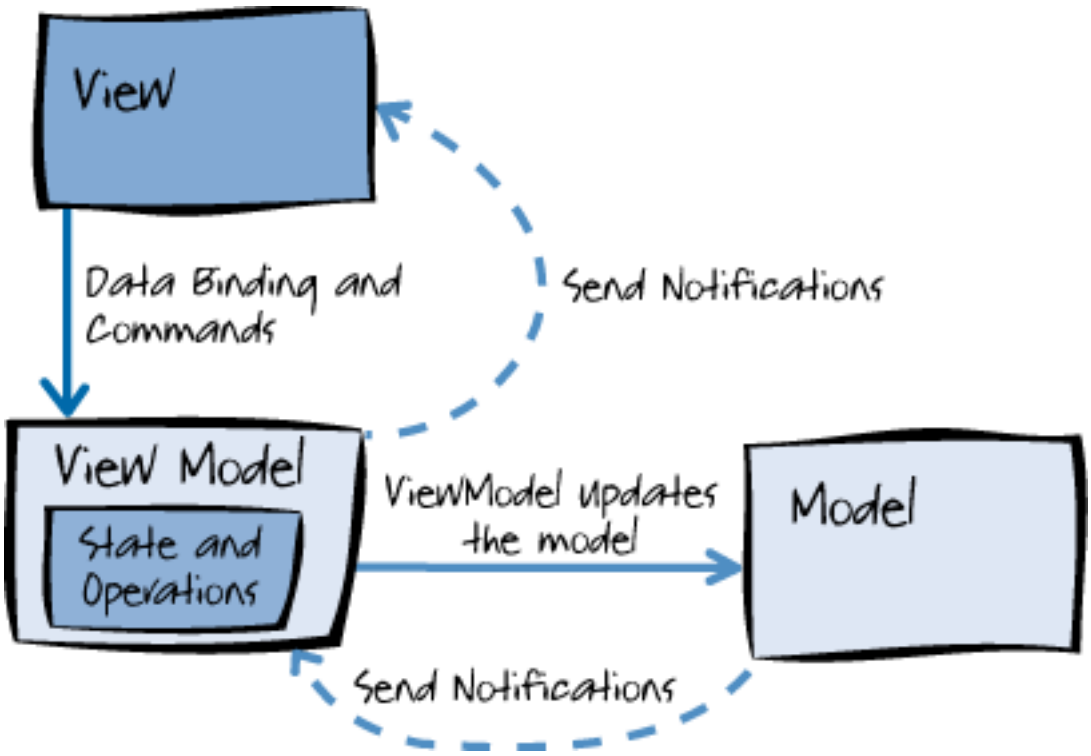

Ilustración 11: modelo vista vista-modelo

Fuente: https://msdn.microsoft.com/en-us/library/hh848246.aspx

La comunicación entre la vista y el modelo de la vista es directa, al igual que ocurre entre el modelo de la vista y el modelo. Al contrario no ocurre lo mismo, ya que para comunicarse el modelo de la vista con la vista, debe hacerlo a través de eventos, al igual que el modelo con el modelo de la vista.

Este patrón también es compatible con arquitecturas web, por lo que también es necesario tenerlo en cuenta.

4.6.3. Modelo vista controlador (MVC)

El patrón modelo vista controlador (MVC) [16] separa los datos y la lógica de negocio de la interfaz de usuario de la aplicación. Para realizar esto, este patrón propone la separación de estructura de la aplicación en tres componentes:

- Modelo: es la representación de la información con la que el sistema opera, por tanto, gestiona todos los accesos a dicha información (consultas, actualizaciones, etc) como la lógica de negocio. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de manipulación de datos le llegan a través del controlador.
- Controlador: responde a eventos (usualmente acciones del usuario) e invoca peticiones al modelo cuando se realiza alguna solicitud de datos. También puede enviar comandos a su vista asociada si se le solicita un cambio en la forma en la que se presenta el modelo. Se podría decir que el controlador hace de intermediario entre la vista y el modelo.
- Vista: es la representación del modelo. Por tanto, se encarga de mostrar los datos del modelo al usuario y realiza la interacción con el mismo.

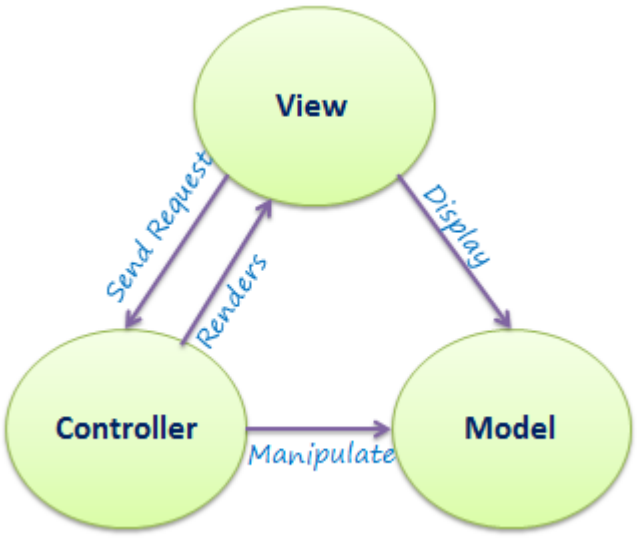

Ilustración 12: Modelo Vista Controlador

Fuente: http://www.tutorialsteacher.com/mvc/mvc-architecture

Este patrón, al igual que los dos anteriores, también está pensado para arquitecturas web. Por tanto, es conveniente tenerlo en cuenta en la decisión final.

4.6.4. Comparaciones

Todos los patrones vistos (MVP, MVVM y MVC) son factibles para el desarrollo de aplicaciones web, pero analizando el caso que nos ocupa, el patrón MVP está más enfocado a realizar pruebas automáticas de la interfaz gráfica. Con el patrón MVVM los datos de la vista se actualizan automáticamente con el modelo y viceversa. Esto puede llegar a ser una desventaja para el caso que nos ocupa, ya que muchas veces los datos de entrada que provienen del usuario pueden llegar a ser inconsistentes, por lo que no interesaría actualizar el modelo en esas circunstancias. El patrón MVC ofrece la mismas posibilidades que el patrón MVVM, pero las actualizaciones entre modelo y vista se realizan de manera manual, por lo que el problema anterior quedaría solventado. Por tanto, el patrón elegido será Modelo Vista Controlador.

5. IMPLEMENTACIÓN Y DESARROLLO

En esta sección se detalla el proceso de implementación de la aplicación Pop Framework. Dado que es posible que existan nuevas versiones de este sistema, se procura detallar los procesos seguidos con un alto nivel de detalle.

5.1. CONTROL DE VERSIONES

Un sistema de control de versiones es un proceso que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueda recuperar versiones específicas más adelante.

Un método de control de versiones utilizado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en que directorio te encuentras y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo controles de versiones locales que contenían una simple base de datos en la que se llevaba el registro de todos los cambios realizados sobre los archivos.

El siguiente problema que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solventar este problema, se desarrollaron los sistemas de control de versiones centralizados. Estos sistemas tienen un único servidor que contiene todos los archivos versionados y varios clientes descargan los archivos desde ese lugar central.

De estos tipos de control de versiones, sin duda el más famoso es GIT [11], desarrollado por *Linus Torvalds* (creador del kernel de Linux), y pensado para la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuándo estas tienen un gran número de archivos de código fuente.

Ilustración 13: página de inicio de GIT

Github: es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones GIT. Permite alojar repositorios de código y brinda herramientas muy útiles para el trabajo en equipo dentro de un proyecto. Además puede contribuir a mejorar el software de los demás. Para alcanzar esta meta, Github provee de funcionalidades para hacer *fork* (clonar un repositorio ajeno) y solicitar *pulls*(subir modificaciones a un repositorio).

Para este proyecto se han utilizado 2 repositorios, ambos alojados en Github, uno para el desarrollo del cliente, y otro para el servidor web.

La url del repositorio para el desarrollo de la parte cliente (frontend) es https://github.com/jufragoso/FrontEnd-PopFramework.git

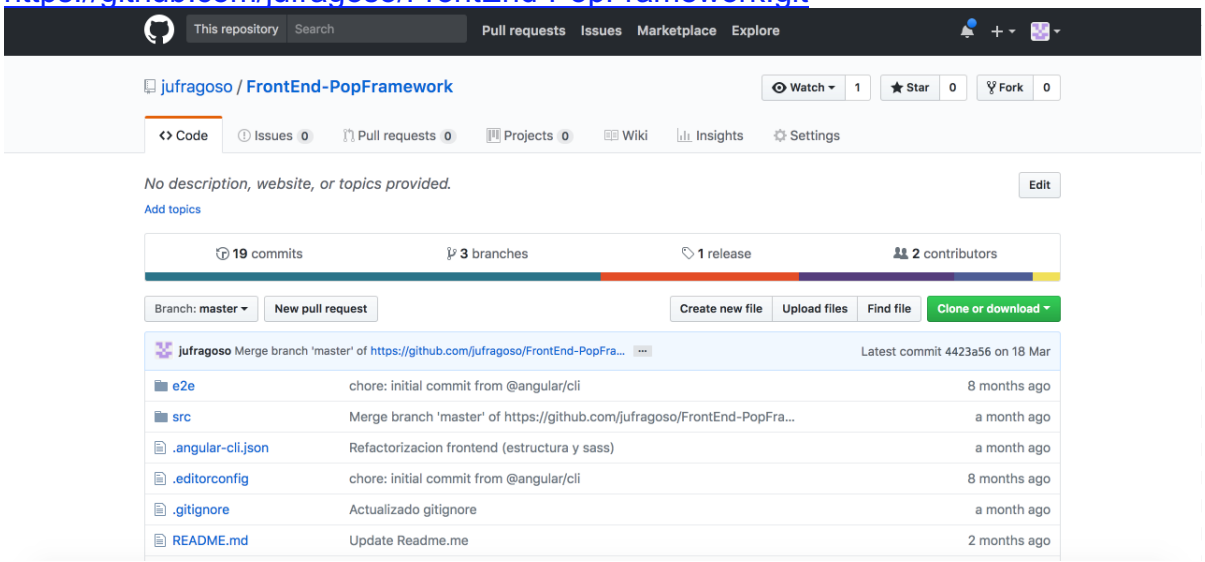

Ilustración 14: repositorio frontend

Y la url del repositorio para el desarrollo del servidor web y la API REST (back-end) es https://github.com/jufragoso/BackEnd-PopFramework.git .

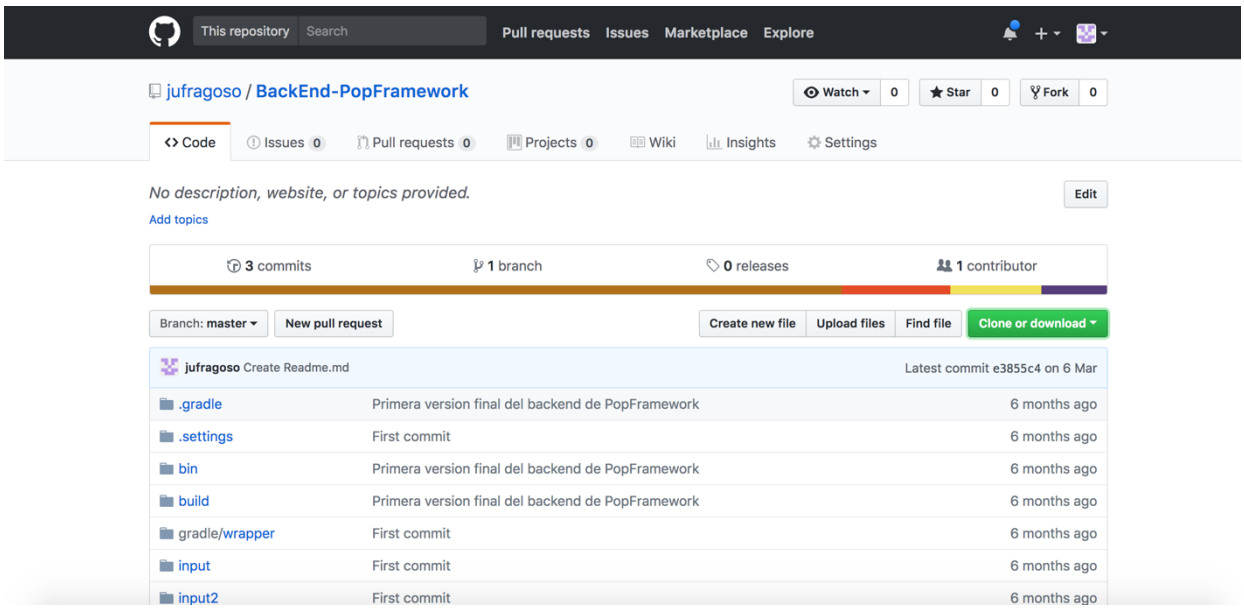

Ilustración 15: repositorio backend

El código correspondiente al desarrollo de la aplicación es el que se encuentra en la rama master (para ambos repositorios). El resto de ramas existentes se justifica por problemas encontrados durante la fase de desarrollo y que se detallarán durante el capítulo 6.

5.2. DESARROLLO DEL SERVIDOR WEB

En este apartado se documenta todo lo relacionado con el desarrollo del servidor web (backend) que, como se ha definido anteriormente, es una de las partes esenciales en la arquitectura de la aplicación y permite al usuario interactuar con la información desde el cliente web.

5.2.1. Tecnologías utilizadas

Para la construcción del servidor web se pueden utilizar diversos lenguajes y frameworks (PHP, Python, Java, Node.js, Ruby, Symfony, …), pero como se puede apreciar en la siguiente imagen, el lenguaje de programación más usado a día de hoy en el mundo es Java.

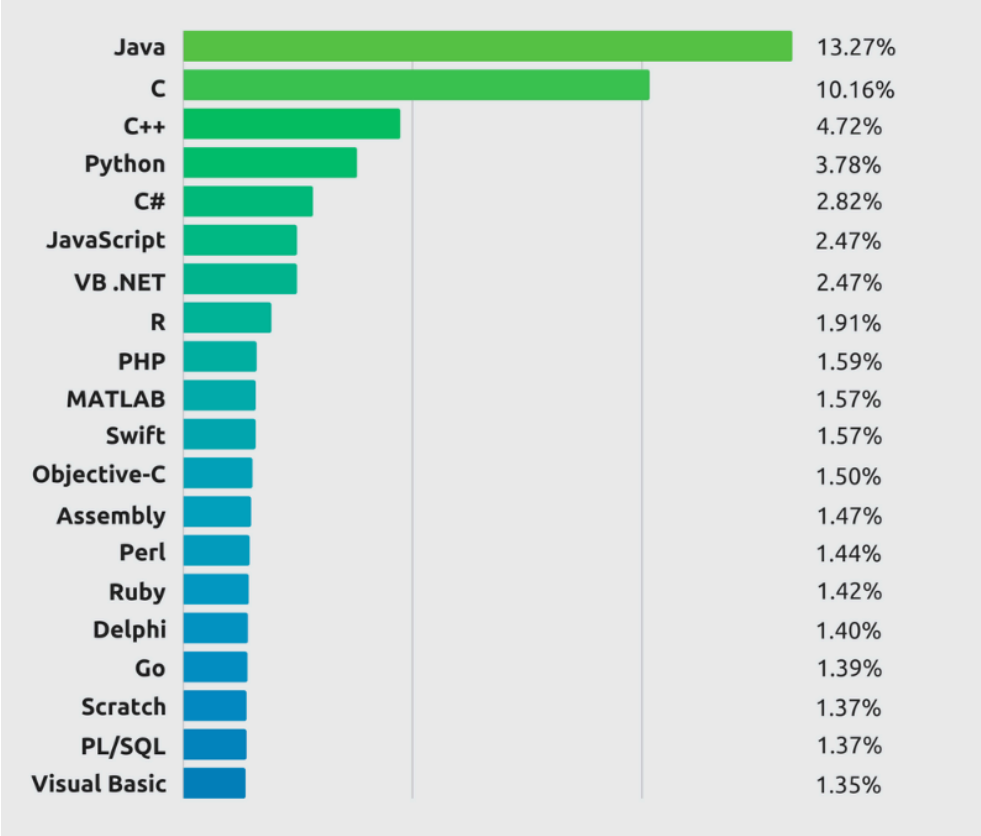

Ilustración 16: lenguajes de programación más usados.

Fuente: https://stackify.com/popular-programming-languages-2018/

Por ese motivo, en este apartado nos vamos centrar estudiar los frameworks de Java más populares.

- **Spring** → A pesar de ser un framework veterano, sigue siendo el más utilizado. Empezó enfocándose en el patrón MVC, pero siguió evolucionando hasta convertirse en un framework a gran escala de Java para aplicaciones de Internet, ofreciendo a los ingenieros grandes herramientas para el desarrollo de aplicaciones web, así como para proyectos de seguridad. Ocupa un lugar especial en las preferencias de los desarrolladores y ofrece una amplia gama de servicios: API REST, servicios web SOAP, seguridad, etc. A continuación se detallan sus ventajas e inconvenientes:

  VENTAJAS:

  - Inyección simplificada de datos de prueba mediante el uso de POJOs

  - Modularidad mejorada, lo que provoca una mejor legibilidad del código.
  - Acoplamiento flexible entre diferentes módulos.
  - Inyección de dependencias con un uso flexible.
  - Gran soporte de la comunidad.

  DESVENTAJAS:

  - Curva de aprendizaje elevada.
  - En ocasiones la información puede ser diferente entre versiones.

- **Struts 2**: sucesor de Apache Struts 2. Es otro de los frameworks Java más populares dentro de la comunidad de desarrolladores. Con este framework Apache entrega un extenso abanico de herramientas para la construcción de apliaciones web empresariales, optimizando el proceso de desarrollo desde el inicio hasta el final y el posterior mantenimiento. Es una buena elección si la aplicación va a tener una carga de datos alta.

  VENTAJAS:

  - La clase Action (encargada de procesar las peticiones de usuario) ya se encuentra incorporada, por lo que no es necesaria su implementación.
  - El código para incorporar repeticiones no es necesario porque los interceptores se ocupan de ello.

  DESVENTAJAS:

  - Framework poco ágil para desarrollar.

- **Hibernate**: framework de mapeo Java objeto-relacional que sustituye permite el abstraer el acceso a bases de datos. Está equipado con características que permite a los desarrolladores backend acceder a datos, abstrayéndose de las particularidades del lenguaje SQL. Este framework, más que un framework para el desarrollo de un servidor web, se trata de un ORM (del inglés Object Relational Mapping)

avanzado que permite realizar operaciones con la base de datos desde en lenguaje Java.

VENTAJAS:

- Este framework permite la comunicación con cualquier base de datos evitando el uso del lenguaje SQL, lo que permite abstraernos de tareas propias de los administradores de bases de datos.

DESVENTAJAS:

- A pesar de que el hecho de abstraernos de las particularidades de SQL puede llegar a ser beneficioso, algunos puntos como las inserciones múltiples están limitadas con Hibernate.

- **Vaadin**: uno de los frameworks más populares para el desarrollo de aplicaciones de negocio. Utiliza un enfoque basado en componentes conocidos, quitando peso a los desarrolladores a la hora de comunicar los cambios realizados al navegador. Un amplio conjunto de componentes de interfaz de usuario, junto con diversos widgets y controles, permite a los desarrolladores crear sistemas muy interesantes en poco tiempo.

  VENTAJAS:

  - Haciendo el diseño en Java, HTML o ambos, Vaadin permite enlazar datos utilizando el patrón MVC o MVP.
  - La característica *drag and drop*, sumadas a otras muchas más, hacen posible crear fácilmente SPAs con Java.

  DESVENTAJAS:

  - A la hora de desarrollar aplicaciones móviles ofrece una experiencia de usuario algo lenta, ya que Vaadin envía cada evento de vuelta al servidor.

- **Google Web Toolkit (GWT)**: framework libre de Google para Java que permite a los desarrolladores crear y optimizar sofisticadas aplicaciones basadas en la web. El kit de desarrollo de software GWT

ofrece APIs Java y widgets básicos para la construcción de aplicaciones compiladas después de aplicarles JavaScript.

VENTAJAS:

- Tiene una gran complementación con otras tecnologías, lo que permite a los desarrolladores integrar widgets de GWT en sus páginas web.
- La capacidad de personalizar widgets es otro aspecto importante que puede ser creado con la ayuda de este framework.

