



POLITÉCNICA
"Ingeniamos el futuro"

CAMPUS
DE EXCELENCIA
INTERNACIONAL



Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Integración y puesta en valor de conjuntos de datos abiertos

Autor: Jesús Pérez Melero

Director: Raúl García Castro

MADRID, ENERO 2018

Scientia potentia est

F. Bacon

Agradecimientos

El presente trabajo ha sido un gran reto para mí en los últimos meses. Finalizarlo me ha costado mucho tiempo y esfuerzo, pero no ha sido nada en comparación con la escritura de estas líneas.

Quiero dedicar este trabajo a mis abuelos, especialmente a Antonio, quién me enseñó a apreciar el valor y la belleza de la naturaleza que nos rodea. He tratado de poner lo mejor de mí mismo en la realización de este proyecto, aun siendo consciente de que no podrá ver los resultados obtenidos, con la intención de que se sintiese orgulloso no solo de este trabajo, sino también de mí.

Si bien nunca me consideré un buen estudiante, todo cambió el día que entré en la UPM. Gracias a la universidad he podido confirmar mi vocación hacia el mundo tecnológico. Este trabajo es la prueba de mi dedicación, esfuerzo y sacrificio durante cuatro años. Quiero también dar las gracias a Raúl, mi tutor durante estos meses.

Este proyecto es el final de una etapa, y estoy profundamente agradecido por todas aquellas personas que me han ayudado a superarla. En primer lugar, a mis padres, por todo su apoyo y comprensión durante estos cuatro años que, sin duda, han sido difíciles. A mis hermanas, por su incansable paciencia y dedicación conmigo, ya que gracias a ellas y a sus clases conseguí entrar en la universidad.

En segundo lugar quiero agradecer a dos personas el tiempo que han pasado conmigo estos años: Sergio y Daniel. Con ellos he vivido momentos increíbles y han sido el mayor apoyo que he podido tener en los peores días de la vida universitaria.

Por último, quiero dar las gracias a Eduardo, Carlos, Christian, Carlos, Diego y Ángela, por su invaluable apoyo, cariño y comprensión en estos meses que no han sido fáciles.

Madrid, Enero de 2018

Abstract

This final degree Project involves the realization of a system that takes as input different sets of open data from the open data portal of the city of Madrid. The information contained in these data sets will be enriched with information in other sources such as DBpedia, esDBpedia and OpenStreetMap. Subsequently, it will be published following the linked data principles, specifically in the RDF format.

Thanks to the linked data, structured and interconnected information can be published, which besides serving to nourish web pages for human use, extends its use to automatic reading by machines. This is a fundamental pillar of the Semantic Web.

The generated RDF files will be the base of Madrid on You, an application which allows automatic random tours of the city of Madrid, most of which are of a tourist and cultural nature.

Note that all the resources used in the realization of this project (software, documentation, works, etc.) have free licenses.

Resumen

El presente Trabajo Fin de Grado consiste en la realización de un sistema que tome como entrada distintos conjuntos de datos abiertos del portal de datos abiertos de la ciudad de Madrid. La información contenida en dichos conjuntos de datos se enriquecerá con información existente en otras fuentes como *DBpedia*, *esDBpedia* y *OpenStreetMap*. Posteriormente, se publicará siguiendo los principios de los datos enlazados, concretamente en el formato RDF.

Los datos enlazados¹ constituyen un método de publicación de información estructurada e interconectada, que además de servir para nutrir páginas web para uso de los humanos, extiende su uso a la lectura automática por parte de las máquinas, siendo esto un pilar fundamental de la Web Semántica.

Los ficheros RDF generados serán la base de la aplicación *Madrid on You*, que permite realizar de forma automática recorridos aleatorios por la ciudad de Madrid, la mayoría de índole turístico y cultural.

Nótese que todos los recursos utilizados en la realización de este proyecto (software, documentación, obras, etc.) constan de licencias libres.

Índice general

Capítulo 1

Introducción	1
--------------------	---

Capítulo 2

Estudio del dominio	2
---------------------------	---

2.1. Datos abiertos	2
2.2. Datos enlazados	3
2.3. La Web Semántica	5
2.4. Resource Description Framework (RDF)	6
2.5. SPARQL	10
2.6. Ciclo de vida de los datos enlazados	16
2.7. Proceso de generación de datos enlazados.....	18
2.8. Ontologías	20

Capítulo 3

Análisis de sistemas relevantes	21
---------------------------------------	----

3.1. Fuentes de información.....	21
3.1.1. Portal de datos abiertos de Madrid.....	21
3.1.2. DBpedia y esDBpedia	22
3.1.3. Open Street Map	22
3.2. Modificación y generación de datos enlazados	24
3.2.1. LOD Refine.....	24
3.2.2. Protégé	25
3.3. Desarrollo y despliegue.....	26

3.3.1. Spring Framework.....	26
3.3.2. Amazon Web Services	27
3.4. Otros.....	28
3.4.1. Apache JENA.....	28

Capítulo 4

Especificación de requisitos	29
4.1. Propósito	29
4.2. Ámbito	29
4.3. Perspectiva del producto	30
4.4. Requisitos específicos.....	30
4.5.1. Feature #1 – Inicio de sesión	31
4.5.2. Feature #2 – Registro en la aplicación	32
4.5.3. Feature #3 – Actualización del perfil	33
4.5.4. Feature #3 – Atributos.....	34
4.5.5. Feature #3 – Rendimiento.....	35
4.5.6. Feature #4 – Generación de rutas.....	36
4.5.7. Feature 5# - Visualización de rutas	37

Capítulo 5

Diseño e implementación	38
5.1. Introducción	38
5.2. Arquitectura del sistema	38
5.3. Front – End.....	41
5.3.1. Vistas	42
5.3.2. JavaScript.....	53
5.3.3. Hojas de estilo CSS	58
5.4. Back – End	63
5.4.1. User Management	63