DESVENTAJAS:

- Para los diseñadores de páginas web, GWT no es el mejor framework, ya que si se pretende incorporar a un HTML estático contenido dinámico a posteriori, GWT no facilitará el trabajo.

Tras ver las ventajas y desventajas de los frameworks más populares en Java, podemos extraer las siguientes conclulsiones:

- Hibernate no es una opción válida para el desarrollo del servidor web, ya que más que un framework Java para el desarrollo de un servidor web, se trata más bien de un ORM para mapear el acceso a base de datos.
- Vaadin es un framework que está enfocado al desarrollo de una aplicación web entera, no de un servidor web únicamente (la parte del cliente web se construirá de otra manera) por lo que esta opción también queda descartada.
- Google Web Toolkit podría ser una solución que nos lleve a buen puerto, pero teniendo en cuenta que tanto Struts como Spring gozan de mayor popularidad y, por tanto, mayor apoyo de la comunidad, a la hora de buscar soluciones a los posibles problemas que surjan durante el desarrollo será más fácil encontrar información si se trabaja con Spring como con Struts.

Por tanto, tras estas consideraciones, nos quedaría Struts o Spring. Uno de los objetivos de este trabajo es aprender tecnologías que sean útiles posteriormente en el mundo laboral. Sin duda alguna, Spring goza de mayor uso y popularidad que Struts. Esto, unido a que Spring permite un desarrollo más ágil que Struts (como se ha visto en los párrafos anteriores), decantan la balanza por Spring. Por tanto, el servidor web será desarrollado con Spring Framework.

5.2.1.1. Spring

En esta sección se profundizará más en el framework Spring, con el objetivo de conocer todas sus vertientes y particularidades.

Spring [17] es un framework para desarrollo de aplicaciones de código abierto para la plataforma Java, cuyo principal objetivo consiste en facilitar el desarrollo de aplicaciones Java acoplándose a las aplicaciones sin necesidad de modificar el código para tener acceso a las distintas funcionalidades y ventajas que proporciona.

Un punto fuerte de Spring se encuentra en la inyección de dependencias. Mediante anotaciones Spring se encarga de construir los objetos que la aplicación va a utilizar. De esta manera, únicamente se cargará en memoria los objetos que sean necesarios en un momento dado y favorece a reducir el acoplamiento entre las clases que componen la aplicación.

Spring cuenta con algunas variantes, cada una apropiada para distintas funciones. A continuación se detallan las más conocidas:

- Spring Boot: es el punto de partida para construir cualquier aplicación web basada en Spring. Está diseñado para ponerlo en funcionamiento lo más rápido posible, con una configuración inicial proporcionada por el propio Spring.
    - Es apropiado para la construcción de API REST, WebSocket, Web Streaming, …
    - Proporciona seguridad.

  - Tiene soporte para SQL y NoSQL.
  - Soporte en tiempo de ejecución para Tomcat, Jetty y Undertow.
  - Proporciona productividad al desarrollador con herramientas como el livereload o el reinicio automático.
  - Puede incorporarse ciertas características útiles para entornos en producción, como las métricas.

- Spring Cloud: Facilita la construcción de sistemas distribuidos. Proporciona un modelo de programación simple y accesible para los patrones de sistemas distribuidos más comunes, ayudando a los desarrolladores a construir aplicaciones flexibles, confiables y coordinadas. Spring Cloud está construido sobre Spring Boot.
- Spring Cloud Data Flow: facilita la creación y canalización de datos en la nube para casos de uso como el procesamiento de datos, el análisis en tiempo real y la importación/exportación de datos. Simplifica la conexión de sistemas al proporcionar conectores listos para su uso en los escenarios de integración más comunes.
  - Admite el procesamiento de datos en tiempo real así como transformarlos y analizarlos.
  - Soporta conectores para FTP, RDBMS, Cassandra, RabbitMQ, GemFire, Redis y otros.
  - Admite los middleware Kafka y RabbitMQ.
  - Compatible con otras plataformas como Kubernetes, Apache YARN, …

Como se puede apreciar, Spring ofrece un sinfín de funcionalidades que van más allá del objetivo de este trabajo, pero si nos fijamos en el soporte que nos ofrece Spring Boot (servidor de aplicaciones en tiempo real, inicialización de scafolding del proyecto de manera rápida, …), nos puede resultar de gran utilidad su uso para la implementación del servidor web.

5.2.1.2. Gradle

Spring se basa en un gestor de dependencias para gestionar las librerías externas que requiere el proyecto. Las más conocidas son Maven [18] y Gradle [19]. A pesar de que actualmente Spring es frecuentemente utilizado con Maven, Gradle está ganando terreno en ese aspecto, debido a que se trata de un gestor de dependencias más moderno y cada vez es más los desarrolladores que deciden integrarlo en sus proyectos.

Dado que Gradle ya ha sido utilizado durante el grado, en la asignatura *Arquitecturas Software para Entornos Empresariales*, y por tanto se parte de una cierta base a la hora de manejar este gestor de dependencias, se decide utilizar Gradle junto a Spring.

5.2.1.3. IDE utilizado

Un IDE (*Integrated Development Environment*) es un sistema software que proporciona al desarrollador las herramientas básicas necesarias para crear y testear aplicaciones. Normalmente contiente un editor de código, un compilador o intérprete (dependiendo si el lenguaje con el que se está trabajando es compilado o interpretado) y un depurador que permite al usuario pulir errores en el código a través de una interfaz gráfica. Todo esto hace que el desarrollo de software sea mucho más rápido y ameno para el desarrollador.

Existe una inmensa variedad de IDEs actualmente, ya sean gratuitos o de pago, pero para trabajar con Spring destacan principalmente tres, los cuales se detallan a continuación:

- **IntelliJ IDEA [20]:** primer IDE en Java en cuestión de características y precio. Desarrollado por JetBrains, empresa reconocida por el desarrollo de este tipo de herramientas. Contiene dos ediciones, ***Community Edition*** (versión gratuita) que proporciona características como soporte para Java, Groovy, Kotlin o Scala, integración con Maven o Gradle y herramienta gráfica para control de versiones (SVN, Git, Mercurial,…). También se puede encontrar la versión ***Commercial*** que extiende las características de la versión Community, con soporte para más lenguajes (TypeScript, JavaScript)

o detección de código duplicado (ayuda a la refactorización) entre varias más. IntelliJ tiene una integración más que aceptable con Spring, por lo que podría ser una opción a tener en cuenta. Sin embargo, el uso de este IDE requiere de un periodo de aprendizaje y adaptación, ya que en el grado se ha utilizado IDEs de otro tipo (comentados a continuación) por lo que el tiempo de aprendizaje de esta herramienta, sumado al tiempo empleado para entender el funcionamiento de Spring, dispararía mucho el tiempo empleado para el desarrollo de este trabajo. Por estos motivos el uso de esta herramienta queda descartada en un principio.

- **Eclipse IDE [21]**: es el IDE para Java más popular, ya que es de código libre y abierto y está escrito principalmente en Java. Su arquitectura permite añadir nuevas extensiones para dar soporte a otros lenguajes de programación de una manera rápida y simple. Es multiplataforma y tiene un gran soporte de la comunidad de desarrolladores. Es el IDE que más se ha utilizado durante el transcurso del grado y por tanto es familiar su uso y no requiere curva de aprendizaje.
- **STS (Spring Tool Suite) [22]**: IDE desarrollado por Spring y basado en Eclipse (es una modificación de su código fuente) que proporciona especial integración con el desarrollo en Java y el framework Spring. Dado el alcance que se pretende conseguir con el servidor web (al menos en esta primera versión del proyecto) no se encuentran gracias diferencias entre usar STS y usar Eclipse, ya que la dinámica de trabajo es prácticamente igual. Por tanto, se decide usar Eclipse por estar más familiarizado con este IDE y tener mayor experiencia a la hora de solventar problemas de configuración que puedan surgir durante el desarrollo.

Por tanto, como el IDE elegido es Eclipse, a continuación se detalla su instalación y configuración. Este proceso se explica para el sistema operativo Mac Os High Sierra (utilizado durante todo el desarrollo del trabajo), pero esto no es un gran impedimento ya que la forma de instalación

y configuración de Eclipse es similar para todas las plataformas (Linux y Windows).

La versión utilizada será Eclipse Oxygen de 64 bits. En primer lugar, será necesario acceder a la página de descarga de Eclipse [23] y descargar el instalador de Eclipse.

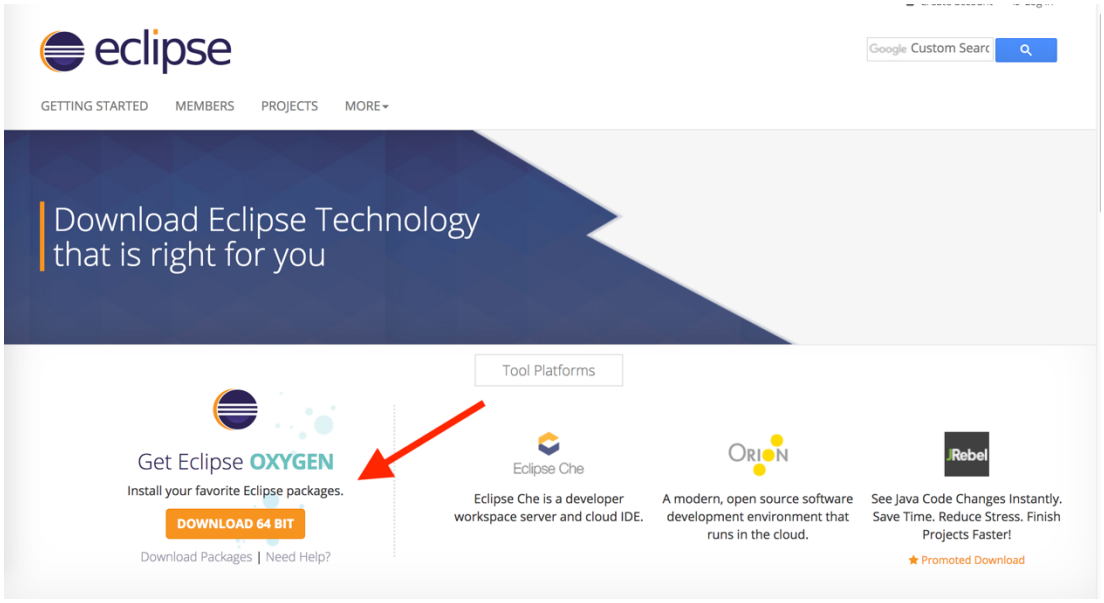

Ilustración 17: página de descarga de Eclipse IDE

Una vez descargado el instalador, se ejecuta y seguimos los pasos como se detalla a continuación:

1- En el cuadro de búsqueda de la pantalla inicial, filtramos por **"java ee"** y seleccionamos la opción **Eclipse IDE for Java EE Developers:**

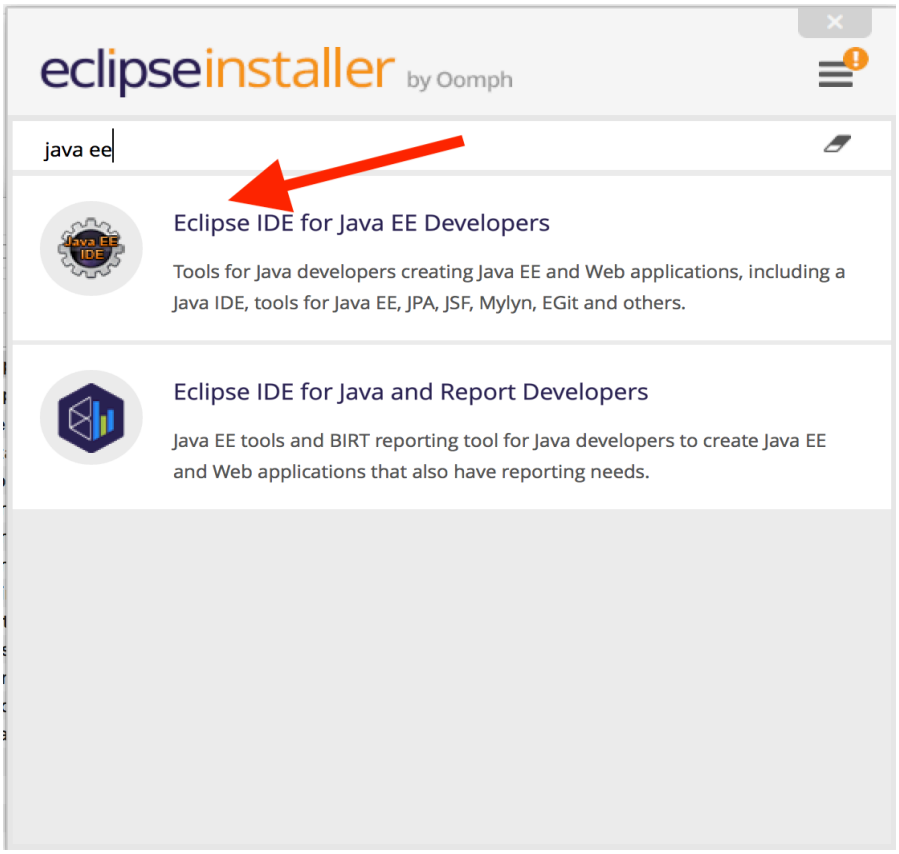

Ilustración 18: Instalador Eclipse IDE (1)

2- En la segunda pantalla, se indica la ruta donde estarán los archivos de instalación de Eclipse y posteriormente, lanzamos la instalación a través del botón **Install**.

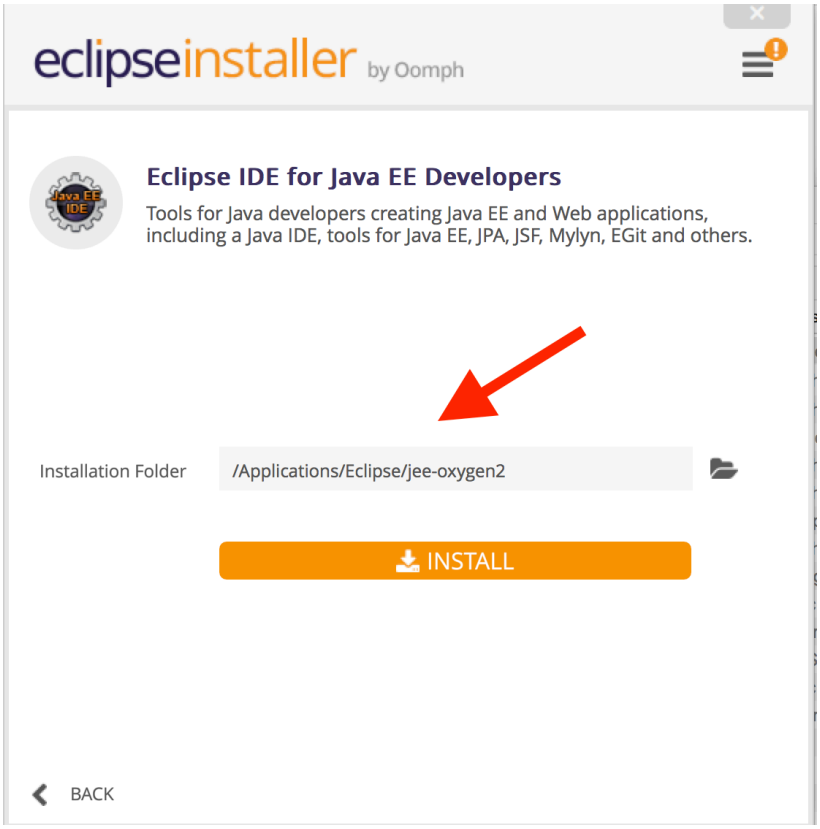

Ilustración 19: Instalador Eclipse IDE (2)

3- A continuación, Eclipse nos pedirá aceptar las condiciones de uso. Para evitar que pida aceptar las licencias una por una, se puede marcar la opción “Remember accepted licenses” y posteriormente Aceptar su uso.

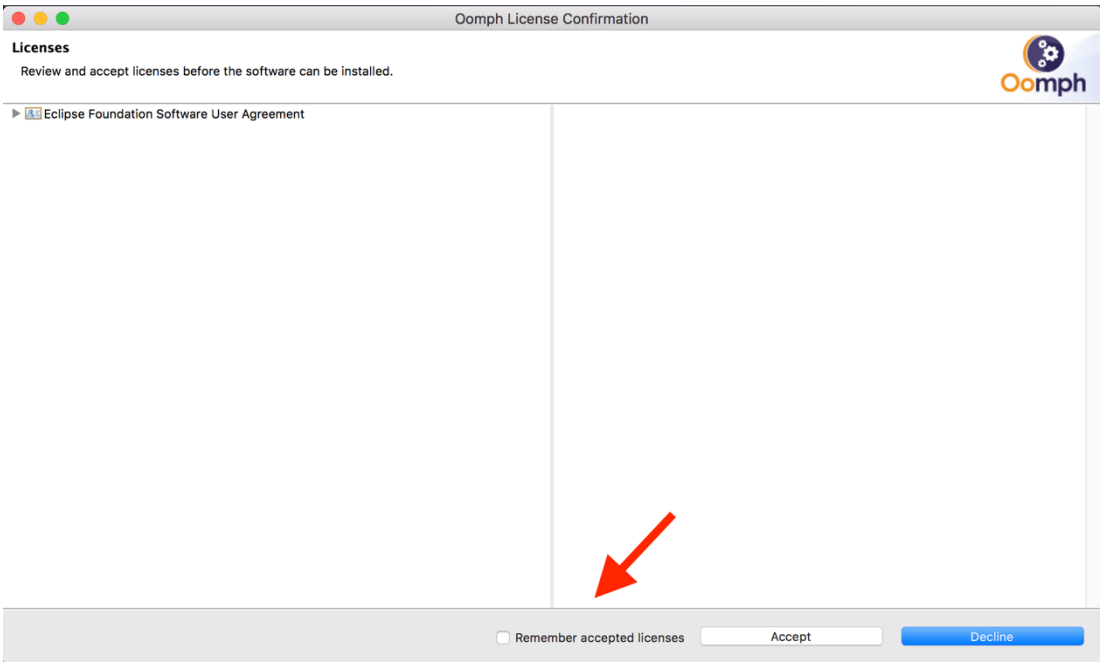

Ilustración 20: Aceptar condiciones de uso Eclipse IDE

4- Una vez finalizado el proceso, ya tendremos Eclipse disponible para desarrollar, pero aún faltaría instalar el soporte para Gradle, ya que Eclipse no lo incluye por defecto y es necesario instalarlo como un plugin adicional. Para realizar esta instalación, se siguen los siguientes pasos

   a. Se lanza Eclipse y, una vez termine el proceso de carga, seleccionamos en la barra de acciones Help > Eclipse Marketplace.