5.4.2. Buildings Management.....	68
5.4.3. Dispatcher	78
5.4.3. Mail Engine.....	79
5.5. Ontología.....	81
5.5.1. Clases.....	82
5.5.2. Relaciones.....	82
5.5.3. Propiedades.....	83
5.6. Ficheros RDF	84

Capítulo 6

Casos de uso	86
6.1. Actores	86
6.2. Casos de uso	86
6.2.1. Caso de uso #1: Login en la aplicación.....	87
6.2.2. Caso de uso #2: Registro en la aplicación	88
6.2.3. Caso de uso #3: Actualización del perfil.....	89
6.2.4. Caso de uso #4: Generación de rutas	90

Capítulo 7

Evaluación del sistema	91
7.1. Pruebas unitarias.....	91
7.2. Pruebas funcionales	92
7.2.1. Pruebas funcionales de gestión de usuarios.....	92
7.2.2. Pruebas funcionales de generación de rutas	97

Capítulo 8

Usabilidad.....	102
------------------------	------------

8.1. Resultados de la evaluación	104
--	-----

Capítulo 9

Conclusiones.....	107
--------------------------	------------

9.1. Concurso de datos abiertos de Madrid.....	109
--	-----

9.2 Aplicación a la docencia	110
------------------------------------	-----

Capítulo 10

Líneas futuras	111
-----------------------------	------------

12.1. Front-End	111
-----------------------	-----

12.2. Back-End.....	112
---------------------	-----

12.3. Ontología y RDF	113
-----------------------------	-----

Bibliografía.....	114
--------------------------	------------

ANEXO I: Distribución de código.....	116
---	------------

ANEXO II: Licencia	117
---------------------------------	------------

Índice de ilustraciones

Ilustración 1 - LOD Cloud.....	4
Ilustración 2 - Tripleta RDF simple	6
Ilustración 3 - Tripleta RDF compuesta.....	6
Ilustración 4 - Prefijos en un RDF	7
Ilustración 5 - RDF en formato TTL.....	8
Ilustración 6 - RDF en formato XML	9
Ilustración 7 - Prefijos en SPARQL.....	10
Ilustración 8 - Query simple en SPARQL.....	10
Ilustración 9 - Ordenación y limitación de resultados en SPARQL	11
Ilustración 10 - Query simple de ejemplo completa	11
Ilustración 11 - Optional SPARQL.....	12
Ilustración 12 - Cláusula SERVICE para queries federadas	12
Ilustración 13 - Query compleja en SPARQL	14
Ilustración 14 - Ciclo de vida de los datos enlazados	16
Ilustración 15 - Proceso de generación de datos enlazados	18
Ilustración 16 - Open Street Map.....	23
Ilustración 17 - GREL.....	24
Ilustración 18 - Enlazado de datos con LOD Refine	25
Ilustración 19 - Logo de Spring	26
Ilustración 20 - Logo de AWS.....	27
Ilustración 21 - Arquitectura de JENA	28
Ilustración 22 - Arquitectura de la aplicación	38
Ilustración 23 - Vista index.html (PC)	43
Ilustración 24 - Vista home.html (PC)	44
Ilustración 25 - Vista profile.html (PC)	45
Ilustración 26 - Vista tourist.html (PC).....	46
Ilustración 27 - Vista Map-osm.html - Elementos de la ruta (PC)	47
Ilustración 28 - Vista Map-osm.html - Elemento de la ruta (PC).....	48
Ilustración 29 - Vista Map-osm.html - Farmacias cercanas al elemento de la ruta (PC)	49

Ilustración 30 - Vista Map-osm.html - Teatros cercanos a varios elementos (PC)	51
Ilustración 31 - Función initmap.....	53
Ilustración 32 – Añadir elementos al mapa (fragmento de AskForPlots).....	54
Ilustración 33 - Código de la petición AJAX de inicio de sesión.....	55
Ilustración 34 - Código de la petición AJAX de registro.....	56
Ilustración 35 - Código de la petición AJAX de actualización	57
Ilustración 36 - Vista de perfil (Tableta).....	59
Ilustración 37 - Vista de perfil (Móvil)	59
Ilustración 38 - Vista de definición de ruta (Tableta)	60
Ilustración 39 - Vista de definición de ruta (Móvil)	60
Ilustración 40 - Mapa con los elementos de la ruta (Tableta)	61
Ilustración 41 - Mapa con los elementos de la ruta (Móvil)	61
Ilustración 42 - Elemento de la ruta (Tableta).....	62
Ilustración 43 - Elemento de la ruta (Móvil)	62
Ilustración 44 - Estructura del Micro de User Management	63
Ilustración 45 - Input para login	64
Ilustración 46 - Input para register.....	64
Ilustración 47 - Clases utilizadas en User Management	65
Ilustración 48 - UML de la clase que interactúa con Dynamo DB.....	66
Ilustración 49 - Tabla en Dynamo DB	66
Ilustración 50 - Diagrama UML de clases de User Management.....	67
Ilustración 51 - Estructura del micro de Buildings Management	68
Ilustración 52 - Input para random	69
Ilustración 53 - UML de las clases que realizan queries contra los RDF y DBpedia.....	70
Ilustración 54 - UML de las clases que interactúan con OSM	71
Ilustración 55 - UML de las clases necesarias para mapear la respuesta de OSM	72
Ilustración 56 - Diagrama UML de clases de Buildings Management	73
Ilustración 57 - Consulta SPARQL	74
Ilustración 58 - Script XML para la API Overpass	75
Ilustración 59 - Diagrama UML de clases relativas a integración con DBpedia y OSM	77
Ilustración 60 - Estructura del proyecto mail Engine	79

Ilustración 61 - UML de las clases de Mail Engine.....	80
Ilustración 62 - Ontología desarrollada.....	81
Ilustración 63 - Clases de la ontología	82
Ilustración 64 - Relaciones de la ontología	82
Ilustración 65 - Propiedades de la ontología	83
Ilustración 66 - Proceso de generación de ficheros RDF.....	84
Ilustración 67 - Diagrama de secuencia del caso de uso #1	87
Ilustración 68 - Diagrama de secuencia del caso de uso #2	88
Ilustración 69 - Diagrama de secuencia del caso de uso #3	89
Ilustración 70 - Diagrama de secuencia del caso de uso #4	90
Ilustración 73 - Enriquecimiento progresivo de los datos.....	108
Ilustración 74 - Presentación de la app en Medialab Prado (Madrid).....	109
Ilustración 75 - Fragmento del mail enviado a alumnos	110

Índice de tablas

Tabla 1 - Resultado de la consulta SPARQL	12
Tabla 2 - Resultados de la query SPARQL avanzada.....	15
Tabla 3 - API de User Management	64
Tabla 4 – API de Buildings Management	69
Tabla 5 - API del dispatcher	78
Tabla 6 - Api de Mail Engine	79
Tabla 7 - Caso de uso #1	87
Tabla 8 - Caso de uso #2	88
Tabla 9 - Caso de uso #3	89
Tabla 10 - Caso de uso #4	90
Tabla 11 - Prueba funcional 01.....	92
Tabla 12 - Prueba funcional 02.....	93
Tabla 13 - Prueba funcional 03.....	94
Tabla 14 - Prueba funcional 04.....	95
Tabla 15 - Prueba funcional 05.....	96
Tabla 16 - Prueba funcional 06.....	96
Tabla 17 - Prueba funcional 07.....	97
Tabla 18 - Prueba funcional 08.....	98
Tabla 19 - Prueba funcional 09.....	99
Tabla 20 - Prueba funcional 10.....	100
Tabla 21 - Prueba funcional 11.....	101
Tabla 22 - Formulario de resultados de usabilidad.....	104
Tabla 23 - Resultados del primer usuario.....	104
Tabla 24 - Resultados del segundo usuario	105
Tabla 25 - Resultados del tercer usuario	105
Tabla 26 - Resultados del cuarto usuario	106

Capítulo 1

Introducción

Actualmente podemos ver una gran cantidad de apps en la *Apple Store* o en *Google Play Store* que ofrecen algún tipo de información sobre la comunidad de Madrid, ya sea turística, de ocio o de transporte. Estas apps son usadas ampliamente por mucha gente. Es muy común ver a las personas mirando cuándo llegará el siguiente tren o bus a través de una app en su smartphone, o ver a turistas visitando edificios monumentales a través de una ruta que una app ha sugerido. Estas aplicaciones suelen tener detrás una base de datos monolítica con una ingente cantidad de información, la cual es utilizada para ser mostrada al usuario.

Aunque esta práctica es útil para hacer que las apps funcionen correctamente, existen otras alternativas, como los datos abiertos enlazados, que son probablemente una mejor opción. Usar una base de datos monolítica como fuente de datos tiene varios problemas. Por ejemplo, la app fallará si la base de datos está fuera de servicio. Pero eso no es todo. La información de la base de datos debe estar actualizada y validada. Esto es un problema para el dueño de la app ya que debe realizar dicha actualización manualmente, a veces en cortos períodos de tiempo. Las bases de datos abiertos enlazados también necesitan ser actualizadas y validadas, pero en este caso, el problema recae en la persona que publica los datos.

Estos problemas podrían solventarse si los desarrolladores utilizasen datos abiertos enlazados¹, proporcionados por las autoridades oficiales de la comunidad. Los datos abiertos enlazados no son datos comunes. Estos datos deben ser públicos y estar enlazados con otros conjuntos de datos de manera que la información pueda ser enriquecida.

Capítulo 2

Estudio del dominio

En este capítulo se presentan los conceptos básicos que se van a tratar en la realización del presente trabajo fin de grado, así como sus características principales.

2.1. Datos abiertos

Los datos abiertos² (*open data*) forman parte de una filosofía que ha ido tomando cada vez más importancia en nuestro tiempo. Esta filosofía tiene como objetivo principal que ciertos tipos de datos estén disponibles y accesibles para todo el mundo, sin ningún tipo de restricción de derechos de autor, patentes u otros mecanismos de control. Este movimiento se identifica con otros similares como pueden ser el del software libre o el de código abierto (*open source*).

Para que los datos puedan considerarse *abiertos*, deben ser accesibles y reutilizables sin necesidad de ningún tipo de permiso. Habitualmente los datos abiertos contienen información sobre geografía, cartografía, meteorología, datos médicos, contabilidad... Históricamente esos datos han sido propiedad de organizaciones públicas o privadas pero ahora gracias a este movimiento esos datos se han vuelto accesibles, para la vista y consumo de todos.

Con la intención de sumarse a este movimiento, el gobierno de España tiene habilitado un portal, de carácter nacional, con información de diversos tipos, accesible y reutilizable en *datos.gob.es*.

Por su parte, la ciudad de Madrid también cuenta con un portal de datos abiertos, donde se pueden encontrar conjuntos de datos de muchos tipos, como por ejemplo ciencia, demografía, deporte, economía, transporte, seguridad o empleo. Todo ello a través de la página *datos.madrid.es/portal/site/egob*. Para la realización de este trabajo se han obtenido como primera fuente de información 4 conjuntos de datos del portal de datos

abiertos de Madrid, todos pertenecientes a la categoría de *Cultura y Ocio*. Los conjuntos en cuestión son:

- **Monumentos o edificios de carácter monumental**
- **Bibliotecas**
- **Museos**
- **Templos católicos**

2.2. Datos enlazados

Actualmente multitud de datos son expuestos en la web en formatos muy conocidos como pueden ser HTML, PDF, XML, etc. Las páginas web y los servidores se nutren de bases de datos de distinta índole para proporcionar dichos datos a los usuarios, formando así los documentos que todos conocemos y dando lugar a la **web 2.0**³. Sin embargo, si queremos realizar consultas complejas sobre varias páginas o fuentes de datos, encontramos muchas dificultades.