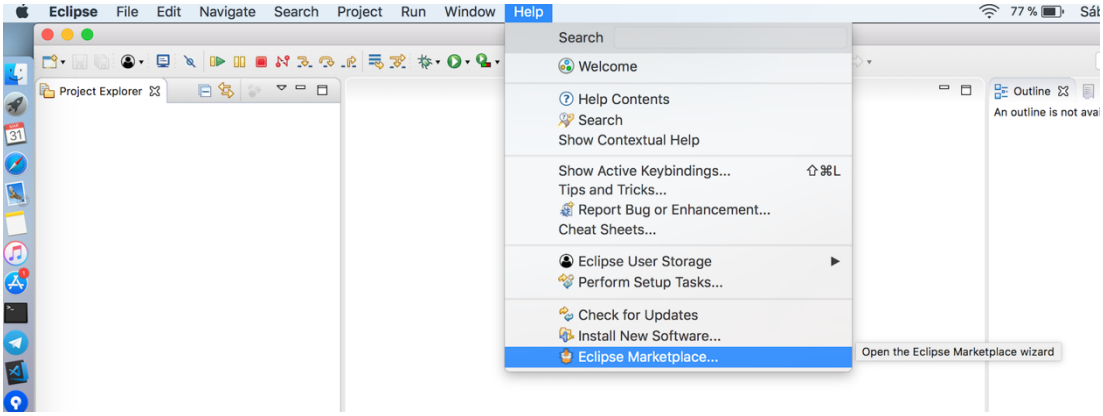

Ilustración 21: Eclipse MarketPlace (1)

   b. Se abrirá un nuevo cuadro de diálogo. Dentro del input “Find” buscamos “Buildship Gradle” esperamos a que fitre las extensiones por los criterios de búsqueda:

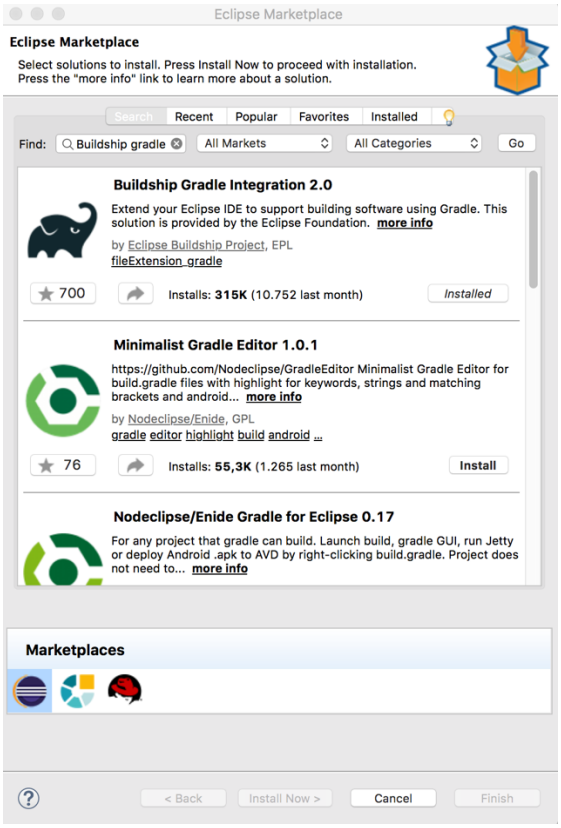

Ilustración 22: Eclipse MartketPlace (2)

c. Seleccionamos el plugin “Buildship Gradle Integration 2.0” y, pulsando en “install”, se lanzará su instalación. Posteriormente, pedirá aceptar la licencia y el proceso de instalación comenzará:+

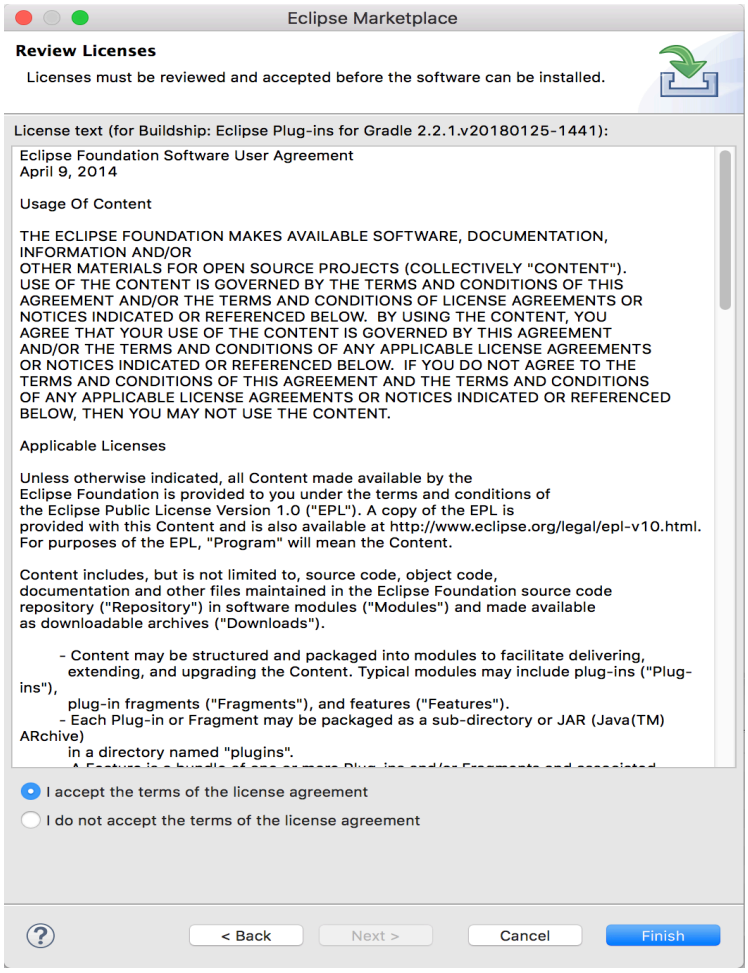

Ilustración 23: Instalación plugin Gradle (1)

d. Una vez finalizado, se pedirá reinciar Eclipse. Seleccionamos “Restart now” y Eclipse ya tendrá soporte para Gradle.

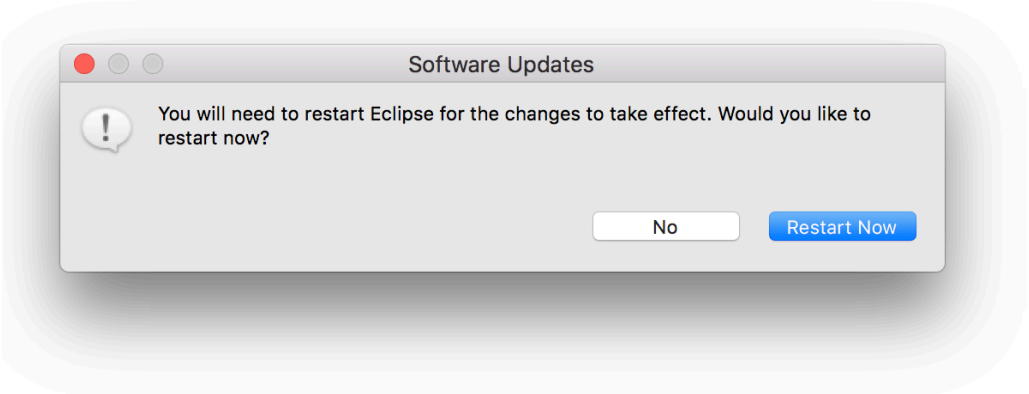

Ilustración 24: instalación plugin Gradle (2)

5.2.2. CREACIÓN DE PROYECTO SPRING

5.2.2.1. Generación de estructura básica del proyecto

Para crear el proyecto base con Spring, se utiliza una herramienta online que proporciona el propio Spring llamada Spring Initializr [24].

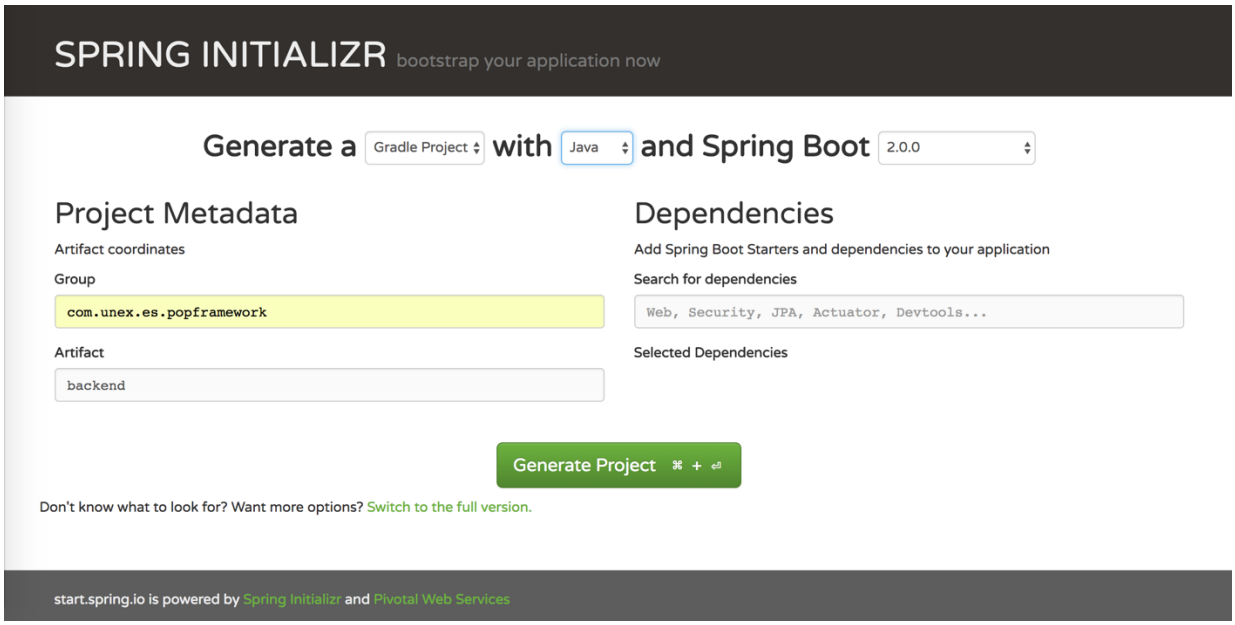

Ilustración 25: Eclipse Initalizr

Esta herramienta proporciona un scafolding básico de un proyecto de Spring Boot a partir de ciertos parámetros indicados por el desarrollador. Para el caso que nos ocupa, se genera la siguiente configuración:

- En la parte superior, indicamos “Gradle Project” para crear un proyecto con Java y Spring Boot en su versión 2.0.0.
- Como valor de Group, se indica com.unex.es.popframework. Este será el nombre de los paquetes base del proyecto y a partir del cual se basará la nomenclatura a seguir para la generación del scafolding del proyecto.
- En Artifact, indicamos como nombre “backend”. Este será el nombre del proyecto que se creará.

Una vez indicados todos los parámetros comentados, seleccionando “Generate Project” se descargará un proyecto Gradle disponible para ser importado en Eclipse.

5.2.2.2. Importar proyecto en Eclipse

El proyecto generado por Spring Initializr debe ser importado a Eclipse como un proyecto Gradle. Esto se puede realizar de la siguiente manera:

- Abrimos Eclipse y seleccionamos la opción Import:

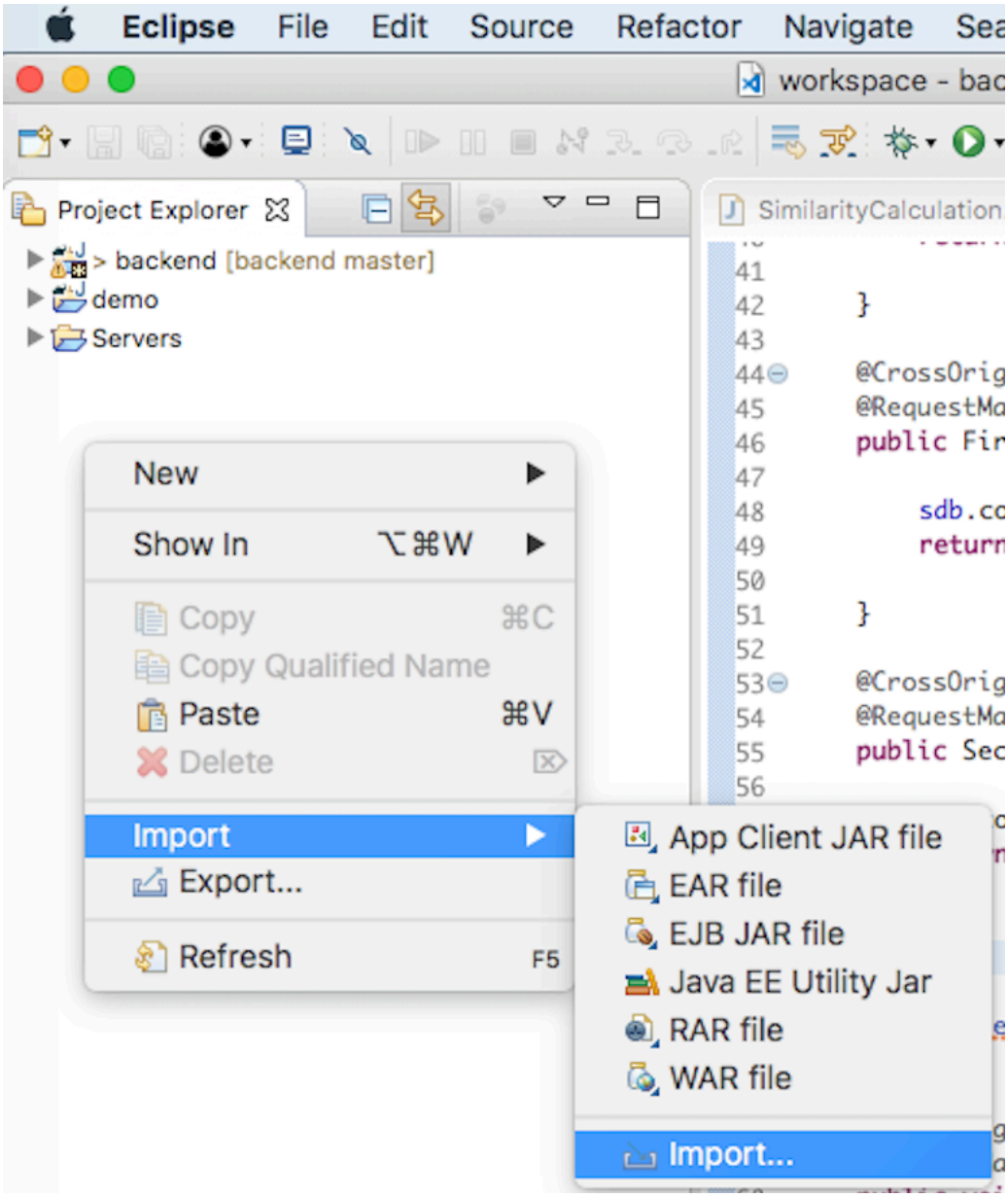

Ilustración 26: Importar proyecto Gradle (1)

- A continuación, seleccionamos la opción “Existing Gradle Project”.

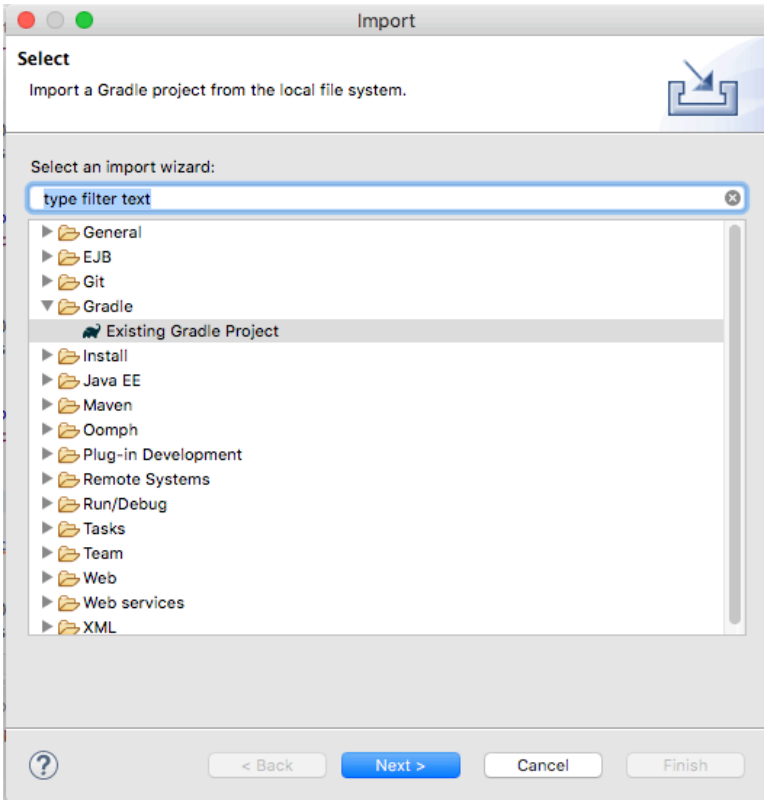

Ilustración 27: Importar proyecto Gradle (2)

- Posteriormente podremos visualizar una pantalla de bienvenida al asistente de importación de proyectos Gradle. En esta pantalla no hay información relevante por lo que puede omitirse pulsando sobre "Next".

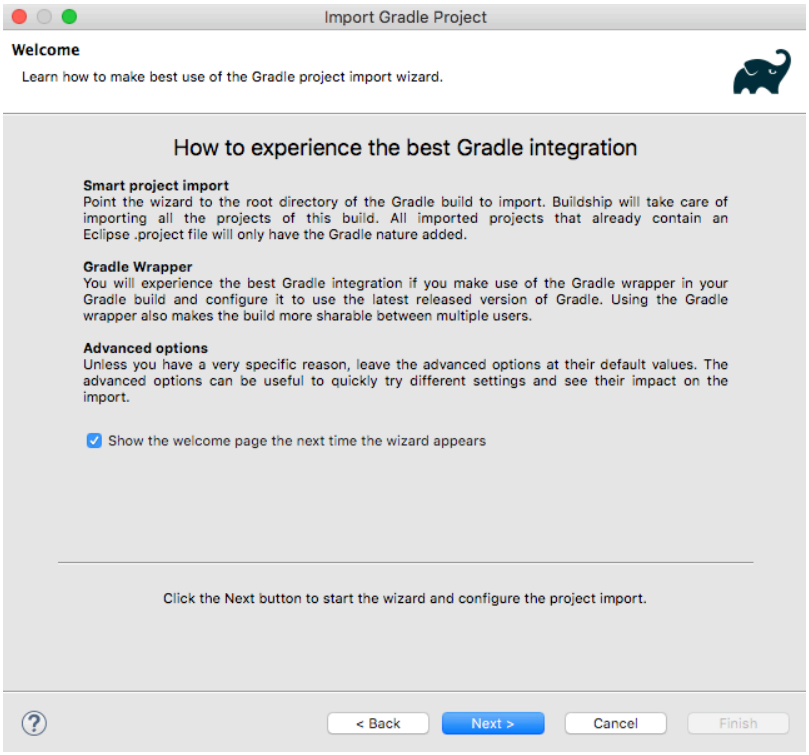

Ilustración 28: Importar proyecto Gradle (3)

- En última instancia, en la siguiente pantalla seleccionamos el proyecto generado por Spring Initializr (se debe descomprimir previamente) y pulsamos sobre “Finish”. Esto importará nuestro proyecto Gradle al workspace de Eclipse y se podrá empezar a trabajar en el desarrollo.

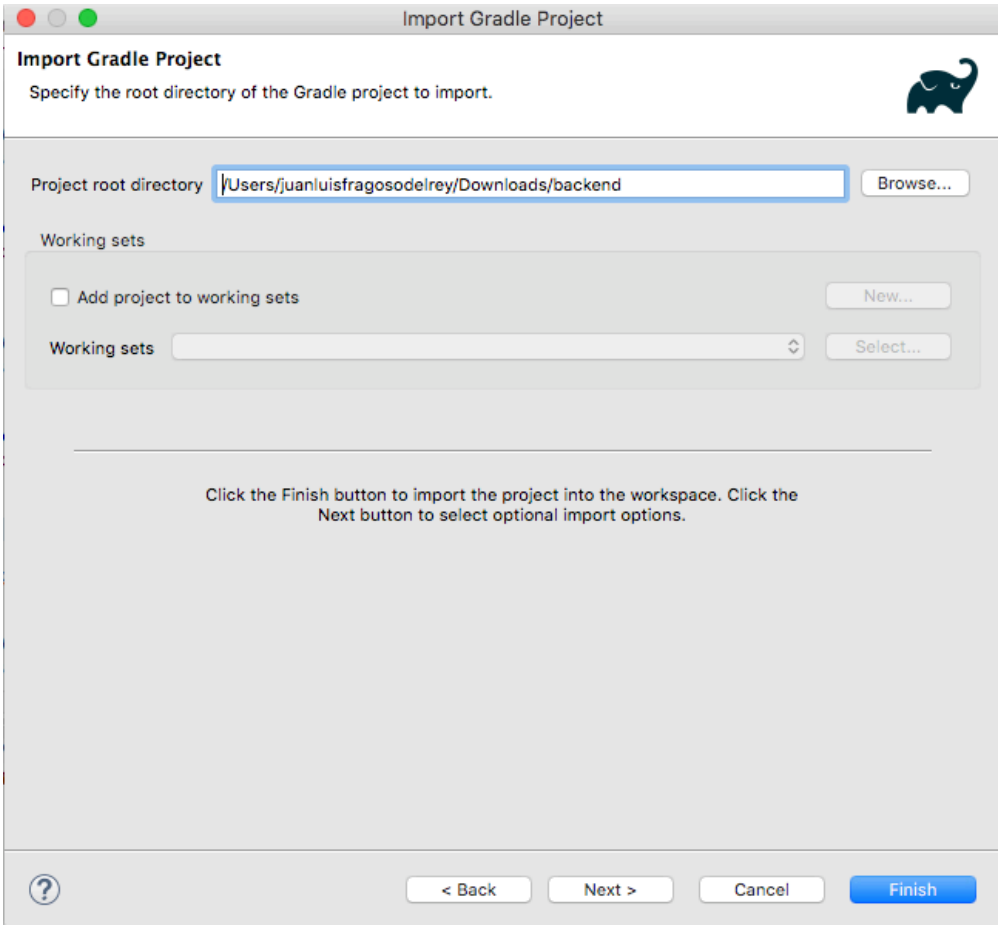

Ilustración 29: Importar proyecto Gradle (4)

5.2.2.3. Estructura básica

La estructura básica del proyecto ha sido modificada para adaptarla a las necesidades y por dotarlo de cierta separación lógica en función de la lógica de negocio que implementa cada sección. Por tanto, la división elegida ha sido la siguiente:

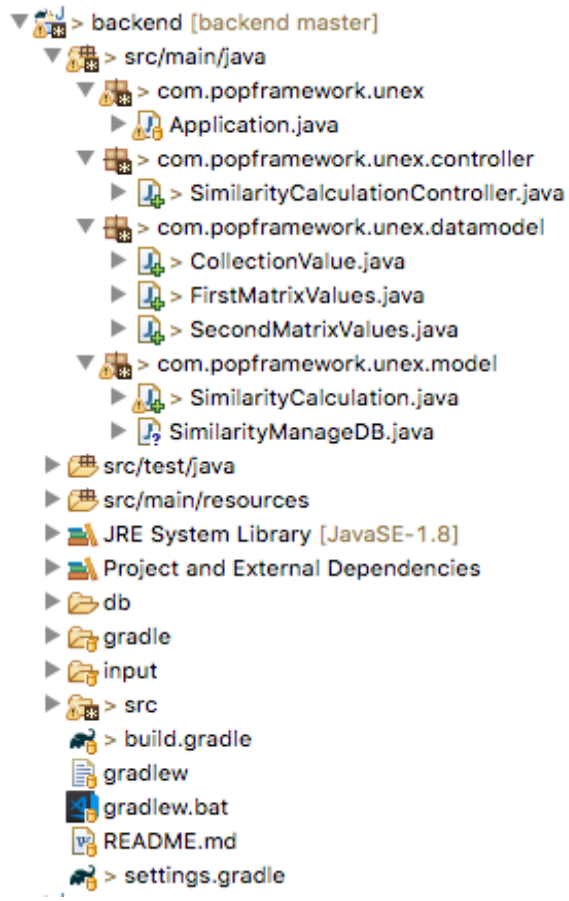

Ilustración 30: estructura servidor web

En el interior del paquete com.popframework.unex se puede encontrar los siguientes paquetes:

- "com.popframework.unex.resources" → contiene recursos utilizados como apoyo para hacer el código más legible y mantenible. Dentro de este paquete encontramos los siguientes archivos:
    - ReferencesMatrix.java → clase que representa los datos de la matriz de referencias (será comentado posteriormente). Esta clase es instanciada cada vez que se solicite dicha matriz mediante peticiones REST.
    - CharacteristicsMatrix.java → clase que representa los datos de la matriz de características (será comentada posteriormente). Al igual que en el caso anterior, esta clase es instanciada con los valores correspondientes cuando se solicite la matriz mediante una petición REST.

    - CollectionValue.java → clase auxiliar utilizada para construir los datos correctamente tanto por ReferenceMatrix.ajva como por CharacteristicsValue.java.

- "com.popframework.unex.model" → Este módulo implementa la lógica de negocio propia del servidor web y proporciona datos a la API REST cuando esta los demande. Se puede encontrar los siguientes ficheros:
    - SimilarityManageDB.java → gestiona toda la interacción con la base de datos. Cualquier comunicación con base de datos se hace a través de esta clase.
    - SimilarityCalculation → Realiza los cálculos propios de lógica de negocio (categorización, restauración de categorías,…). Se apoya en SimilarityManageDB.java para el acceso a base de datos.
- "com.popframework.unex.controller" → Implementa toda la lógica de la API REST. Define las peticiones que se pueden realizar y de qué modo. Esto se encuentra en el fichero SimilarityCalculationController.java.
- "Application.java" → clase propia de proyectos Spring Boot que encapsula el proyecto como Spring Application y permite desplegar un servidor Tomcat a través del cual se accede al servidor web. Este fichero cuelga directametne de "com.popframework.unex".

Por otro lado, existe una serie de recursos extras que son imprescindibles para el desarrollo y correcto despliegue del servidor web:

- Carpeta db → contiene los scripts sql de la base de datos y permite el acceso a la misma.
- Carpeta input→ tiene distintos ficheros con datos sobre temas, categorías, etc que son utilizados por "SimilarityCalculation.java" para implementar la lógica de negocio.

- build.gradle →fichero de configuración de Gradle en el que se indican todas las dependencias necesarias del proyecto, versión del servidor web, nombre de los ejecutables generados, …

5.2.2.4. Categorización de datasets

En este apartado se explica la implementación realizada para llevar a cabo la categorización automática de datasets.

En primer lugar, serán necesarios una serie de ficheros que contienen información básica para poder llevar a cabo el proceso de categorización. Estos ficheros son:

- categories.txt → Contiene las diferentes categorías en la que puede ser clasificado tema.
- identificadores.txt → Contiene el identificador de cada uno de los datasets que se categorizará.

En la siguiente imagen se puede ver el proceso de carga de la información contenida en dichos ficheros. Con este algoritmo se reducen los tiempos de ejecución, evitando accesos a ficheros constantemente, ya que estos accesos siempre son más lentos.

```
Algorythm loadFilesInfo (file categorias, file identificadores, set categorias, set indicadores)
begin
    for each categoria of file categorias
        add categoria to set categorias
    end for

    for each line of file indicadores
        add indicador to set indicadores
    end for
end
```

Ilustración 31: carga de datos de ficheros

En la siguiente imagen se puede apreciar el proceso de categorización implementado.

```
Algorythm procesar ()
begin
    set categories = loadFilesInfo()
    set indicators = loadFilesInfo()
    set themes
    themes = database.getThemes()

    for each theme of set themes
        var themeCategorized = false
        for each category of categories
            if theme is equal to category
                database.updateThemeCategory(theme, category)
                themeCategorized = true
            end if
        end for

        if themeCategorized is false
            var maxScore
            var categoryOfMaxScore
            for each category of categories
                var score = calculateSimilarity(theme, category)
                if score > maxScore
                    maxScore = score
                    categoryOfMaxScore = category
                end if
            end for
            if maxScore >= minScoreForCategorized
                database.updateThemeCategory(theme, categoryOfMaxScore)
            else
                database.updateThemeCategory(theme, 'Others')
            end if
        end if
    end for
end
```

Ilustración 32: proceso de categorización

Este método en primer lugar consulta a base de datos todos los temas. Posteriormente se realiza el proceso de carga de la información de los ficheros que ha sido comentado anteriormente. En última instancia se realiza el propio proceso de categorización. Este proceso se divide en los siguientes pasos:

1. Por cada tema, en primer lugar se comprueba si coincide con algunas de las categorías disponibles para la categorización. Si el tema y la categoría son iguales, se entiende que el nivel de similitud semántica es máximo, por lo que se asigna en base de datos dicha categoría.
2. En caso de no ser este el caso, se realiza el proceso de comparación del tema con cada una de las categorías. Tras realizar el cálculo de

similitud del tema en cuestión con todas las categorías, se seleccionará la categoría de mayor grado de similitud.

3. Una vez comparado el tema con todas las categorías, se comprueba categoría que mayor similitud tiene con el tema actual supera el umbral mínimo establecido. Si supera ese umbral, se asigna en base de datos dicha categoría para ese tema. En caso de no superar el umbral mínimo, se asignará la categorías “Others” para dicho tema.

En última instancia, el proceso de comparación de dos cadenas es el algoritmo de Jaro Winkler, pero adaptado para poder comparar frases, tal y como se puede apreciar en la siguiente imagen:

```
Algorythm similitud (String p1, String p2)
begin
    var count
    var sum
    for each word of p1
        for each word of p2
            sum = sum + JaroWinkler.similarity(p1.word, p2.word)
        end for p2
    end for p1

    var totalSimilarity = sum / count

end
```

Ilustración 33: cálculo de similitud de dos cadenas

En este método, como se puede apreciar, se compara cada una de las palabras que compone una cadena con el resto de palabras que componen la segunda cadena. En una variable se guarda el sumatorio de todas las medidas parciales (actúa de acumulador). Por último, se divide este acumulador entre el número de comparaciones realizadas, por lo que se obtiene la media aritmética de todas las comparaciones como medida de similitud.

5.2.2.5. Creación y destrucción de vistas

La clase SimilarityManageDB gestiona todo los accesos y manejo de datos directamente con base de datos. Por tanto, contiene los métodos para conexión y desconexión, creación y destrucción de vistas, así como

obtención de información. Esta clase implementa el patrón Singleton, comentado en la sección 4. La implementación de este patrón se define de la siguiente manera:

```
class similarityManageDB
begin

    private static instance of class similarityManageDB


    public static method getInstance()
    begin
        if instance of class if null
            call private constructor
        end if
        return instance of class
    end


    private constructor similarityManageDB()
    begin
        init instance of class
    end

end
```

Ilustración 34: Patrón Singleton

Para implementar dicho patrón se implementa un constructor privado, una instancia estática de la propia clase y un método estático que permite acceder a dicha instancia. De esta manera, a la hora de inicializar una instancia de la clase SimilarityManageDB, se accede siempre a la instancia estática definida como atributo de la clase, y únicamente se llamará al constructor cuando dicha instancia sea null.

A partir de la categorización realizada previamente, se construye una serie de vistas que facilitan la creación de los datos que el cliente web necesita para poder realizar la toma de decisiones multicriterio con el método AHP. Estas vistas son borradas previamente a su construcción en este método, ya que la base de datos SQLite no permite la modificación de vistas. Por tanto, la única alternativa para poder editarlas es borrarlas y crearlas de nuevo con los valores procedentes de la recategorización.

```
class deleteViews()
begin

    set nameOfViews = database.getAllNameOfViews()

    for each name of nameOfViews
        database.deleteView(name)
    end for

end
```

Ilustración 35: borrado de vistas

Este método realiza la sentencia sql de borrado de vista basándose en el nombre de la misma. Este nombre es consultado a la base de datos.

Por el contrario, el proceso de creación de vistas llama en primera instancia al borrado de vistas para, posteriormente, llamar a los distintos métodos de creación de vistas. Cada uno de estos métodos son mostrados en el anexo 1.

También se dispone del método que permite consultar los temas de los distintos datasets existentes en base de datos. Cabe recordar que este método se llama en el método que realiza la categorización con el fin de obtener todos los temas de los datasets que se deben categorizar automáticamente.

5.2.2.6. Creación de matrices

Para que el cliente web pueda realizar la toma de decisiones multicriterio, es necesario una serie de datos que nacen de la recategorización de datasets. Estos datos son los siguientes:

- Objeto de referencias → contiene los datos pertinentes a las referencias de los datasets que componen cada categoría en Github.
- Objeto de características → contiene los datos correspondientes a las características de los repositorios que utilizan los datasets que componen cada categoría en Github.

Estos datos son calculados a partir de las vistas generadas anteriormente y explicadas en el apartado anterior.

La función que realiza la generación del objeto de referencias es la siguiente:

```
class getReferences()
begin

    init auxiliarObject

    for each view of all views needed
        set dataReferences = database.queryToView()
        CollectionValue.add(dataReferences)
    end for

    return

end
```

Ilustración 36: obtención de objeto referencia

En este método realiza varias consultas, una a cada una de las diferentes vistas que son necesarias para construir el objeto referencias. En total se hace 5 consultas a diferentes vistas. Una vez obtenido los datos, se puede construir el objeto referencias con la ayuda de un objeto auxiliar que permite la asignación de los datos obtenidos en la consulta a cada una de las vistas, de manera que facilita la creación del fichero JSON que será comentado posteriormente.

Para la generación del objeto de características, se realiza un proceso similar. Se sigue la misma lógica que el método de generación del objeto de referencias. La única diferencia es la consulta a vistas, que en este caso son otras distintas, además de realizar 4 consultas en lugar de 5.

En los anexos 2 y 3 se puede ver la definición de la clase que permite inicializar los objetos de referencias y características. El objeto auxiliar y la clase que lo define, denominada CollectionValue, también se encuentra disponible para su consulta en el anexo 4.

5.2.2.7. Servicios Rest

Como se comentó en el apartado 4, el servidor web implementa una API REST que permitirá ejecutar las demás funcionalidades del servidor web y que ya han sido detalladas anteriormente. Para poder acceder a dichas funcionalidades, la API REST pone a disposición del cliente web algunas urls a través de las cuales se identifican las distintas funcionalidades. Cabe destacar que REST se basa en los métodos estándar HTTP, definidos por 4 verbos que son:

- GET: Proporciona un acceso de lectura de recursos.
- POST: Posibilita crear un nuevo recurso.
- PUT: Permite la modificación de un recurso ya existente.
- DELETE: Proporciona la posibilidad de borrar un recurso existente.

Dado que se usa Spring para la implementación de la API REST, estas acciones se especifican con anotaciones Java que proporciona el propio framework Spring. Las anotaciones más destacadas y utilizadas son:

- @RestController → define que los métodos implementados en dicha corresponden a un servicio REST.
- @RequestMapping → permite estableces la ruta de acceso a una funcionalidad así como el verbo con el que se realiza, es decir, que tipo de recurso HTTP estamos utilizando.
- @CrossOrigin → para permitir acceso desde orígenes distintos, es decir, imposibilitar que se produzca un fallo por acceso a recursos desde URLs distintas.

Los servicios REST pueden proporcionar los recursos de diferentes maneras, siendo las más conocidas y utilizadas JSON y XML. Para el caso que nos ocupa, se utiliza JSON como modo de proveer un recurso.

En la tabla mostrada a continuación, se puede visualizar el enrutamiento con las rutas más relevantes:

Tabla 19: métodos REST

MÉTODO	URL	FUNCIÓN
GET	getConnection	Permite la conexión con base de datos
GET	disconnect	Permite la desconexión con base de datos
GET	procesar	Lanza la categorización de datasets
GET	firstMatrix	Devuelve el objeto de referencias
GET	secondMatrix	Devuelve el objeto de características

5.2.2.8. Ejecución del servidor web

Dado que el servidor web se ha implementado con Spring Boot, podemos beneficiarnos de sus características. Esto implica que no es necesario configurar un servidor de aplicaciones como puede ser Tomcat o GlashFish, ya que el propio Spring Boot embebe un servidor de aplicaciones Tomcat a la hora de ejecutarlo.

Para poder realizar esto, Spring Boot empaqueta todo el servidor web en un ejecutable jar que puede ser lanzado por consola y que, al ser compilado, genera en la carpeta libs un fichero jar llamado PopFramework-Backend.0.1.0.jar. Para poder ejecutar dicho ejecutable basta con posicionarnos en la ruta en la que se encuentra y lanzar el comando java –jar PopFramewor-Backend.0.1.0.jar.

Al lanzar este comando, en primer lugar, se inicializa el servidor embebido Tomcat. Posteriormente se inicializa el contexto de Spring, lanzando una Spring Aplication que expone los servicios REST comentados con anterioridad al exterior.

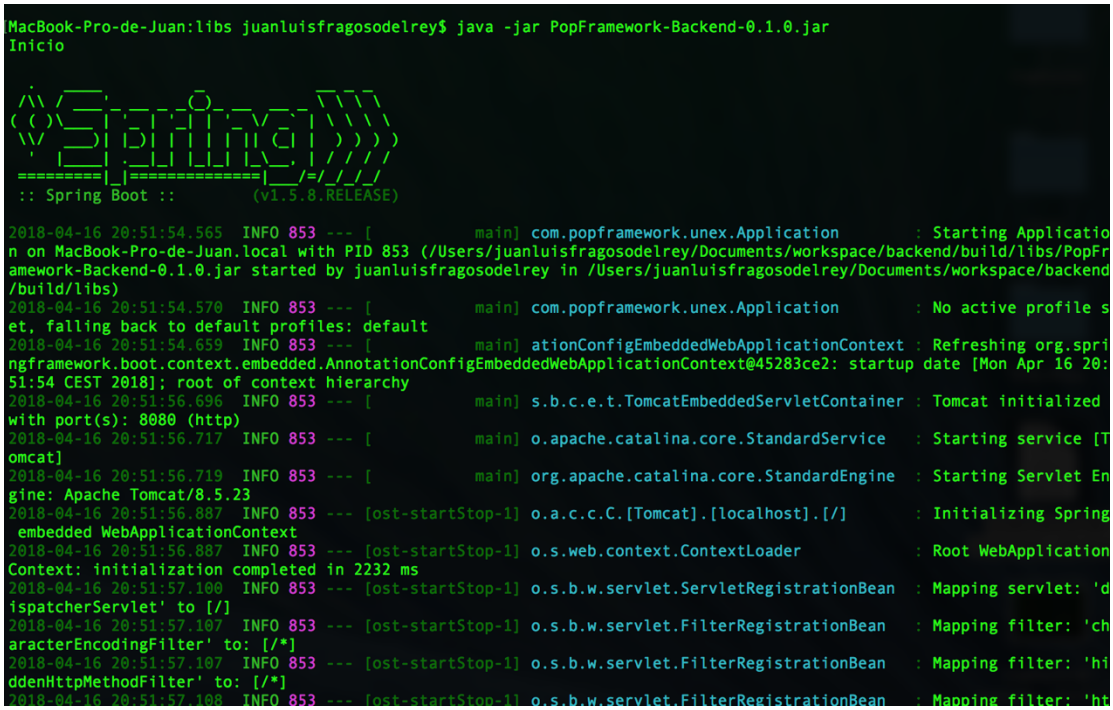

Ilustración 37: ejecución de servidor web

5.3. DESARROLLO DEL CLIENTE WEB

En este apartado se comenta todas las consideraciones tenidas en cuenta durante la implementación del cliente web.

5.3.1. Tecnologías utilizadas

5.3.1.1. Frameworks

Existen multitud de lenguajes que son muy utilizados para desarrollar aplicaciones que se ejecutan en navegadores web. Así mismo, basándose en estos lenguajes, podemos encontrar quizás incluso más frameworks que lenguajes disponibles.

A continuación se mencionan algunos de los frameworks más populares hasta la fecha:

Ilustración 38: Vue.js

Vue.js [25] es un framework Javascript del lado del cliente basado en el patrón MVVM. Vincula un modelo y una vista y proporciona enlace de datos bidireccional (de vista a modelo y viceversa). Es de código abierto y tiene un buen respaldo de la comunidad de desarrolladores. Parte de su éxito se basa en que corrige ciertos errores encontrados en otros frameworks populares como React.js o AngularJS.

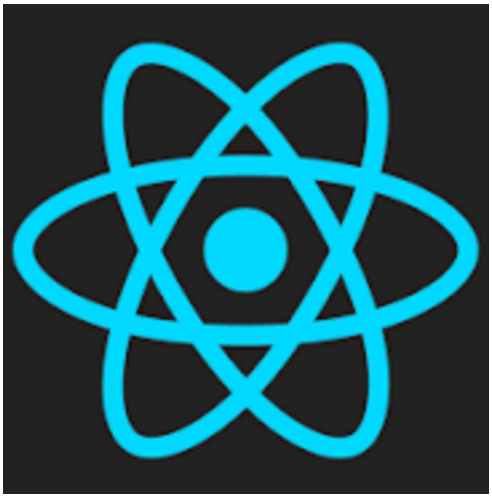

Ilustración 39: React.js

React.js [26] es un framework también del lado del cliente y desarrollado por Facebook. Su principal característica es que permite realizar renderizados rápidos. Proporciona una arquitectura view-level, es decir, otorga un lenguaje de plantillas pero con algunas funciones de devolución de llamadas para renderizar código HTML.

Ilustración 40: Ember.js

Ember.js [27] es un framework Javascript de código libre basado en la arquitectura Modelo Vista Controlador. Permite crear SPAs (Single Page Aplications) escalables. Una de sus principales características es que proporciona un data binding (enlace de datos) bidireccional, es decir, enlaza datos del modelo a la vista y de la vista al modelo, al igual que Vue.js

Ilustración 41: Meteor.js

Meteor [28] es un framework Javascript fullstack, es decir, permite implementar tanto la parte del cliente web como la del servidor. Su principal caractéristica es que destaca por el desarrollo de aplicaciones en tiempo

real. Es muy utilizado a la hora de crear aplicaciones híbridas, por tanto, es una buena opción si se desea implementar una aplicación web y una aplicación móvil basada en dicha web, ya que permite reutilizar código.