Una característica clave de los datos enlazados es **utilizar la web como una única base de datos global**, de la cual poder leer y escribir información, convirtiendo así la web de documentos en una web de datos. De acuerdo con *Tim Berners Lee*⁴, los cuatro principios de los datos enlazados son:

- **Utilizar URIs como nombres** para las cosas.
- **Utilizar URIs HTTP para que la gente puede obtener información** a través de esos nombres.
- **Utilizar tecnologías como SPARQL o RDFs para obtener información útil** cuando se accede a una URI.
- **Utilizar referencias a otras URIs, cuya información sea valiosa para el propio recurso**, de forma que se cree un enlace y la información de un extremo se enriquezca con la información del otro extremo del enlace.

De esta forma y siguiendo estos cuatro principios, los datos enlazados posibilitan la creación de la web semántica, que es a su vez una parte fundamental de la **web 3.0**⁵. Los datos enlazados son el mecanismo utilizado para vincular los distintos elementos que

están distribuidos en la web, de forma que puedan referenciarse unos a otros de la misma forma en la que lo harían mediante enlaces estáticos en páginas web convencionales.

La web de datos sólo puede ser creada mediante el uso de datos enlazados, ya que es el único mecanismo para conseguir una gran base de datos en la cual todos los nodos estén conectados unos con otros y además distribuidos, al estar en un entorno web. Cuando esto se consigue, se crea lo que se conoce como **LOD Cloud**⁶ (*Linked Open Data Cloud*).

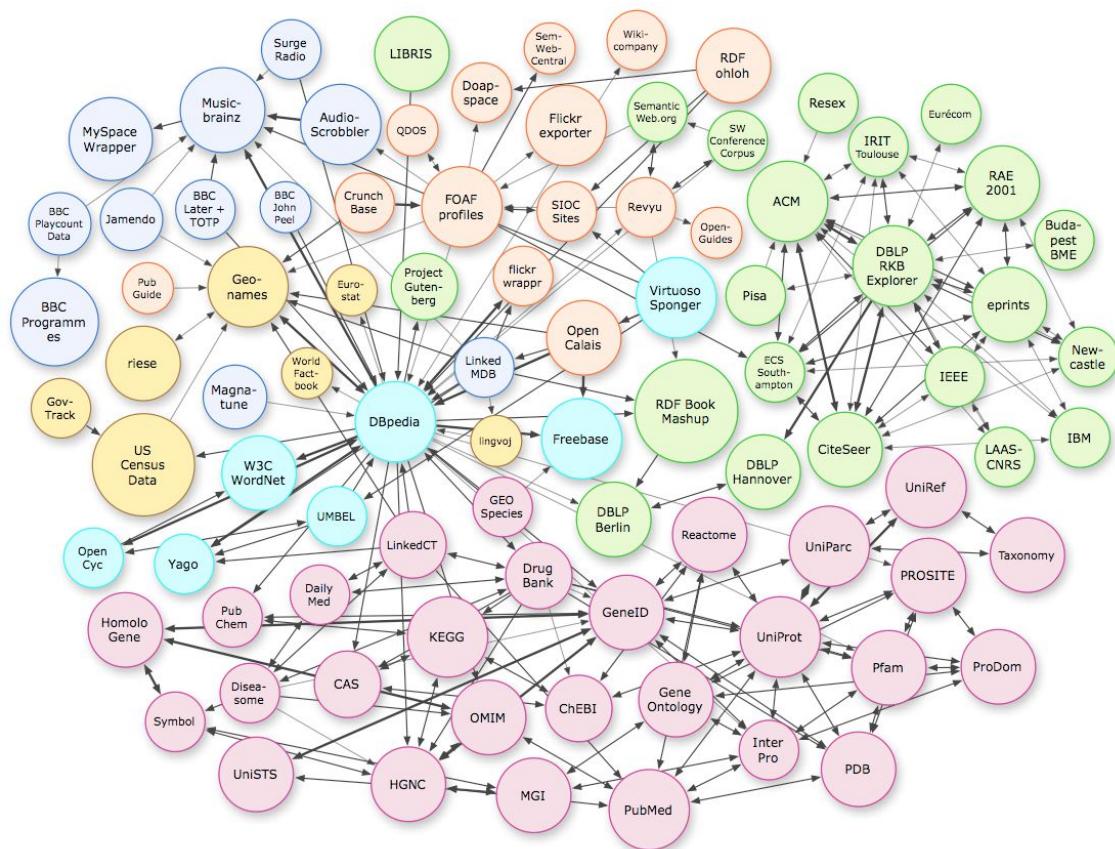


Ilustración 1 - LOD Cloud

2.3. La Web Semántica

La web semántica⁷ es una extensión de la web actual, que se basa en dos grandes pilares:

- **La descripción** del significado
- **La manipulación automática** de dichas descripciones mediante lógica y motores de inferencia.

La descripción del significado pasa por utilizar la semántica, los metadatos y las ontologías:

- **La semántica persigue el estudio del significado de los términos lingüísticos.**
La web semántica busca dotar de significado a la información existente en la web. Este significado debe ser interpretable por las máquinas, siendo así necesario que la información adicional que se añada pueda ser procesada por un computador.
- **Los metadatos son datos que describen otros datos.** En el contexto de la web semántica, son los datos que describen los recursos que hay en la web.
- **Las ontologías son jerarquías de conceptos, que contienen atributos y relaciones.** Definen una terminología concreta para definir redes semánticas.

Algunas de las ventajas de la web semántica son:

- **Incorporación de contenido semántico a los datos** que existen en la web. Esto hace posible búsquedas más precisas basadas en significado y no en contenido.
- **Hace posible que los computadores gestionen el conocimiento** (mediante inteligencia artificial).