Ilustración 42: Angular.js

AngularJS [29] es un framework Javascript mantenido por Google y de código abierto. Es especialmente útil para el desarrollo de aplicaciones web de tipo SPA. Se basa en la arquitectura MVC. Sus componentes principales son tanto los controladores como los servicios. Se hizo especialmente popular a raíz de su lanzamiento y a día de hoy es muy utilizado en el mundo empresarial, pero está cediendo terreno a favor de su sucesor Angular.

Ilustración 43: Angular

Angular [30] es un framework Javascript de código abierto mantenido por Google dieñado especialmente para desarrollas aplicaciones web de tipo SPA así como aplicaciones móviles. Se trata de una evolución de AngularJS, pero no es una nueva versión, sino un framework completamente nuevo que

aprovecha los puntos fuertes de AngularJS y refuerza los puntos débiles. A diferencia de AngularJS, se base en componentes y servicios. El lenguaje utilizado en Angular por defecto es TypeScript, a pesar de ser también compatible con Javascript.

A modo resumen, se muestra la siguiente tabla con los puntos fuertes y los puntos débiles cada uno de los frameworks mencionados:

Tabla 20: frameworks frontend

Framework	Fortalezas	Debilidades
Vue.js	-Buen respaldo de la comunidad -Corrige errores encontrados en otros frameworks (React, AngularJS)	-Relativamente nuevo y poco establecido a nivel empresarial actualmente
React	-Renderizados rápidos -Compatible con TypeScript	-Curva de aprendizaje elevada
Ember.js	-Orientado a SPAs -Proporciona data-binding bidireccional	-Poco utilizado laboralmente
Meteor	-Framework fullstack -Desarrollo en tiempo real	-Poco utilizado laboralmente
AngularJS	-Gran soporte de la comunidad -Orientado a SPAs -Basado en arquitectura MVC	-Errores de diseño corregidos en Angular (siguiente versión)
Angular	-Compatible con TypeScript - Corrige errores de AngularJS	-Curva de aprendizaje elevada

	-Orientado a componentes reutilizables. -Orientado a SPAs	

Como se puede comprobar, las opciones son múltiples y la mayoría basadas en Javascript, a excepción de Angular, que va un paso más allá y utiliza TypeScript de forma predeterminada. En la asignatura de Programación en Internet impartida durante este grado, se desarrolló una aplicación utilizando el framework AngularJS. Dado que uno de los motivos que me impulsaron a desarrollar este trabajo fue aprender nuevas tecnología, la opción de utilizar AngularJS nuevamente no satisface esta motivación, a pesar de que podría suponer un hándicap ya que el conocimiento sobre este framework es mayor sobre los demás a la hora de realizar este trabajo.

Por otro lado, tanto React como Angular están acaparando gran parte de cuota de mercado en el mundo laboral, seguidos por Vue.js, que a pesar de ser un framework relativamente nuevo, está empezando a ser muy aceptado por la comunidad. Pero dado que de lo que se trata es de conocer una nueva tecnología, un factor primordial es poder encontrar una gran cantidad de recursos en internet, y en este ámbito, Angular es el primero de la lista. En último lugar, Angular permite poder desarrollar aplicaciones web SPA con gran facilidad, cosa que con React es más complicado.

Tanto Ember.js como Meteor son muy poco utilizados laboralmente con respecto a los otros frameworks, por lo que, pensando también en aprender una nueva tecnología que me sea útil en el ámbito laboral, y teniendo en cuenta todo lo anterior, la decisión final será utilizar el framework Angular junto con TypeScript.

5.3.1.2. IDES

La principal medida a tener en cuenta a la hora de elegir el IDE para el desarrollo del cliente web es la comodidad a la hora de trabajar, ya que esta

elección no afecta al rendimiento de la aplicación. Por otra parte, ya que se va a trabajar con TypeScript [31], es necesario tener en cuenta el soporte que ofrece el IDE elegido con este lenguaje.

Existen multitud de IDEs para poder desarrollar la parte front de la aplicación, como pueden ser Eclipse, Sublime Text, Atom, etc. A modo de resumen, en la figura 43 se muestra los IDEs más populares para este tipo de desarrollos.

Ilustración 44: IDEs cliente web

Las opciones son múltiples y todas válidas, pero entre todas ellas destaca Visual Studio Code [32], ya que entre todas las opciones tenidas en cuenta, es la que mayor integración tienen con TypeScript. Visual Studio Code ha sido desarrollado por Microsoft, al igual que TypeScript, por lo que no es de extrañar que su propio IDE sea el que mejor integración tenga con dicho lenguaje. Además, cuenta con un sistema de plugins o extensiones que permite enriquecer enormemente el IDE, aumentando mucho su funcionalidad y agilizando el desarrollo. Para desarrollar en Angular cuenta con numerosos plugins. Además, permite depurar desde el propio IDE el código fuente, y cuenta con una terminal integrada. Por tanto, la opción elegida es Visual Studio Code.

5.3.2. Creación del proyecto para cliente web

5.3.2.1. Node

Para poder desarrollar una aplicación en Angular, es necesario tener instalado NodeJS [33]. Como su propia página oficial indica, “es un entorno de ejecución JavaScript construido con el motor de JavaScript V8 de

Chrome". Es decir, permite ejecutar aplicaciones JavaScript además de instalar dependencias necesarias para el desarrollo, gracias a su gestor de paquetes npm.

La instalación de NodeJS es muy sencilla. Basta con acceder a su página oficial y descargar el instalador proporcionado. Una vez descargado, se sigue los pasos del instalador. Al instalar NodeJS, se instala automáticamente el gestor de paquetes npm.

Ilustración 45: Node.js

Es necesario destacar que la versión utilizada para el desarrollo de este trabajo ha sido la versión LTS (Long Than Support), ya que es considerada como versión estable.

5.3.2.2. Angular CLI

El trabajo con Angular necesita una configuración bastante extensa. Para facilitar esta configuración existen una serie de herramientas entre las que destaca Angular CLI [34].

Esta herramienta permite realizar configuraciones propias del proyecto a la vez que nos ahorra escribir mucho código. Para instalarla es necesario tener instalado previamente NodeJS. Simplemente ejecutando el comando npm install -g @angular/cli se lanza la instalación.

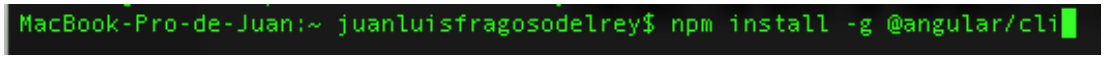

Ilustración 46: Instalación Angular-CLI

Analizando el comando se observa que lleva la anotación –g. Esta anotación es un diminutivo de –global, cuya intencionalidad es instalar Angular CLI a nivel global en el equipo, de manera que esté disponible desde cualquier ruta. Otros ejemplos de este tipo de sufijos son:

--save: el paquete se instala como dependencia exclusiva del proyecto y solo está disponible dentro de la ruta del propio proyecto.

-dev : el paquete se instala como dependencia de desarrollo, de manera que a la hora de realizar un pase a producción de la aplicación dicho paquete no será cargado.

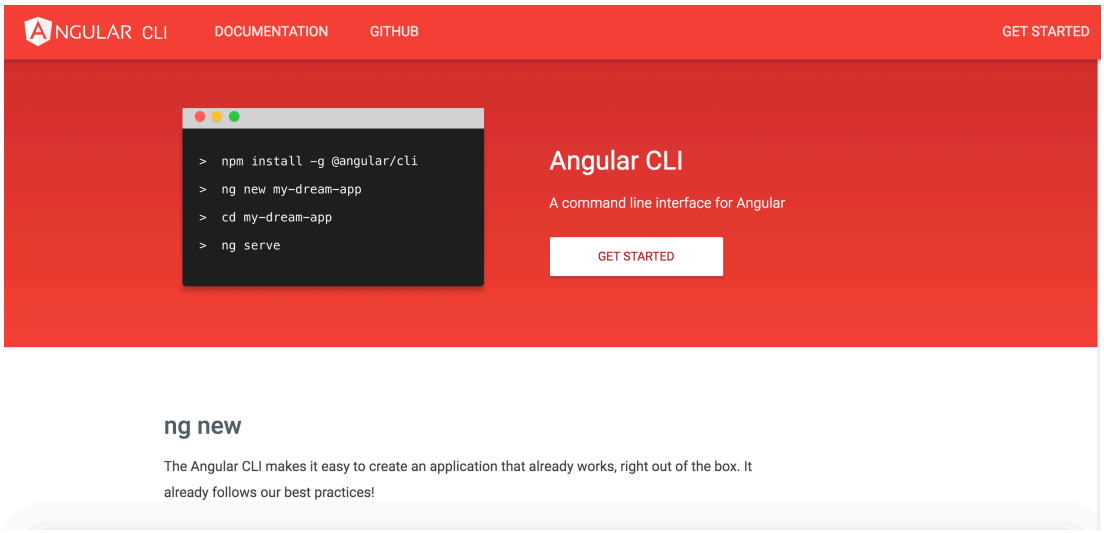

Ilustración 47: sitio web Angular CLI

5.3.2.3. Creación del proyecto y estructura

Para crear el proyecto sobre el cual se implementará el cliente web, nos apoyaremos en Angular CLI.

Con la instrucción que se muestra en la imagen, Angular CLI crea un proyecto con una estructura de ficheros y configuraciones predeterminada que se comentará a continuación.

```
MacBook-Pro-de-Juan:~ juanluisfragosodelrey$ ng new FrontEnd
```

Ilustración 48: creación proyecto cliente web

La estructura de proyectos tras terminar la ejecución del comando es la siguiente:

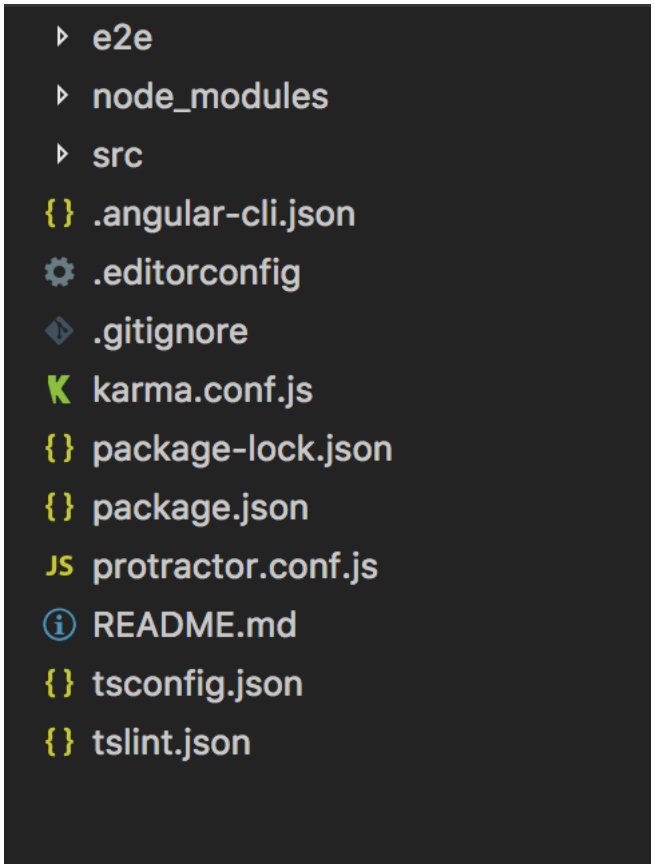

Ilustración 49: estructura aplicación Angular

Cada uno de estos ficheros y carpetas tienen un cometido, los cuales son:

- **e2e** → directorio destinado a la implementación de test end to end, también conocidos como tests de integración.
- **node_modules** → contiene todas las dependecias de NodeJS necesarias para que el proyecto funcione. Este directorio suele ser bastante extenso. Para que las dependencias se guarden en este directorio, se deben instalar con la directiva npm install –sabe *nombredependencia*, como se ha comentado anteriormente.
- **src** → Directorio de trabajo donde se encuentra todos los ficheros fuente además de otros ficheros que se comentan a continuación:

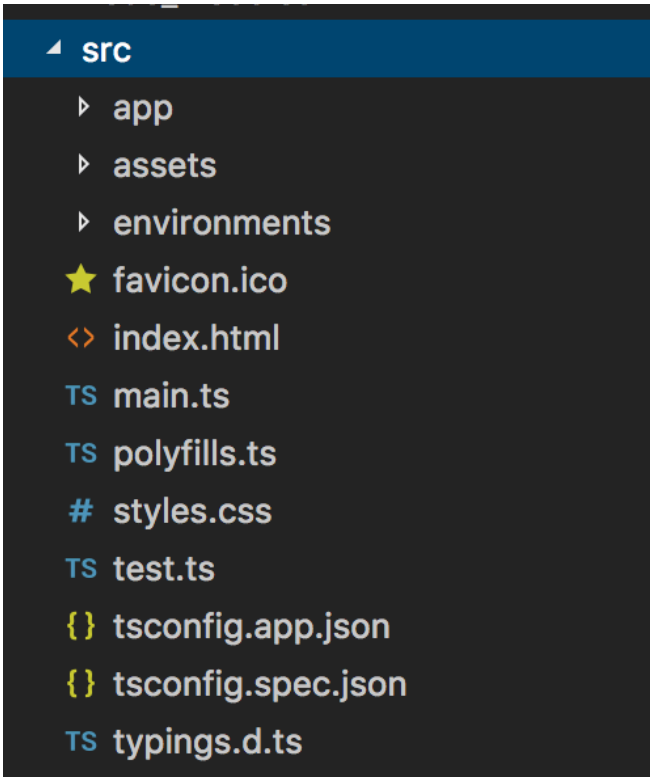

Ilustración 50: contenido carpeta src (Angular)

  - **app** → carpeta que contiene la implementación de todos los componentes y servicios.
  - **assets** → contiene todos los assets (recursos) y archivos adicionales necearios para que el proyecto funcione.
  - **environments** → en esta ruta se encuentra la configuración necesaria para desplegar el proyecto tanto en desarrollo como en producción.
  - **index.html** → archivo inicial del proyecto.
  - **main.ts** → archivo TypeScript inicial del proyecto donde se realizan todas las configuraciones globales del proyecto.

- **angular.cli.json** → este archivo contiene toda la estructura y descripción del proyecto. Además, Angular CLI consulta este fichero para saber cómo está estructurado el proyecto cada vez que tiene que ejecutar algo.
- **.editorconfig** → configuración del texto y su codificación.

- **.gitignore** → contiene los ficheros excluidos para el control de versiones git.
- **karma.conf.js** → describe la configuración para la ejecución de pruebas unitarias.
- **protactor.conf.js** → al igual que karma.conf.js, describe la configuración de tests, pero este caso de los test e2e.
- **tsconfig.json** → se encarga de realizar las configuraciones de TypeScript para la compilación de archivos con extensión ts.
- **tslint.json** → Similar a tsconfig.json, pero también permite validar ciertas características como calidad del código.

5.3.2.4. Creación de componentes

Un componente en Angular se encarga de gestionar una zona de espacio de la pantalla. En otras palabras, un componente es el responsable de la gestión de una vista. Esto lo hace a través de código TypeScript y, como es obvio, el componente lleva declarado cual es la vista que gestiona. También puede declararse que archivo contiene la maquetación de la vista, así como los servicios que utiliza (se hablará de los servicios en el apartado siguiente).

Al igual que para generar el proyecto de cero, Angular CLI se utiliza para generar nuevos componentes. Para ello, desde la raíz del proyecto se lanza el siguiente comando:

```
MacBook-Pro-de-Juan:prueba juanluisfragosodelrey$ ng generate component  nombre-del-componente
```

Ilustración 51: generación nuevo componente

Esta instrucción nos genera un nueva carpeta dentro de la ruta src/app. Esta carpeta contendrá los siguientes archivos:

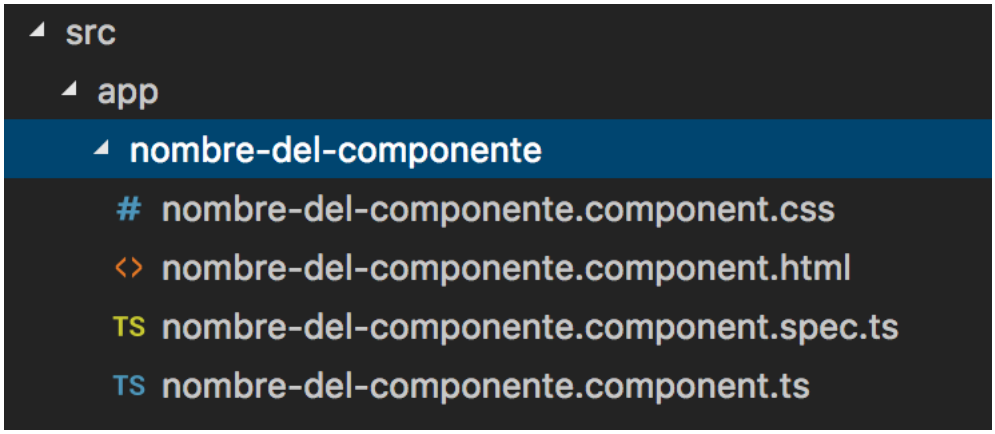

Ilustración 52: ficheros nuevo componente

La explicación de los ficheros creados es la siguiente:

- Fichero ts → contiene la lógica de negocio propia del componente, es decir, el código TypeScript.
- Fichero html → implementa la vista de dicho componente.
- Fichero css → contiene la maquetación de la vista correspondiente a dicho componente.
- Fichero spec.ts → este fichero está destinado a la implementación del test unitario del componente. En el caso de este trabajo no se va realizar tests unitarios, por lo que este fichero será ignorado.

A la hora crear el componente, automáticamente se importa el módulo principal de Angular, cuyo fichero es app.module.ts.

```
import { NombreDelComponenteComponent } from './nombre-del-componente/nombre-del-componente.component';
```

Ilustración 53: nuevo componente - angular.module.ts

A su vez, también dentro del fichero app.module.ts, este componente es instanciado dentro del array declarations, tal y como se muestra en la imagen:

```
declarations: [
  AppComponent,
  NombreDelComponenteComponent
],
```

Ilustración 54: declaración nuevo componente

Dentro del propio fichero ts del componente, nos encontramos el código que se aprecia en la imagen siguiente:

```
@Component({
  selector: 'app-nombre-del-componente',
  templateUrl: './nombre-del-componente.component.html',
  styleUrls: ['./nombre-del-componente.component.css']
})
```

Ilustración 55: selector nuevo componente

Con el decorador @Component se indica que este fichero declara un componente. Además, contiene 3 parámetros más, cuyo significado es:

- Selector → indica el nombre de la directiva con el que se hace referencia a dicho componente desde cualquier otra parte del proyecto.
- templateUrl → vincula la vista html a dicho componente.
- styleUrls → define un array en el que se vincula ficheros css a dicho componente.

Para que el decorador @Component sea reconocido, es necesario que la siguiente importación se encuentre dentro del código (por defecto lo hace Angular CLI al generar el componente):

```
import { Component, OnInit } from '@angular/core';
```

Ilustración 56: imports nuevo coponente

En última instancia, se define una clase para el componente como se muestra en la siguiente imagen:

```
export class NombreDelComponenteComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

Ilustración 57: clase nuevo componente

Esta clase es exportable, es decir, se puede importar en otros lugares de la aplicación. Define un constructor en el que se inicializa los atributos necesarios y contiene todo el código TypeScript que implementa la lógica de negocio.

5.3.2.5. Creación de servicios

Los servicios son elementos de Angular que pueden ser instanciados desde cualquier componente de la aplicación, de manera que el componente que lo instancie puede ejecutar sus métodos.

Las buenas prácticas de Angular [35] recomienda separar tanto las peticiones AJAX (usando HTTP) como la lógica pesada de la aplicación del controlador, de manera que un componente contendrá exclusivamente lógica relacionada exclusivamente con un componente (como puede ser la llamada a servicios para nutrir de datos la vista o el procesamiento de datos de entrada por parte del usuario) y un servicio tendrá tanto las llamadas a las API REST como el código que se pueda categorizar como lógica de negocio de la aplicación y no como responsabilidad de un componente (como puede ser tratamiento de datos devueltos por una petición GET).

Al igual que con los componentes, Angular CLI facilita la creación de servicios a través del comando siguiente:

```
MacBook-Pro-de-Juan:prueba juanluisfragosodelrey$ ng generate service nombre-del-servicio
```

Ilustración 58: creación nuevo servicio

Al ejecutar este comando, se generan los siguientes ficheros:

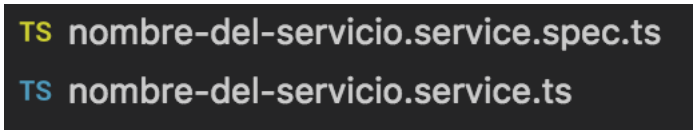

Ilustración 59: ficheros nuevo servicio

A diferencia de un componente, un servicio carece de vista, por lo tanto, no dispone ni de fichero html ni de fichero css. El fichero con extensión ts, al igual que en un componente, contiene la lógica de negocio del servicio y el fichero spec.ts implementará el test unitario correspondiente a dicho servicio.

Estos ficheros son creados en el directorio src/app a no ser que se indique lo contrario.

Un servicio tiene normalmente la siguiente estructura:

```
import { Injectable } from '@angular/core';

@Injectable()
export class NombreDelServicioService {

  constructor() { }

}
```

Ilustración 60: estructura fichero TypeScript Servicio

- @Injectable → indica a Angular que puede ser inyectado en otros lugares de la aplicación, como un componente.
- Crea una clase exportable que contendrá todo el código TypeScript. Al definirlo como exportable podremos realizar imports del servicio en otros lugares de la aplicación.

Normalmente un servicio es utilizado dentro de un componente. Para realizar esto, es necesario realizar los siguientes pasos:

- Importar el servicio dentro del componente deseado.

```
import { NombreDelServicioService } from '../nombre-del-servicio.service';
```

Ilustración 61: import nuevo servicio

- Definir el servicio como un provider dentro de las propiedades de @Component.

```
providers: [NombreDelServicioService]
```

Ilustración 62: inyección nuevo servicio (1)

- Inicializar una instancia del propio servicio. Angular tiene una forma particular de realizar este proceso, y es declarar un atributo privado directamente como parámetro del constructor, tal y como se aprecia en la imagen.

```
constructor(private service: NombreDelServicioService) { }
```

Ilustración 63: inyección nuevo servicio (2)

5.3.2.6. Maquetación

Cualquier aplicación frontend necesita de maquetación, ya que de lo contrario únicamente se mostraría el html sin ningún estilo visual, dando como resultado una aplicación visualmente poco amigable. Existen varias herramientas y frameworks que ayudan a realizar la maquetación de webs con CSS de una manera rápida y eficaz, ahorrando mucho tiempo en esta tarea. Para esta aplicación se han usado las siguientes herramientas.

- Bootstrap 3 [36]→ es un framework css que nos proporciona componentes maquetados previamente, de manera que asignando clases en nuestro html se consigue maquetar con los estilos definidos por Bootstrap. Se basa en un sistema de rejillas, en el que la pantalla se divide en 12 secciones y los componentes se pueden colocar a lo largo de ellas. Facilita la creación de diseños responsives.
- Sass [37]→ preprocesador CSS que facilita la escritura de código CSS. Angular tiene una buena integración con SASS. Permite anidamiento de código CSS y creación de funciones (también llamadas mixins) de manera que no se repita código. El código escrito en SASS se traduce a código CSS, de manera que es reconocido por cualquier navegador.
- FLEX [38]→ Es un sistema que permite colocar los elementos dentro de la pantalla con mucha facilidad gracias a su sistema de cajas. No requiere de importación o configuración alguna ya que forma parte de la especificación de CSS3. También ayuda a la creación de diseños responsives.

Para incorporar Bootstrap en el proyecto, es necesario incluir las siguientes líneas en el fichero index.html, dentro de la sección head:

```
<!--  jquery  -->
<script src="assets/jquery.min.js"></script>

<!--  Bootstrap  -->
<script src="assets/bootstrap.min.js"></script>
<link rel="stylesheet" href="assets/bootstrap.min.css">
```

Ilustración 64: inclusión de Bootstrap

Bootstrap requiere de jQuery, por lo que es incluido en la primera línea. Las dos líneas siguientes incluye los ficheros de Bootstrap minificados, que corresponden con la implementación JavaScript como los estilos CSS que aporta Bootstrap.

Por otro lado, para poder maquetar en Angular con SASS en lugar de CSS, hay que ejecutar la siguiente instrucción en la consola de comandos:

```
ng set defaults.styleExt scss
```

Ilustración 65: declaración de SASS como estilo de Angular (1)

De esta manera, cada vez que se cree un nuevo componente, en lugar de crearse un fichero con extensión css asociado, se creará uno con extensión scss, que es la extensión correspondiente a los ficheros SASS.

Si por el contrario, queremos realizar esta configuración al crear el proyecto con Angular CLI, ejecutando la siguiente instrucción podemos conseguirlo:

```
ng new my-sassy-app --style=scss
```

Ilustración 66: declaración de SASS como estilo de Angular (2)

De esta manera, al crear un proyecto nuevo también se indica que los ficheros de estilos serán de extensión scss, es decir, de tipo SASS.

5.3.2.7. Routing

La particularidad de una aplicación SPA es que no recarga la página completa, si no únicamente la parte que ha cambiado. Por tanto, en función de la URL del navegador, una aplicación SPA cargará un componente u otro, pero ciertos componentes se mantendrán intactos durante todo el tiempo de ejecución.

Angular proporciona un sistema de routing que facilita la gestión de carga de componentes en una SPA. Toda la configuración de rutas se realiza en un fichero denominado app.routing.ts. Este fichero tiene la siguiente estructura:

```
import {ModuleWithProviders} from '@angular/core';
import {Routes, RouterModule} from '@angular/router';


//Componentes
import {AppComponent} from './app.component';
import { HomeComponent } from './components/home/home.component';
import { InformacionComponent } from './components/info/infomacion.component';
import { EjecucionComponent } from './components/execution/ejecucion.component';
import { ErrorComponent } from './components/error/error.component';




const appRoutes: Routes = [

  {path:'', component: HomeComponent},
  {path:'home', component: HomeComponent},
  {path:'info', component: InformacionComponent},
  {path:'ejecucion', component: EjecucionComponent},
  {path:'**', component: ErrorComponent},
];

export const appRoutingProviders: any[] = [];
export const routing: ModuleWithProviders = RouterModule.forRoot(appRoutes);
```

Ilustración 67: routing,

Como se puede apreciar, es necesario importar los componentes ModuleWithProviders (proveniente de @angular/core), Routes y RouterModule (procedentes de @angular/router).

Posteriormente, se crea una constante de tipo Routes en la que se especifica cada ruta que utilizará la aplicación. Cada definición de una de estas rutas especifica la ruta a cargar como el componente a cargar.

Por último, dentro de este fichero, se declaran dos constantes exportables. La primera, appRoutingProviders, permitirá definir este sistema de rutas como un provider. La segunda, routing, permitirá importar el sistema de rutas en un módulo.

Para que este sistema de rutas tenga efecto, es necesario incluir las variables comentadas en el fichero app.module.ts.

En primer lugar, como import se especifica la variable routing:

```
imports: [
  BrowserModule,
  FormsModule,
  HttpModule,
  Ng2SmartTableModule,
  routing,
  BrowserAnimationsModule,
  ProgressSpinnerModule

],
```

Ilustración 68: import de routing (1)

Y en segundo lugar, se define la variable appRoutingProviders como providers el módulo app.module.ts:

```
providers: [appRoutingProviders],
```

Ilustración 69: import de routing (2)

Para que este sistema de rutas tenga efecto, es necesario que en el lugar donde se deba cargar los componentes en función de la ruta se incluya la etiqueta <router-outlet> </router-outlet>.

```
<div><router-outlet></router-outlet></div>
```

Ilustración 70: import de router-outlet

5.3.2.8. Componentes creados

Dado que se trata de una aplicación SPA, disponemos de unos componentes que permanecen estáticos mientras otros componentes son

dinámicos. En la figura 69 se puede apreciar el árbol de compontes del cliente web:

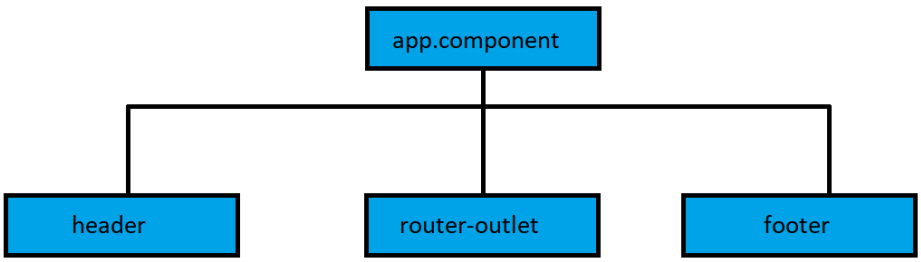

Ilustración 71: Estructura de componentes (1)

Como se puede apreciar, el componente app.component contiene los componentes header y footer además del router-outlet, que como se ha comentado cargará componentes dinámicamente en función de la URL del navegador.

El router-outlet puede cargar los siguientes componentes:

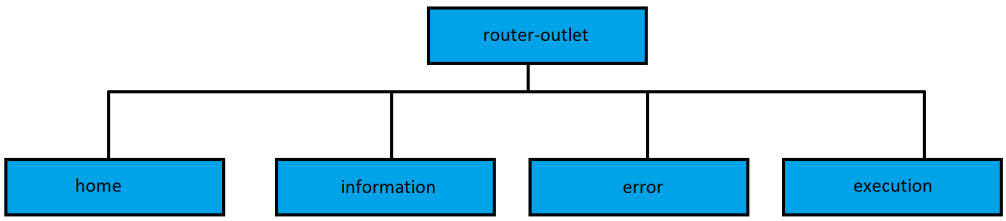

Ilustración 72: Estructura de componentes (2)

Los cuatro componentes disponibles para el router-outlet implementan los casos de uso del sistema:

- Home → Es el componente inicial. Se carga con la ruta por defecto y al iniciar la aplicación. Contiene información abreviada sobre la aplicación, principalmente sobre su funcionalidad.
- Information → implementa el caso de uso "Ver tutorial de la aplicación". Consiste en mostrar cómo se debe ejecutar la aplicación. Solo se cargará cuando la ruta sea http://localhost:4200/info
- Ejecucion → Contiene el resto de casos de uso, es decir:

- Lanzar método rápido.
- Lanzar método profundo.
- Establecer preferencias.
- Recategorizar registros (este caso de uso en realidad se realiza en el servidor web, pero es el frontend quien lanza el proceso).
- Calcular registro (este caso de uso también es ejecutado en backend, pero al igual que con el caso de uso *"Recategorizar registros"*, es lanzado desde el cliente web).
- Guardar categorías en base de datos (ocurre exactamente lo mismo que con los dos últimos casos de usos, es ejecutado en backend pero lanzado desde el frontend).

Podemos diferencias los diferentes casos de uso como sigue:

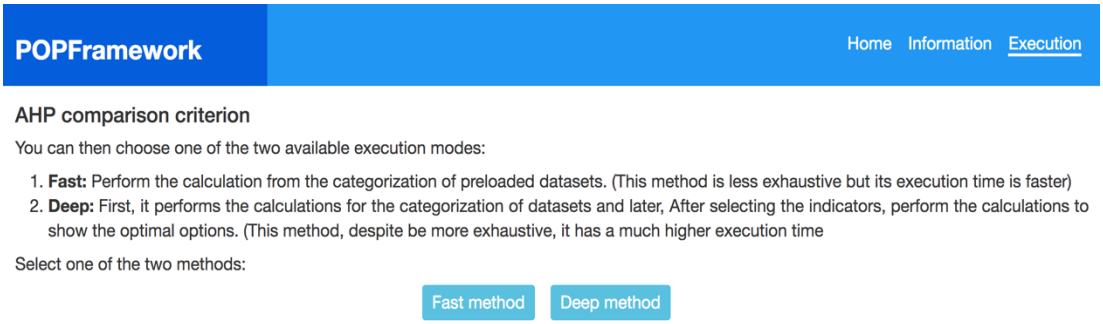

Ilustración 73: implementación componente ejecución

En este componente, como se ha comentado anteriormente, se implementan la mayoría de casos de uso, además de realizar la mayoría de llamadas a servicios. Esto es así ya que, como indican las guías de buenas prácticas de Angular, la lógica de negocio pesada se delega a servicios, por lo que el componente solo se encarga de la lógica de negocio correspondiente a la representación visual de los datos obtenidos.

- Error: Este componente se muestra cuando la URL del navegador no es correcta, es decir, no está configurada en el sistema de rutas del cliente web.

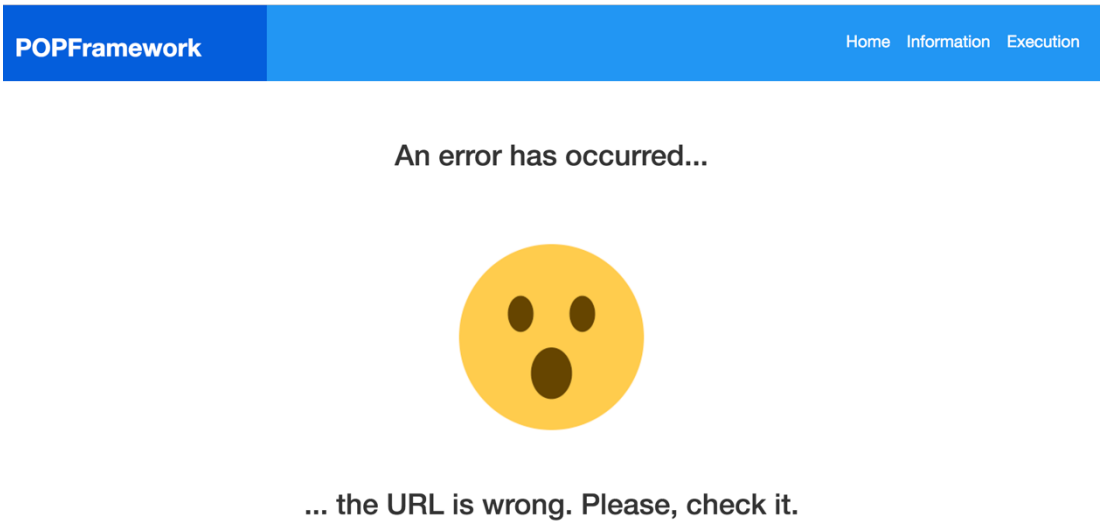

Ilustración 74: implementación componente error

Por último, y a modo resumen, en la siguiente tabla se muestra las diferentes rutas configuradas en la aplicación, así como los componentes que se cargan:

Tabla 21: Roting-componentes

URL	Componentes
Localhost:4200	Header, Home, Footer
Localhost:4200/info	Header, Information, Footer
Localhost:4200/ejecucion	Header, Execution, Footer
Otra url	Header, Error, Footer

5.3.2.9. Servicios creados

Como se ha comentado en el apartado 5.3.2.5 y como indican las buenas prácticas de Angular, los servicios separan las peticiones AJAX y la lógica de negocio pesada del resto de la aplicación.

En el caso que nos ocupa, la mayoría de la funcionalidad implementada se encuentra en el componente *"Ejecucion"*, ya que se es el que mayor casos de uso contiene. Esto no quiere decir que sea un componente pesado, ya que la lógica de negocio menos liviana se encuentra en un servicio. En base a lo explicado, se han creado los siguientes servicios:

- **PeticionesServices** → Este servicio realiza todas las peticiones al servidor web. En la tabla mostrada a continuación se muestra la relación existente entre cada método creado y la url llamada.

Tabla 22: llamada a servicios REST

Método	URL
Connect()	http://localhost:8080/getConnection
getReferences()	http://localhost:8080/FirstMatrix
getCharacteristics()	http://locahost:8080/SecondMatrix
Procesar()	http://localhost:8080/procesar
Disconnect()	http://localhost:8080/disconnect

Los métodos getReferences(), getCharacteristics() y procesar() son métodos asíncronos que son llamados según las directrices de Angular. Es decir, estos métodos realizan la petición http correspondiente y, cuando obtengan la respuesta del servidor, devolverán un tipo de datos llamado Observable y que es propio de TypeScript. Este tipo de dato, al igual que el tipo Promise en Javascript, indica que se devolverá un valor pero no de manera síncrona, sino cuando se tenga respuesta, es decir, en un futuro.

```
getReferecencesRemote(): Observable<any> {


  return this._http.get("http://localhost:8080/firstMatrix")
    .map(res => res.json());
}
```

Ilustración 75: implementación servicios 1

Para poder permanecer a la espera de este valor, desde el componente tenemos que “subscribirnos” a la petición. Esto significa que se invoca a la llamada al servicio con un subscribe de manera que TypeScript reconoce esta acción como que se devolverá un dato de manera asíncrona.

```
this._peticionesService.procesar().subscribe(
  result => {
    if (result === true) {
      this.callRemoteService();
    } else {
      console.log('Error en procesar');
    }
  },
  error => {
    let errorMsg = error;
    if (errorMsg !== null) {
      console.log(errorMsg);
      alert('Error en el procesamiento');
    }
  }

);
}
```

Ilustración 76: implementación servicios 2

- **EjecucionService** → Este servicio contiene toda la lógica de negocio que no corresponde a los componentes. Concretamente realiza funciones como toma de decisiones multicriterio con el método AHP así como la generación de datos intermedios necesarios en este método. Aislando estas funciones del componente se consigue una mejora en cuanto a legibilidad del código. Además, la aplicación gana en consistencia, ya que mejora el mantenimiento de la aplicación porque tiene un menor grado de acoplamiento y un mayor nivel de cohesión.

5.3.2.10. Librerías externas

Existen multitud de librerías externas que pueden ser acopladas con Angular [39]. Estas librerías proporcionan componentes externos preconstruidos a la aplicación que permiten aumentar la funcionalidad de la misma. Con fines de diseño, se ha incluido la librería Prime NG para mostrar un CircularSpinner mientras se produce la carga asíncrona de datos desde el backend.

Se escoge la librería Prime NG ya que había trabajo con ella en un desarrollo personal anterior al desarrollo de este proyecto, pero la forma de trabajar con este tipo de librerías y angular varía muy poco, ya que se trata principalmente de módulos npm que se incluyen como dependencias del proyecto Angular, de manera que pueden ser invocadas dentro del código fuente.

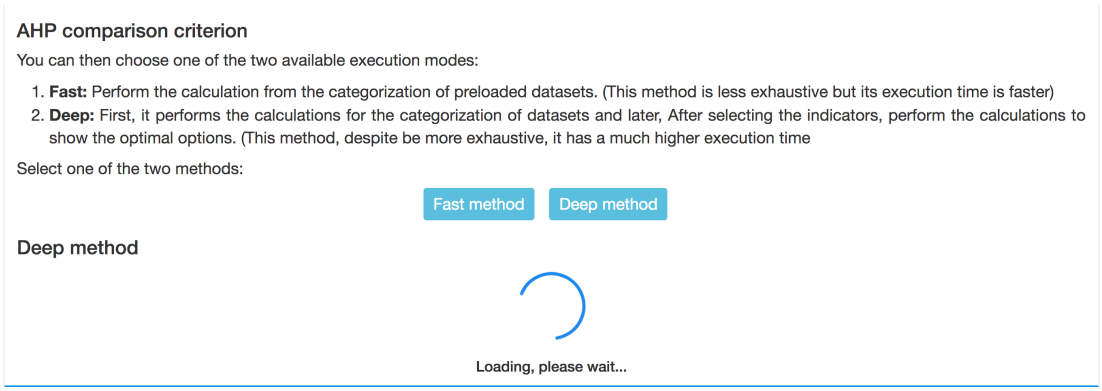

Ilustración 77: Prime NG CircularSpinner

Para incluir la librería Prime NG en el proyecto, en primer lugar se debe instalar el módulo npm que lo contiene como dependencia del proyecto:

```
npm install primeng --save
```

Ilustración 78: instalación Prime NG

Además, Prime NG se basa en el componente Animations que forma parte de Angular. Para asegurarnos que lo tenemos bien instalado ejecutamos el siguiente comando:

```
npm install @angular/animations --save
```

Ilustración 79: instalación angular animations

Una vez finalizado estos dos pasos, importamos en el fichero app.module.ts los módulos *BrowserModule* y *BrowserAnimationsModule*.