Algunas desventajas de la web semántica son:

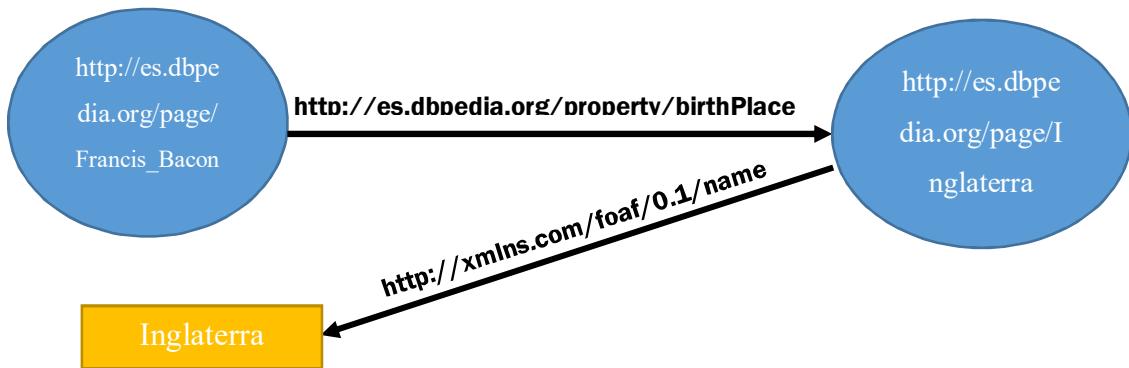
- **Adaptar todos los documentos** de la web para que sean procesados de forma semántica es muy costoso.
- **La necesidad de una unificación de términos junto con la definición de estándares semánticos**, para evitar posibles conflictos entre idiomas (por ejemplo, definir que IVA es igual a VAT).

2.4. Resource Description Framework (RDF)

RDF⁸ es una familia de especificaciones propuesta por la W3C, con la idea original de ser un modelo de datos para metadatos. Establece una serie de normas para clasificar la información dentro de un documento utilizando etiquetas, propiedades y relaciones entre la información, haciendo así posible que sea comprensible por las máquinas. Los ficheros RDF contienen triplets. Una tripla es un conjunto de 3 elementos: un sujeto, un predicado y un objeto, relacionados entre sí. Un ejemplo es el siguiente:



En esta tripla podemos ver que el sujeto viene definido por un recurso web de esDBpedia que contiene información sobre el abogado, político y filósofo inglés Francis Bacon. El predicado de la tripla es <http://es.dbpedia.org/property/name>, un recurso de esDBpedia que hace referencia a la propiedad *nombre* que tienen las cosas. Por último, el objeto es en este caso un literal de tipo String, con el nombre de la persona en cuestión, Francis Bacon. Las triplets pueden ser más complejas ya que los objetos no tienen por qué ser literales. Podrían ser los sujetos de una nueva tripla, como ocurre en este caso:



En este caso vemos como ahora el objeto de la primera tripla es el sujeto de la segunda, que tiene un predicado que indica que el nombre del lugar de nacimiento de Francis Bacon es Inglaterra.

La modelación y descripción de los datos es un proceso complejo ya que la información se modela en un instante de tiempo concreto pero es posible que dicha información cambie en el futuro, siendo así necesario redefinir el modelo. Es necesario por tanto un estudio y esfuerzo considerables para producir un modelo útil que permita obtener los beneficios inherentes a los datos enlazados.

Los ficheros RDF pueden encontrarse bajo varias extensiones, como XML o TTL. A la hora de producir ficheros RDF, se pueden utilizar prefijos que servirán para clarificar la especificación. En los sucesivos ejemplos, se usarán los siguientes prefijos:

```
@prefix dbo: <http://dbpedia.org/ontology/> .  
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .  
@prefix dbp: <http://dbpedia.org/property/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix owl: <http://www.w3.org/2002/07/owl#> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
```

Ilustración 4 - Prefijos en un RDF

Como se puede observar, los prefijos se declaran con la anotación `@prefix` seguida del nombre que se le quiere dar al prefijo. Después, se indica la cadena de caracteres que sustituirá al prefijo cada vez que se escriba. Un ejemplo de RDF en formato TTL es el siguiente:

```
<http://madridonyou.com/resources/285da2198b2b496c9d447cc4ac6b0734> a dbo:HistoricBuilding ;  
  
dbo:name "Basilica Pontificia de San Miguel"^^xsd:string ;  
  
dbo:address "calle san justo 4"^^xsd:string ;  
  
dbp:phoneNumber "915 484 011"^^xsd:string ;  
  
geo:lat "40.414203159177625"^^xsd:float ;  
  
geo:long "-3.7096715906562716"^^xsd:float ;  
  
owl:sameAs  
<http://dbpedia.org/resource/St.\_Michael's\_Basilica\_\(Madrid\)> .
```

Ilustración 5 - RDF en formato TTL

El RDF de la ilustración contiene una descripción del recurso **285 [...] 734**, indicando que se trata de un recurso del tipo *HistoricBuilding*, por lo que probablemente sea algún tipo de monumento o edificio de carácter histórico o monumental.

Bajo la definición del recurso y del tipo del recurso, se encuentran las propiedades que tiene asociado el mismo. Existen dos tipos de propiedades, de datos y de objetos. Las propiedades de datos relacionan entidades con valores literales, como por ejemplo la propiedad *name*, que relaciona una entidad del tipo *HistoricBuilding* con un literal, en este caso de tipo String, que contiene el nombre del recurso. Podemos encontrar más propiedades de este tipo, como *address*, *phoneNumber*, *lat*, *long* y *sameAs*.