```
import {BrowserModule} from '@angular/platform-browser';
import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({
    imports: [
        BrowserModule,
        BrowserAnimationsModule,
        //...
    ],
    //...
})
```

Ilustración 80: importación Prime NG (1)

Como última configuración, se añade en el fichero index.html los siguientes estilos css:

```
<link rel="stylesheet" type="text/css" href="YOUR_PATH/font-awesome.min.css" />
<link rel="stylesheet" type="text/css" href="/node_modules/primeng/resources/themes/omega/theme.css" />
<link rel="stylesheet" type="text/css" href="/node_modules/primeng/resources/primeng.min.css" />
```

Ilustración 81: importación Prime NG (2)

Para poder hacer uso del CircularSpinner en el componente ejecución, hacemos el import del componente que lo implementa:

```
import {ProgressSpinnerModule} from 'primeng/progressspinner';
```

Ilustración 82: inclusión CircularSpinner en componente (1)

Por último, basta con invocar a dicho componente desde el fichero html cuando es necesario:

```
<div class="loadingAction" *ngIf="cargando">
  <p-progressSpinner class="text-center"></p-progressSpinner>
  <h4 class="text-center">Loading, please wait...</h4>
</div>
```

Ilustración 83: inclusión CircularSpinner en componente (2)

5.3.2.11. Ejecución del cliente web

Para ejecutar el cliente web desde un navegador, en primer lugar nos hemos de colocar en el directorio raíz donde se encuentra los archivos de la propia aplicación. En mi caso la ruta:

```
/Users/juanluisfragosodelrey/Documents/TFG/Desarrollo/TFG_Juan_Luis Fragoso/FrontEnd
```

Ilustración 84: ruta cliente web

Si es la primera vez que accedemos al directorio, lo primero que se debe realizar es ejecutar el comando *npm install*. Con este comando se descarga todas las dependencias existentes en el fichero *package.json*. Sin estas dependencias es inviable poder realizar ninguna tarea con la aplicación, ya que no contiene los ficheros necesarios para su correcto funcionamiento.

Posteriormente, para ejecutar la aplicación, Angular CLI nos provee de un servidor local donde poder desplegar la aplicación y poder acceder desde un navegador. Para lanzar este servidor se ejecuta el comando *ng serve*, Una vez lanzado el comando, Angular CLI compila los ficheros TypeScript, comprueba que las configuraciones el proyecto son correctas y, si todo ha ido bien, aparecerá un mensaje en consola como el siguiente:

```
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **
Date: 2018-04-24T22:00:12.851Z
Hash: f35735e4d71ec59cf932
Time: 14699ms
chunk {inline} inline.bundle.js (inline) 3.85 kB [entry] [rendered]
chunk {main} main.bundle.js (main) 178 kB [initial] [rendered]
chunk {polyfills} polyfills.bundle.js (polyfills) 598 kB [initial] [rendered]
chunk {styles} styles.bundle.js (styles) 466 kB [initial] [rendered]
chunk {vendor} vendor.bundle.js (vendor) 12.7 MB [initial] [rendered]

webpack: Compiled successfully.
```

Ilustración 85:ejecución cliente web

Una vez se aprecie este mensaje en consola, podremos acceder a la aplicación desde la url http://localhost:4200 .

6. MANUAL DE USUARIO

En este apartado se detalla el manual de usuario de la aplicación Pop Framework. Para explicar el funcionamiento de la misma, se realizará una navegación por toda la aplicación comentado las funcionalidades y mostrando imágenes de las mismas.

6.1. Requisitos

Para acceder a Pop Framework se requiere:

- Acceso a internet.
- Un navegador web instalado en el equipo.

6.2. Guía de la aplicación

6.2.1. Inicio

Para acceder a la aplicación es necesario abrir el navegador web y acceder a la URL https://localhost:4200 . Esto nos llevará a la página de inicio que se muestra en la imagen 86.

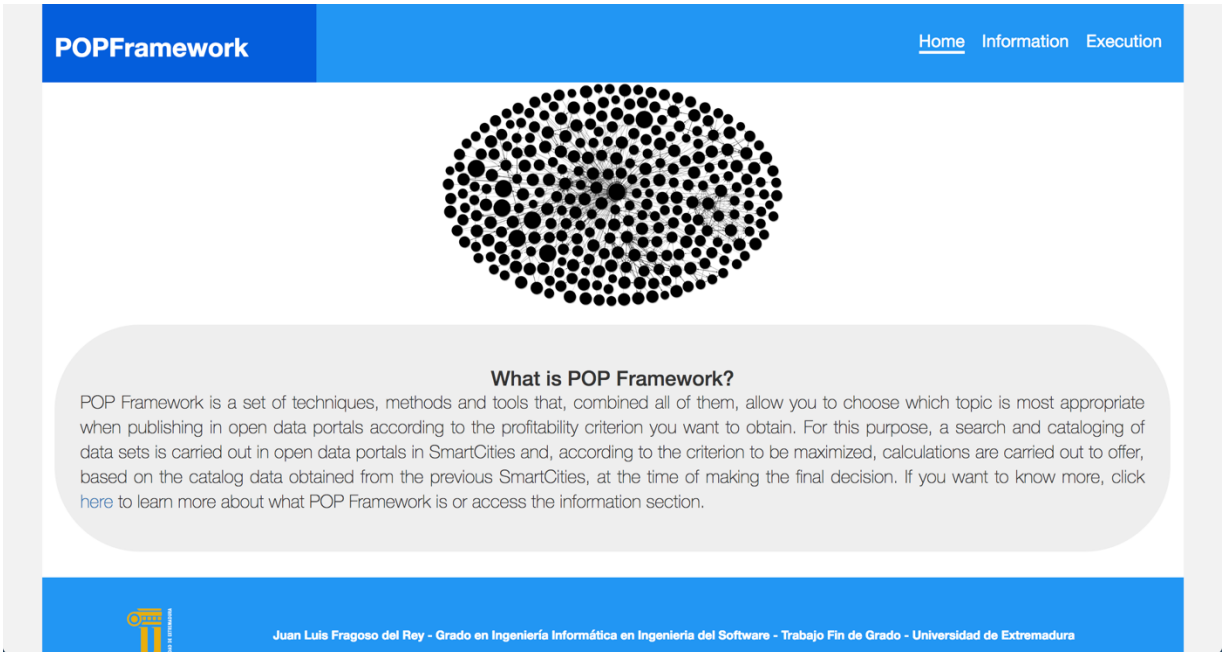

Ilustración 86: Página de inicio

En esta imagen se puede observar las diferentes secciones:

- En la parte superior se encuentra la barra de navegación que nos permite navegar por las diferentes funcionalidades de Pop Framework a través de los enlaces disponibles en la parte superior izquierda. La

sección activa actualmente se muestra subrayada por una línea blanca dentro de esta parte superior izquierda.

- En la parte central se encuentra el contenido de la aplicación y que será cambiante en función de la sección en la que nos encontremos.
- En la parte inferior se encuentra el pie de página de Pop Framework. Este contiene información extra de la aplicación que no es relevante para el funcionamiento del sistema.

Centrándonos en la parte central, se observa un pequeño cuadro de texto en el que se hace una breve introducción a Pop Framework. Al final del mismo se puede encontrar un enlace hacia la sección ***"Information"***, que contiene información más detallada.

También se puede acceder a dicha sección mediante la barra de navegación, pulsando sobre el enlace denominado ***Information***.

6.2.2. Información

La sección de *"Information"* se visualiza de manera idéntica a la imagen 87.

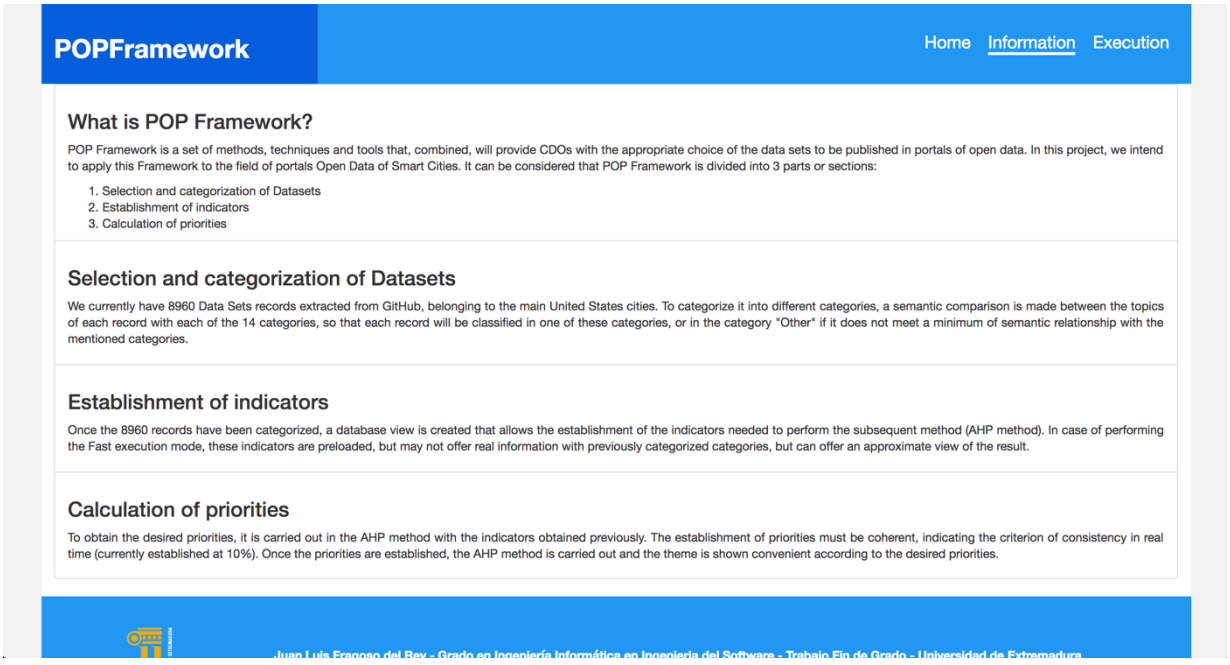

Ilustración 87: página de información

Esta sección pretende ser un mini tutorial de cómo funciona el sistema, es decir, que es Pop Framework, como se realiza la selección y categorización de datasets, el establecimiento de indicadores y el cálculo de prioridades.

Toda los datos mostrados es meramente información, no existe interacción con el usuario en esta pantalla.

6.2.3. Ejecución

A esta sección se puede acceder mediante la barra de navegación superior, pulsando sobre el enlace ***Execution***.

En primer lugar, visualizaremos una pantalla como la que mostrada en la imagen 88.

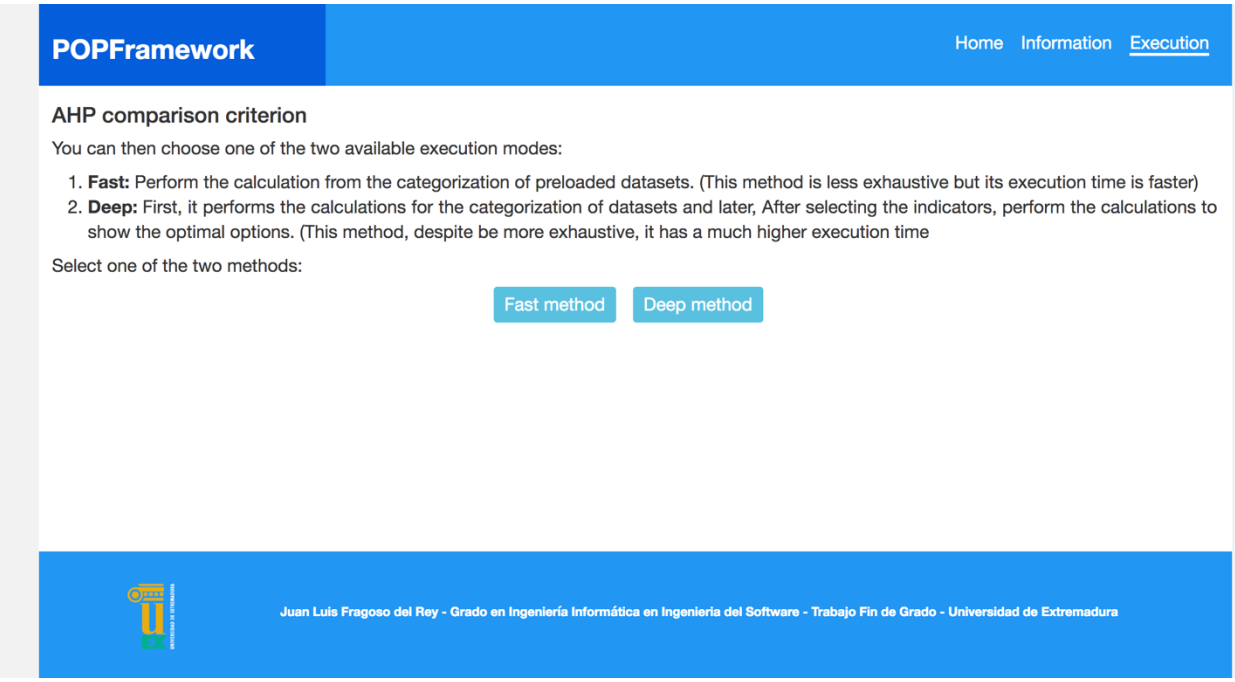

Ilustración 88: página de ejecución (1)

Como se puede apreciar, existen dos botones para elegir entre *"Fast Method"* o *"Deep Method"*. La diferencia entre ambos métodos es que el primero extrae las categorías directamente de base de datos, mientras que el segundo realiza una recategorización de todos los registros del sistema, los almacena en base de datos y, posteriormente, lee esas categorías de la base de datos. Este proceso es explicado en el texto que aparece en la parte superior de los botones.

Tras elegir una de las dos opciones deberemos esperar a que se carguen los datos internamente. Mientras se realiza este proceso, se le pide al usuario que espere mediante un indicador como el mostrado en la siguiente imagen:

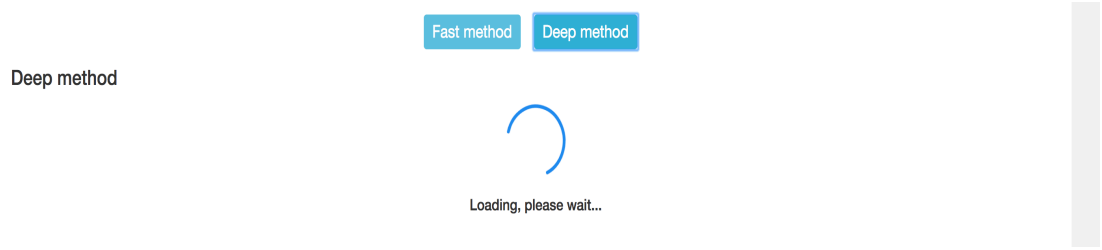

Ilustración 89: página de ejecución (2)

Tras finalizar el proceso de carga, podremos visualizar algo similar a la imagen 90.

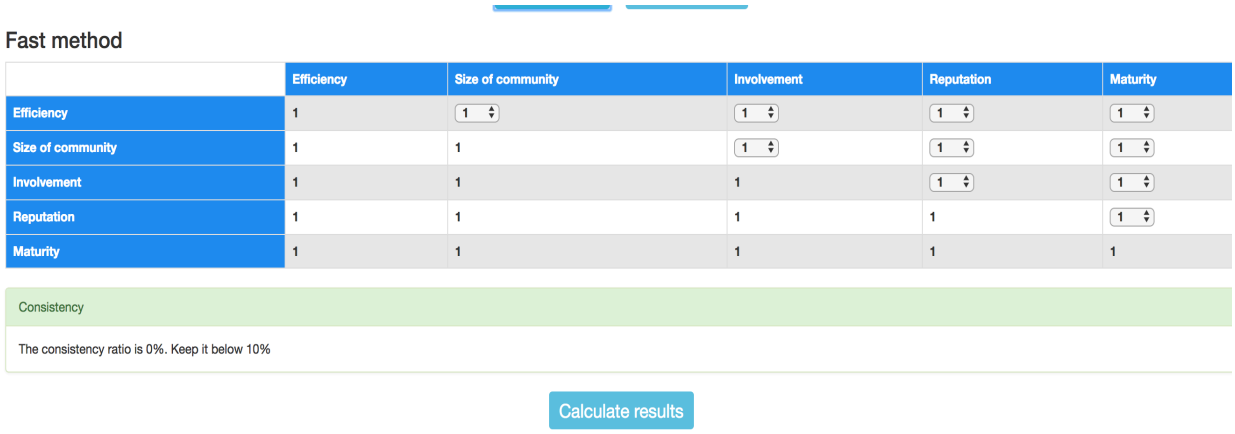

Ilustración 90: página de ejecución (3)

En la tabla mostrada, se debe indicar las preferencias en términos de la eficiencia, tamaño de la comunidad, involucramiento, reputación o madurez de los datos utilizados para el cálculo de las estimaciones.

Para establecer las preferencias, disponemos de unos selectores que contienen los siguientes valores:

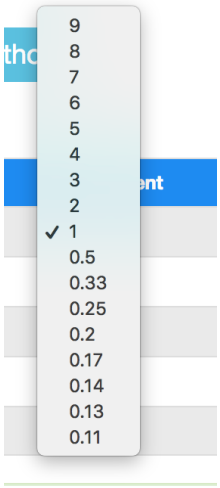

Ilustración 91: selector preferencia

Estos valores establecen la prioridad de un parámetro sobre otro. Situándonos en la fila *"Efficiency"* y en la columna *"Size of community"*, se está estableciendo cuanto es más importante la eficiencia sobre el tamaño de la comunidad. Una vez seleccionado, automáticamente se recalcula la realición inversa, es decir, cuanto es más importante el tamaño de la comunidad sobre la eficiencia. Haciendo real este ejemplo, si indicamos que la eficiencia en 2 veces más importante que el tamaño de la comunidad, se estable automáticamente que el tamaño de la comunidad es 0,5 veces más importante que la eficiencia, como se puede comprobar en la siguiente imagen:

	Efficiency	Size of community	Involvement	Reputation	Maturity
Efficiency	1	2	1	1	1
Size of community	0.5	1	1	1	1
Involvement	1	1	1	1	1
Reputation	1	1	1	1	1
Maturity	1	1	1	1	1

Ilustración 92: tabla con preferencia seleccionada

En la parte inferior de la tabla visualizamos un diálogo que nos indica la consistencia de las preferencias establecidas. Esto representa la medida en que las preferencias establecidas tienen cierto sentido. Si esta consistencia

supera el 10%, el diálogo se mostrará en rojo, por lo que los resultados ofrecidos no serán muy fiables.


Fast method

	Efficiency	Size of community	Involvement	Reputation	Maturity
Efficiency	1	2	1	4	1
Size of community	0.5	1	0.2	1	1
Involvement	1	5	1	1	1
Reputation	0.25	1	1	1	1
Maturity	1	1	1	1	1

Consistency

The consistency ratio is 11%. Keep it below 10%

Ilustración 93: consistencia erríonea

Una vez establecidas las preferencias y el nivel consistencia sea correcto, pulsando sobre el botón *"Calculate Results"* se muestra las estimaciones realizadas por el sistema.

Results	Value
1 - Ethics & Democracy	0.7585190437028994
2 - Welfare	0.7046014201245612
3 - Geospatial	0.6863758240509593
3 - Education	0.6218703336118119
4 - Safety	0.6206390918836582
5 - Demographics	0.5482385924556044
6 - Services	0.5464942213489435
7 - Recreation & Culture	0.5076025780933581
8 - Sustainability	0.4614055591389063
9 - Health	0.4549687048090853
10 - Administration & Finances	0.44774886608842446
11 - Transport & Infrastructure	0.4391087957031108
12 - Urban Planning & Housing	0.3981259458346053
13 - Business	0.39524890758352366

Ilustración 94: listado resultados

Estos resultados están ordenados de mejor opción a peor opción, mostrándose también el valor obtenido a la hora de realizar el cálculo de resultados por si se decidiera tomar otra opción que no sea primera, poder saber qué resultados ha obtenido en el sistema dicha categoría.

7. PROBLEMAS ENCONTRADOS, SOLUCIONES Y POSIBLES MEJORAS

En este apartado se documenta los problemas encontrados durante el desarrollo de este proyecto y las soluciones aportadas a dichos problemas. Por otro lado, se indica las posibles mejoras que se pueden incluir en Pop Framework en versiones sucesivas.

7.1. Problemas encontrados y soluciones

La mayoría de los problemas encontrados han sido durante la etapa de implementación del mismo, quizás motivados por la falta de manejo con las herramientas utilizadas (Angular y Spring Boot).

A continuación se detallan los distintos problemas junto a las soluciones encontradas:

- A la hora de elegir el *"Deep Method"*, es necesario realizar dos llamadas asíncronas al backend, pero la respuestas de ambas peticiones eran necesarias en cierto punto de la aplicación para poder continuar. En AngularJS hubiese resuelto el problema con promesas de JavaScript, pero el cambio a TypeScript y el hecho de que Angular recomienda utilizar Observables en lugar de Promesas me dificultaba esperar a dichos datos con el tipo Observable. Tras probar varias soluciones que no eran óptimas y buscando en internet, este tipo de problemas se puede solucionar con el método Observable.forkJoin(), el cual devuelve un array con los datos resueltos de cada llamada al backend, de manera sólo que tienes que acceder a la posición idónea en cada momento en función del dato que se necesite.
- En cuanto a maquetación, encontraba grandes problemas a la hora de mantener el pie de página siempre en el inferior de la misma cuando el resto del contenido no ocupaba el 100%. Realicé varios intentos, todos ellos basados en insertar etiquetas div y asignarles un margin dinámicamente, pero eso requería introducir jQuery dentro del código del cliente web y, a mi juicio, lo haría menos mantenible. La solución optada fue maquetarlo con CSS Flex que, gracias a su

sistema de cajas, permite tener un footer siempre el borde inferior, lo que se conoce como *"Sticky Footer"*.

- Respecto al tratamiento de datos con Angular, la tabla que permite establecer las preferencias está vinculada al modelo a través de data binding. Tras realizar varias pruebas de la funcionalidad, descubrí que con frecuencia solía actualizar mal los datos, es decir, asignaba los datos de una celda de la matriz a otra que no le correspondía. Esto se debe a que Angular presenta estos problemas al tratar con datos de tipos primitivos (enteros, booleanos, …). La solución fue encapsular cada datos dentro de un objeto, por lo que la matriz pasó de contener datos de tipo float a contener datos de tipo Object.

7.2. Posibles mejoras

En este apartado se sugieren algunas ideas de mejoras para futuras versiones de Pop Framework, muchas de las cuales surgieron durante el desarrollo del mismo ya que se iba teniendo mayor comprensión del sistema y a la vez mayor desenvoltura con las tecnologías utilizadas:

- Para la parte frontend existen unos test llamados tests de integración o tests end to end (e2e). Estos test compruebas que el flujo de la aplicación es correcto. En Angular se ejecutan bajo un entorno llamado Protactor, que permite acceder al DOM de la aplicación y comprobar que los datos obtenidos del mismo son correctos. Sería una posible mejora introducir estos tests ya que, al igual que los tests unitarios, aseguran una correcta puesta en producción con herramientas de integración continua.
- En el servidor web, se propone implementar toda la conexión a la base de datos así como la manipulación de datos a través de la misma a través de un ORM. Como se ha visto en el apartado 5, Hibernate puede ser una solución, pero existe multitud de opciones disponibles.
- En cuanto a nivel funcional, se sugiere la carga de preferencias directamente por parte del usuario. Esto se podría conseguir mediante la creación de un *wizard* donde el usuario podría establecer los

valores de sus preferencias directamente. Este wizard podría contener una ayuda, de manera que fuese guiando al usuario a establecer dichas preferencias.

8. CONCLUSIONES

En este apartado se exponen las conclusiones obtenidas tras la realización de este trabajo, tanto a nivel de objetivos como a un nivel personal.

8.1. Objetivos

En este apartado se va a analizar si se ha logrado cumplir con los objetivos marcados previamente a la realización de este trabajo, y definidos en la sección 1.1.. A continuación se vuelve a listar los objetivos del mismo:

- **Categorizar automáticamente** los diferentes conjuntos de datos almacenados utilizando algoritmos semánticos
- **Desarrollar una aplicación web** que permita realizar la toma de decisiones multicriterio a través de una interfaz amigable y con una buena experiencia de usuario. De esto objetivo nacen los siguientes subobjetivos:
    - **Desarrollar el backend** de la aplicación web para que se pueda atender las peticiones de todos los usuarios desde el navegador web.
    - **Desarrollar el frontend** de la aplicación web para que los usuarios tengan una buena experiencia de usuario y puedan interactuar con la aplicación.

La categorización automática de datasets se realiza de manera automática mediante el algoritmo de Jaro-Winkler, el cuál es un algoritmo semántico. En cuanto al desarrollo de una aplicación web, también ha sido desarrollada utilizando Spring Boot para el backend y Angular con TypeScript para el frontend, utilizando herramientas como Prime NG o Bootstrap para ofrecer una buena experiencia de usuario.

En la siguiente tabla se muestra a modo de resumen los objetivos planteados así como su grado de consecución, siendo el nivel bajo en torno a un 35%, el nivel medio cubierto alrededor de un 65% y el valor alto completamente cubierto.

Tabla 23: objetivos TFG

Objetivo	Bajo	Medio	Alto
Categorización automática			√
Desarrollo de backend de app. Web			√
Desarrollo de frontend de app. Web			√

Por tanto, se puede decir que se ha cumplido con todos los objetivos planteados con la realización de este proyecto.

8.2. Conclusiones personales

Tras realizar este trabajo, puedo decir que estoy muy satisfecho, ya que, a pesar de tardar en realizarlo bastante tiempo, me ha permitido valorar el trabajo que conlleva la construcción de un sistema de estas características y su puesta en producción.

Tengo que mencionar especialmente que el proceso de categorización ha sido una tarea bastante costosa de realizar, ya que las herramientas que encontraba en internet o bien tenían un tiempo de respuesta muy alto o bien no ofrecían los resultados esperados. Esto, sumado a que todo el campo de análisis semántico me era desconocido cuando empecé con su desarrollo, llego a ser bastante frustrante. Pero gracias a la ayuda del tutor aconsejándome vías de investigación, se consiguió llegar a una solución óptima.

En la parte frontend he de decir que tenía algo experiencia con AngularJS debido a que trabajaba con este framework en el mundo laboral, por eso, a pesar de ser un framework bastante distinto de su sucesor (Angular), aprender la siguiente versión no me resultó tan costoso, ya que muchos conceptos web los tenía asimilados.

El uso de Spring Boot me ha resultado interesante, a pesar de no haber profundizado mucho en esta herramienta. Considero que se trata de un framework muy amplio y con una curva de aprendizaje alta, requiere de horas de dedicación a aprenderlo, y en un futuro me gustaría ahondar más en esta materia.

También he aprendido a utilizar mejores técnicas de programación, ya que era de vital importancia para que no se viera afectado el rendimiento de la aplicación. Esto es algo que me va a resultar muy útil durante el resto de mi carrera profesional.

Como conclusión final, este trabajo me ha servido para crecer a nivel personal, ya que, a pesar de las dificultades encontradas, con trabajo y esfuerzo se ha conseguido realizarlo y llevarlo a buen puerto.

9. ANEXOS

Anexo 1 – Código para crear vistas

```
private void repository_useful_data_for_indicators() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
repository_useful_data_for_indicators as "
                          + "select distinct
repository_id,total_contributors,total_contributions,subscribe
rs_count,created_at,updated_at"
                          + " from measures_repository where
created_at!='' order by repository_id";
                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }

     private void
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing as "
                          + "select distinct
c.identifier,c.category,d.repository_id"
                          + " from results_search_open_data
as d  join usa_city_datasets_categorized as c on
(d.identifier=c.identifier)"
                          + " where repository_id in (select
repository_id from repository_useful_data_for_indicators)"
                          + " order by
c.category,c.identifier,d.repository_id";

                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }
```

```
	private void datasets_categorized_not_referenced() {
		try {

			Statement stmtCreate = conn.createStatement();
			String sqlCreate = "create view
datasets_categorized_not_referenced as"
					+ " select distinct
identifier,category,NULL as repository_id from
usa_city_datasets_categorized"
					+ " where identifier not in"
					+ "(select identifier from
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing)"
					+ " order by category,identifier";

			stmtCreate.executeUpdate(sqlCreate);

		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	private void categories_total_references() {
		try {

			Statement stmtCreate = conn.createStatement();
			String sqlCreate = "create view
categories_total_references"
					+ " as select category,count (*) as
total_references"
					+ " from
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing"
					+ " group by category order by
category";

			stmtCreate.executeUpdate(sqlCreate);

		} catch (SQLException e) {
			e.printStackTrace();
		}
	}

	private void categories_total_datasets_in() {
		try {

			Statement stmtCreate = conn.createStatement();
```

```
                String sqlCreate = "create view
categories_total_datasets_in as"
                          + " select category,count (distinct
identifier) total_datasets_in_category"
                          + " from
usa_city_datasets_categorized group by category order by
category";

                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }

     private void categories_total_datasets_no_referenced() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_total_datasets_no_referenced as select
category,count (distinct identifier) as
total_datasets_no_referenced from
datasets_categorized_not_referenced group by category order by
category";

                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }

     private void categories_total_datasets_referenced() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_total_datasets_referenced as"
                          + " select category, count
(distinct identifier) as total_datasets_referenced"
                          + " from
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing"
                          + " group by category order by
category";
```

```
                stmtCreate.executeUpdate(sqlCreate);

            } catch (SQLException e) {

                e.printStackTrace();
            }
        }

        private void categories_total_repositories_referencing()
{
            try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_total_repositories_referencing as"
                        + " select category,count (distinct
repository_id) as total_repositories_referencing"
                        + " from
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing"
                        + " group by category order by
category";

                stmtCreate.executeUpdate(sqlCreate);

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        private void categories_contributors() {
            try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_contributors as"
                        + " select
category,sum(total_contributors) as contributors"
                        + " from (select distinct
d.repository_id,r.total_contributors,d.category"
                        + " from
repository_useful_data_for_indicators as r join"
                        + "
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing as d on d.repository_id=r.repository_id"
                        + " order by
d.repository_id,r.total_contributors,d.category) group by
```

```
category order by category";

                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }

     private void categories_contributions() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_contributions as"
                          + " select category,
sum(total_contributions) as contributions from"
                          + " (select distinct
d.repository_id,r.total_contributions,d.category"
                          + " from
repository_useful_data_for_indicators as r join"
                          + "
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing as d on d.repository_id=r.repository_id"
                          + " order by
d.repository_id,r.total_contributions,d.category) group by
category order by category";

                stmtCreate.executeUpdate(sqlCreate);

          } catch (SQLException e) {
                e.printStackTrace();
          }
     }

     private void categories_subscribers() {
          try {

                Statement stmtCreate = conn.createStatement();
                String sqlCreate = "create view
categories_subscribers as"
                          + " select category,
sum(subscribers_count) as subscribers from"
                          + " (select distinct
d.repository_id,r.subscribers_count,d.category"
                          + " from
repository_useful_data_for_indicators as r join"
```

```
                              + "
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing as d on d.repository_id=r.repository_id"
                              + " order by
d.repository_id,r.subscribers_count,d.category) group by
category order by category";

                    stmtCreate.executeUpdate(sqlCreate);

               } catch (SQLException e) {
                    e.printStackTrace();
               }
          }

          private void categories_madurity_total() {
               try {

                    Statement stmtCreate = conn.createStatement();
                    String sqlCreate = "create view
categories_madurity_total as"
                              + " select category, round(sum(
(strftime('%s','2015-11-24 19:19:39 ')-
strftime('%s',created_at))/((strftime('%s','2015-11-24
19:19:39 ')-strftime('%s',updated_at))*1.0)),3) as
madurity_total"
                              + " from (select distinct
d.repository_id,r.created_at,r.updated_at,d.category from
repository_useful_data_for_indicators as r join
datasets_categorized_referenced_and_distinct_repository_id_ref
erencing as d on d.repository_id=r.repository_id order by
d.repository_id,r.created_at,r.updated_at,d.category) group by
category order by category";

                    stmtCreate.executeUpdate(sqlCreate);

               } catch (SQLException e) {
                    e.printStackTrace();
               }
          }
```

Anexo 2 – Clase FirstMatrixValues.java

```
package com.popframework.unex.datamodel;
```

```
public class FirstMatrixValues {

	public CollectionValue
total_references_category_datasets;
	public CollectionValue datasets_references_ornot_github;
	public CollectionValue datasets_not_references_github;
	public CollectionValue datasets_references_github;
	public CollectionValue
distinct_repositories_referencing_category;

	public FirstMatrixValues(){

	}

	public CollectionValue
getTotal_references_category_datasets() {
		return total_references_category_datasets;
	}

	public void
setTotal_references_category_datasets(CollectionValue
total_references_category_datasets) {
		this.total_references_category_datasets =
total_references_category_datasets;
	}

	public CollectionValue
getDatasets_references_ornot_github() {
		return datasets_references_ornot_github;
	}

	public void
setDatasets_references_ornot_github(CollectionValue
datasets_references_ornot_github) {
		this.datasets_references_ornot_github =
datasets_references_ornot_github;
	}

	public CollectionValue
getDatasets_not_references_github() {
		return datasets_not_references_github;
	}

	public void
setDatasets_not_references_github(CollectionValue
datasets_not_references_github) {
		this.datasets_not_references_github =
```

```
datasets_not_references_github;
	}

	public CollectionValue getDatasets_references_github() {
		return datasets_references_github;
	}

	public void
setDatasets_references_github(CollectionValue
datasets_references_github) {
		this.datasets_references_github =
datasets_references_github;
	}

	public CollectionValue
getDistinct_repositories_referencing_category() {
		return distinct_repositories_referencing_category;
	}

	public void
setDistinct_repositories_referencing_category(CollectionValue
distinct_repositories_referencing_category) {
		this.distinct_repositories_referencing_category =
distinct_repositories_referencing_category;
	}



}
```

Anexo 3 – Clase SecondMatrixValues.java

```
package com.popframework.unex.datamodel;

public class SecondMatrixValues {

	private CollectionValue contributors;
	private CollectionValue contributions;
	private CollectionValue subscribers;
	private CollectionValue maturity;

	public SecondMatrixValues(){

	}

	public CollectionValue getContributors() {
		return contributors;
	}

	public void setContributors(CollectionValue
contributors) {
		this.contributors = contributors;
	}

	public CollectionValue getContributions() {
		return contributions;
	}

	public void setContributions(CollectionValue
contributions) {
		this.contributions = contributions;
	}

	public CollectionValue getSubscribers() {
		return subscribers;
	}

	public void setSubscribers(CollectionValue subscribers)
{
		this.subscribers = subscribers;
	}

	public CollectionValue getMaturity() {
		return maturity;
	}

	public void setMaturity(CollectionValue maturity) {
		this.maturity = maturity;
```

```
	}


}
```

Anexo 4 – Clase CollectionValues.java

```
package com.popframework.unex.datamodel;

public class CollectionValue {

	private double administration_finances;
	private double business;
	private double demographics;
	private double education;
	private double ethics_democracy;
	private double geospatial;
	private double health;
	private double recreation_culture;
	private double safety;
	private double services;
	private double sustainability;
	private double transport_infrastructure;
	private double urban_planning_housing;
	private double welfare;

	public CollectionValue(){}

	public double getAdministration_finances() {
		return administration_finances;
	}
	public void setAdministration_finances(double
administration_finances) {
		this.administration_finances =
administration_finances;
	}
	public double getBusiness() {
		return business;
	}
	public void setBusiness(double business) {
		this.business = business;
	}
	public double getDemographics() {
		return demographics;
	}
	public void setDemographics(double demographics) {
```

```
            this.demographics = demographics;
      }
      public double getEducation() {
            return education;
      }
      public void setEducation(double education) {
            this.education = education;
      }
      public double getEthics_democracy() {
            return ethics_democracy;
      }
      public void setEthics_democracy(double ethics_democracy)
{
            this.ethics_democracy = ethics_democracy;
      }
      public double getGeospatial() {
            return geospatial;
      }
      public void setGeospatial(double geospatial) {
            this.geospatial = geospatial;
      }
      public double getHealth() {
            return health;
      }
      public void setHealth(double health) {
            this.health = health;
      }
      public double getRecreation_culture() {
            return recreation_culture;
      }
      public void setRecreation_culture(double
recreation_culture) {
            this.recreation_culture = recreation_culture;
      }
      public double getSafety() {
            return safety;
      }
      public void setSafety(double safety) {
            this.safety = safety;
      }
      public double getServices() {
            return services;
      }
      public void setServices(double services) {
            this.services = services;
      }
      public double getSustainability() {
```

```
            return sustainability;
      }
      public void setSustainability(double sustainability) {
            this.sustainability = sustainability;
      }
      public double getTransport_infrastructure() {
            return transport_infrastructure;
      }
      public void setTransport_infrastructure(double
transport_infrastructure) {
            this.transport_infrastructure =
transport_infrastructure;
      }
      public double getUrban_planning_housing() {
            return urban_planning_housing;
      }
      public void setUrban_planning_housing(double
urban_planning_housing) {
            this.urban_planning_housing =
urban_planning_housing;
      }
      public double getWelfare() {
            return welfare;
      }
      public void setWelfare(double welfare) {
            this.welfare = welfare;
      }



}
```

10. BIBLIOGRAFÍA

1.- Definición de Smart City.

https://es.wikipedia.org/wiki/Ciudad_inteligente

2.- Saaty, T.L. 1980. *The Analytic Hierarchy Process*. McGraw-Hill.

3.- Portal Web.

http://rua.ua.es/dspace/handle/10045/16995

4.- Apps Nativas vs Apps Híbridas vs Apps Webs.
http://www.raona.com/aplicacion-nativa-web-hibrida/

5.- Sitio web de WordNet.

https://wordnet.princeton.edu/

6.- Algoritmo Wu & Palmer

https://dl.acm.org/citation.cfm?id=981751

7.- Algoritmo Leacock-Chodorow
https://www.researchgate.net/profile/Jer_Hayes/publication/220837848_An_Intrinsic_Information_Content_Metric_for_Semantic_Similarity_in_WordNet/links/0fcfd50a2122fa6910000000.pdf

8.- Sitio web con la implementación de los algoritmos para Wordnet.
https://code.google.com/archive/p/ws4j/

9.- Algorimo Jaro-Winkler
https://www.cs.cmu.edu/afs/cs/Web/People/wcohen/postscript/kdd-2003-match-ws.pdf

10.- Sitio web con la implementación del algoritmo Jaro-Winkler.
https://github.com/rrice/java-string-similarity

11.- Tipos de aplicaciones web.

https://es.linkedin.com/pulse/6-tipos-de-aplicaciones-web-hector-badal-mba

12.- Arquitectura de una aplicación web.
https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/

13.- REST.

https://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html

14.- Modelo Vista Presentador

https://msdn.microsoft.com/en-us/library/ff649571.aspx

15.- Modelo Vista Vista-Modelo

https://msdn.microsoft.com/en-us/library/hh848246.aspx

16.- Model Vista Controlador

https://msdn.microsoft.com/en-us/library/ff649643.aspx

17.- Spring Framework

https://spring.io/

18.- Apache Maven

https://maven.apache.org/

19.- Gradle Build Tool

https://gradle.org/

20.- IntelliJ

https://www.jetbrains.com/idea/

21.- Eclipse IDE

https://www.eclipse.org/

22.- Spring Tool Suite

https://spring.io/tools

23.- Página de descarga de Eclipse IDE

https://www.eclipse.org/downloads/

24.- Spring Initializr

https://start.spring.io/

25.- Vue.js

https://vuejs.org/

26.- React

https://reactjs.org/

27.- Ember.js

https://www.emberjs.com/

28.- Meteor

https://www.meteor.com/

29.- AngularJS

https://angularjs.org/

30.- Angular

https://angular.io/

31.- TypeScript

https://www.typescriptlang.org/

32.- Visual Studio Code

https://code.visualstudio.com/

33.- NodeJS

https://nodejs.org/es/

34.- Angular CLI

https://cli.angular.io/

35.- Buenas prácticas en Angular

https://angular.io/guide/styleguide

36.- Bootstrap 3

https://getbootstrap.com/docs/3.3/

37.- Sass

https://sass-lang.com/

38.- CSS Flex

https://css-tricks.com/snippets/css/a-guide-to-flexbox/

39.- Librerías externas para Angular

https://blog.bitsrc.io/11-angular-component-libraries-you-should-know-in-2018-e9f9c9d544ff