A continuación se muestra el mismo recurso pero en un RDF bajo el formato XML. Como se puede observar cambia ligeramente la notación y la sintaxis con respecto al método anterior. El uso de uno u otro tipo de RDF dependerá del uso que se le quiera dar a los datos y de cómo los maneje la aplicación que los use.

```
<rdf:Description  
    rdf:about="http://madridonyou.es/resources/285da2198b2b496c9d447cc4ac6b0734">  
    <rdf:type  
        rdf:resource="http://dbpedia.org/ontology/HistoricBuilding" />  
        <dbo:name  
            rdf:datatype="http://www.w3.org/2001/XMLSchema#string">Basilica Pontificia de San Miguel</dbo:name>  
  
        <owl:sameAs  
            rdf:resource="http://dbpedia.org/resource/St.\_Michael's\_Basilica\_\(Madrid\)" />  
  
        <dbo:address  
            rdf:datatype="http://www.w3.org/2001/XMLSchema#string">calle san justo 4</dbo:address>  
  
        <geo:lat  
            rdf:datatype="http://www.w3.org/2001/XMLSchema#float">40.414203159177625</geo:lat>  
  
        <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#float">-3.7096715906562716</geo:long>  
  
        <dbp:phoneNumber  
            rdf:datatype="http://www.w3.org/2001/XMLSchema#string">915 484 011</dbp:phoneNumber>
```

Ilustración 6 - RDF en formato XML

2.5. SPARQL

SPARQL⁹ (*SPARQL Protocol and RDF Query Language*) es un lenguaje utilizado para la consulta de grafos RDF, normalizado por la W3C. Se trata de una parte clave de la web semántica. Una de sus características más atractivas es que **permite utilizar varias fuentes de datos**, lo que hace que las búsquedas sean más amplias y complejas.

Su sintaxis es similar a la de SQL, haciendo uso de palabras clave como *SELECT*, *FROM*, *WHERE*, *FILTER*, *ORDER BY*, *LIMIT*... sin olvidar que **cuenta con una capa semántica de la que SQL carece**.

Al igual que con los RDF, es posible definir prefijos para hacer que las queries sean más legibles. La sintaxis es como sigue:

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX db: <http://dbpedia.org/>
```

Ilustración 7 - Prefijos en SPARQL

A la hora de realizar una query es necesario definir qué información se quiere devolver, así como la fuente de los datos. En los sucesivos ejemplos se utiliza como fuente de datos un RDF cuyos recursos son monumentos o edificios de carácter monumental o histórico.

```
SELECT DISTINCT
?name ?address
?latitude ?longitude ?phoneNumber
WHERE {
?a a dbo:HistoricBuilding .
?a dbo:name ?name .
?a dbo:address ?address .
?a geo:lat ?latitude .
?a geo:long ?longitude .
?a dbp:phoneNumber ?phoneNumber .
}
```

Ilustración 8 - Query simple en SPARQL

En la query de la ilustración podemos ver que los valores que se quieren devolver son *name*, *address*, *latitude*, *longitude*, *phoneNumber* y *url*. La primera sentencia del cuerpo de la query restringe los resultados a recursos que sean del tipo *HistoricBuilding*. Después, se busca si existe algún recurso que cumpla todas las reglas escritas, es decir, que contenga las tripletas especificadas. Si es así, devolverá dicho recurso en la respuesta de la query. Además, es posible limitar el número de resultados e indicar el orden en el cual se quiere que se devuelvan.

ORDER BY RAND()

LIMIT 5

Ilustración 9 - Ordenación y limitación de resultados en SPARQL

Juntando todas las partes indicadas previamente (prefijos, variables de retorno, cuerpo de la query y ordenación y limitación) obtenemos la siguiente query:

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX db: <http://dbpedia.org/>

SELECT DISTINCT
?name ?address
?latitude ?longitude ?phoneNumber
WHERE {
?a a dbo:HistoricBuilding .
?a dbo:name ?name .
?a dbo:address ?address .
?a geo:lat ?latitude .
?a geo:long ?longitude .
?a dbp:phoneNumber ?phoneNumber .
}

ORDER BY RAND()
LIMIT 2
```

Ilustración 10 - Query simple de ejemplo completa

Si intentamos ejecutar esta query sobre la fuente de datos indicada anteriormente, obtendremos los siguientes resultados:

name	address	latitude	longitude	phoneNumber
Ermita de Santa María la Antigua	Calle monseñor Oscar Romero 1	40.380299379931 614	-3.741676977905724	"914 628 536 (Parroquia de San Sebastian Martir)"

Tabla 1 - Resultado de la consulta SPARQL

Sin embargo, si en la query anterior no existe alguna de las triplets indicadas para un recurso concreto, no se devolverá dicho recurso. Esto puede ser útil en ciertas ocasiones y un problema en otras. Por ejemplo, supongamos que queremos obtener toda la información posible de un recurso. La query debería contemplar todas y cada una de las propiedades asociadas al tipo de recurso. En SPARQL se puede conseguir este efecto con el operador *OPTIONAL*.

```
OPTIONAL {
    ?a owl:sameAs ?uri
}
```

Ilustración 11 - Optional SPARQL

De esta forma, la variable `?uri` tendría valor asignado únicamente en el caso de que exista una tripla con las partes indicadas.

Como se ha indicado anteriormente, una de las ventajas de SPARQL es que puede integrarse con multitud de conjuntos de datos¹⁰ para obtener más información y enriquecer aquella de la que ya se dispone. Para ello, es necesario realizar **queries federadas** mediante la cláusula *SERVICE* de SPARQL, tal como se muestra a continuación:

```
?a owl:sameAs ?uri
SERVICE <http://es.dbpedia.org/sparql>
{
    ?uri foaf:depiction ?image
```

Ilustración 12 - Cláusula SERVICE para queries federadas

La query federada necesita una URI en la que situarse, a parte de un *endpoint* distinto al de la query en curso.

En este caso dicho *endpoint* es <http://es.dbpedia.org/sparql>, que es el *endpoint* de esDBpedia. Lo que se hace en esa query federada es, a partir de la URI obtenida de la tripleta con predicado *sameAs*, ir a esDBpedia y ver si el recurso identificado por esa URI contiene una tripleta cuyo predicado sea *foaf:depction* (que es la propiedad que almacena una foto del recurso), en cuyo caso devolverá la foto en la variable *image*.

Una vez comentadas las principales funcionalidades de SPARQL, así como alguna más avanzada, como las queries federadas, se muestra a continuación una de las queries que se realiza en el back-end de la aplicación construida:

```

PREFIX dbp: <http://dbpedia.org/property/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX db: <http://dbpedia.org/>

SELECT DISTINCT ?name ?address ?district ?latitude ?longitude
?phoneNumber ?image ?url ?abstract WHERE {

    ?a a dbo:HistoricBuilding .
    ?a dbo:name ?name .
    ?a dbo:address ?address .
    ?a geo:lat ?latitude .
    ?a geo:long ?longitude .
    ?a dbp:phoneNumber ?phoneNumber .
    ?a dbp:url ?url .

    OPTIONAL {
        ?a owl:sameAs ?uri
        OPTIONAL {
            SERVICE <http://dbpedia.org/sparql> {
                ?uri foaf:depiction ?image .
                OPTIONAL {
                    ?uri dbo:abstract ?abstract .
                    FILTER langMatches(lang(?abstract), 'es')
                }
            }
        }
        OPTIONAL {
            SERVICE <http://es.dbpedia.org/sparql> {
                ?uri foaf:depiction ?image
            }
        }
        OPTIONAL {
            SERVICE <http://es.dbpedia.org/sparql> {
                ?uri dbo:wikiPageRedirects ?another .
                ?another foaf:depiction ?image .
            }
        }
    }

    ?a dbo:district ?location .
    ?location dbo:name ?district .
}

ORDER BY RAND()
LIMIT 1

```

Ilustración 13 - Query compleja en SPARQL

La query anterior es una composición de todos los fragmentos SPARQL mostrados hasta ahora. Inicialmente, se restringe la búsqueda a recursos del tipo *HistoricBuilding* y se buscan recursos que contengan las tripletas básicas cuyos predicados son *name*, *address*, *district*, *latitude*, *longitude*, *phoneNumber*, *image*, *url* y *abstract*. Algunas variables no se encuentran dentro de una cláusula OPTIONAL ya que son la información básica inherente a cualquier edificio histórico y no se permite tener un recurso de este tipo sin esta información mínima.

En segundo lugar, se comprueba, dentro de una cláusula *OPTIONAL* si el recurso tiene una tripleta cuyo predicado sea *sameAs*. Si es así, **quiere decir que este recurso está enlazado** con otro existente y por ende se puede obtener información del primero a través del enlace al segundo. Las fuentes de datos externas en todas las queries SPARQL realizadas en la aplicación son DBpedia y esDBpedia. Los campos que se intentan obtener de dichos endpoints son las **imágenes y pequeñas descripciones** de los recursos.

Finalmente, se obtiene el distrito en el cual está situado el recurso. En este ejemplo se ha limitado la salida a un único elemento, y el resultado es el siguiente:

Name	Address	District
Real Academia Nacional de Farmacia	Calle Farmacia 9	CENTRO
Latitude	Longitude	PhoneNumber
40.423966629698484	-3.6997070566405665	915 310 307 / 915 223 147
Image		
http://commons.wikimedia.org/wiki/Special:FilePath/Real_Academia_Nacional_de_Farmacia_(España)_01.jpg		
Url		
http://www.madrid.es/sites/v/index.jsp?vgnextchannel=9e4c43db40317010VgnVCM10000dc0ca8c0RCRD&vgnextoid=3e9bff632081c010VgnVCM1000000b205a0aRCRD		
Abstract		
La Real Academia Nacional de Farmacia de España tiene su origen en 1737, año en el que una real cédula de Felipe V aprueba los estatutos del Real Colegio de Profesores Boticarios de Madrid.		

Tabla 2 - Resultados de la query SPARQL avanzada

2.6. Ciclo de vida de los datos enlazados

Como en la mayoría de áreas del software, y de la informática en general, los datos enlazados están marcados por un ciclo de vida¹¹:

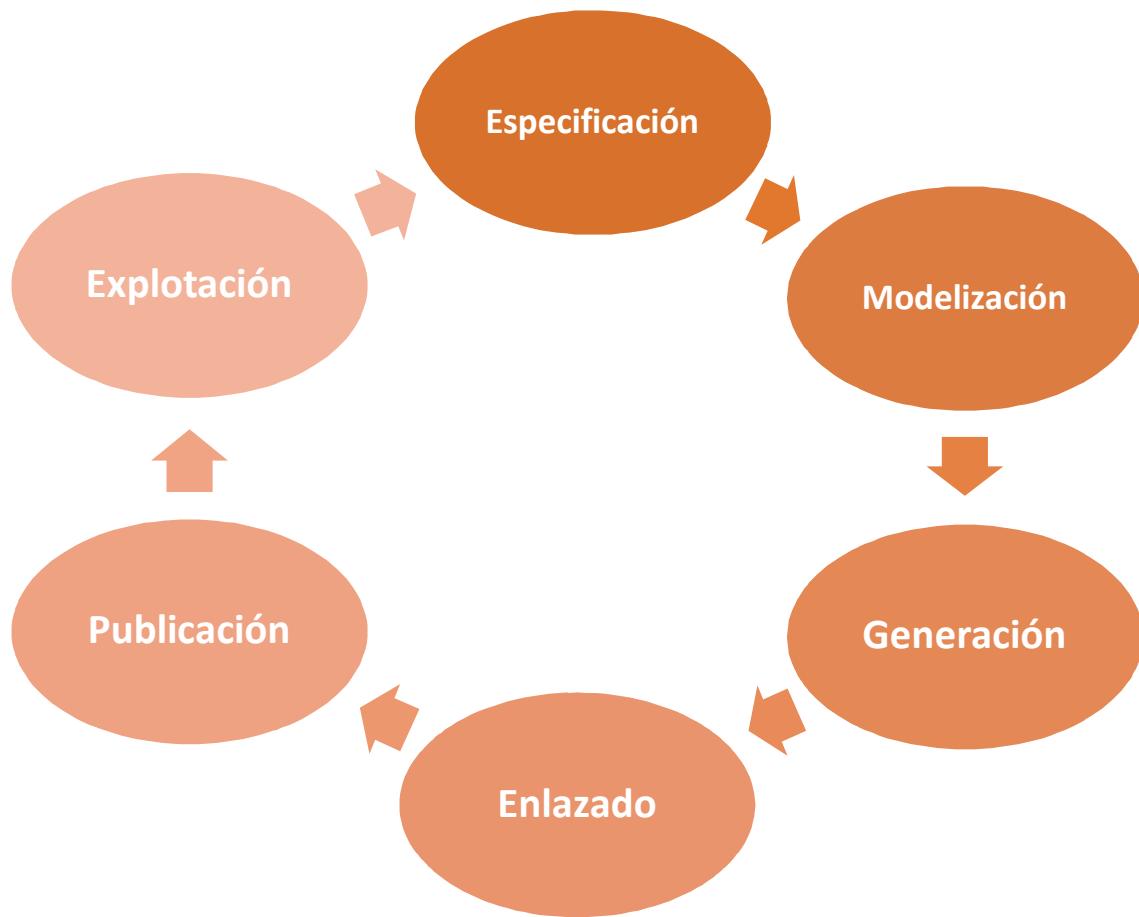


Ilustración 14 - Ciclo de vida de los datos enlazados

Una breve descripción de cada una de las fases es la siguiente:

- **Especificación:** Es una parte fundamental del proceso. Es necesario definir claramente qué tipo de información se tiene o se va a tener (qué conceptos son, qué propiedades tiene, con qué otros conceptos está relacionada, etc.).
- **Modelización:** Una vez acordados los detalles de la especificación se debe modelar una ontología que de soporte a la especificación proporcionada.
- **Generación:** Mediante las herramientas adecuadas, se utilizará la ontología generada junto con los datos abiertos básicos que se tienen para generar ficheros

RDF que contengan tripletas y puedan ser usados por motores de inferencia (SPARQL).

- **Enlazado:** Consiste en enlazar nuestros recursos con otros recursos existentes, siempre y cuando dichos recursos sean idénticos o muy similares a nuestros recursos. Sin duda esta fase es costosa ya que requiere una búsqueda enorme (el dominio es Internet) para encontrar aquellos recursos que puedan ser útiles. Además es otra de las fases clave en el ciclo de los datos enlazados, ya que es la fase que permitirá hacer uso de todas las ventajas de tener recursos conectados.
- **Publicación:** La publicación es la penúltima fase del ciclo, en la cual se crean metadatos para describir el conjunto de datos que se quiere publicar, se hace accesible a través de internet, exponiéndolo en repositorios de datos enlazados, y por supuesto se valida previamente.
- **Explotación:** Es el momento en el cual los datos ya son públicos y accesibles para todo el mundo.

Como se ha dicho antes, esto es un **ciclo**. Es decir, inmediatamente después de la publicación y posterior explotación de los datos, será necesario siempre volver a realizar una labor de estudio y especificación ya que la información y por ende los datos están en continuo cambio, y algo que hoy se modela de una forma podría modelarse mañana de otra muy distinta.

2.7. Proceso de generación de datos enlazados

A la hora de generar datos enlazados es necesario seguir varias metodologías, sin olvidar nunca que estamos trabajando con datos que pueden haber sido creados o proporcionados por otras personas, siendo imprescindible por tanto contar con las licencias y permisos adecuados para consumir, tratar y manipular dichos datos¹².

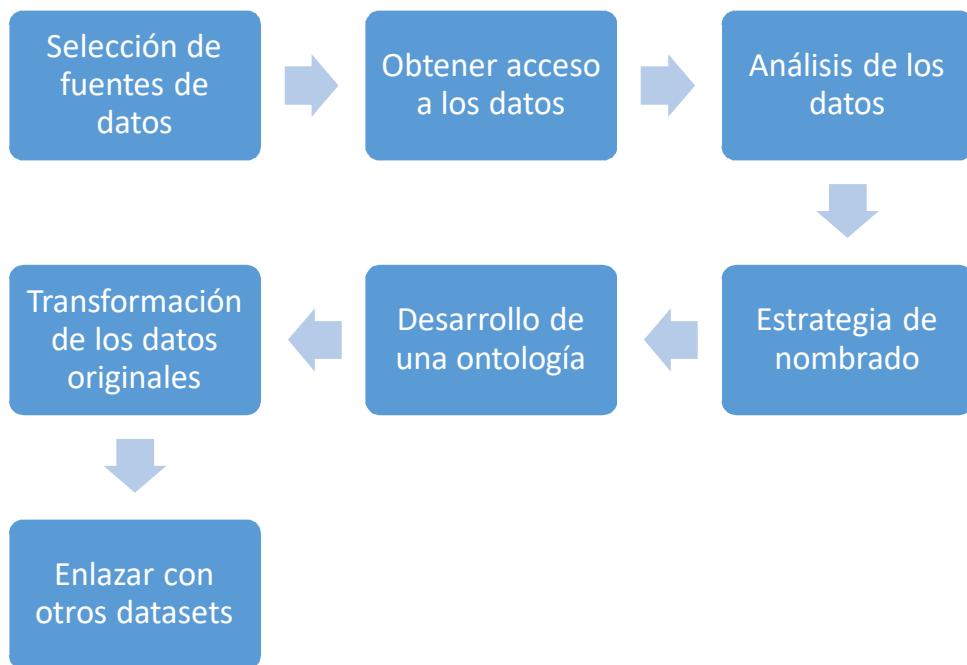


Ilustración 15 - Proceso de generación de datos enlazados

Una breve descripción de cada una de las fases es la siguiente:

- **Selección de fuentes de datos:** Consiste en elegir cuáles serán las fuentes de las que obtengamos la información. Es importante elegir correctamente las fuentes en función del tipo de información que queremos obtener, ya que no todas ofrecen los mismos datos ni la misma calidad en ellos.
- **Obtener acceso a los datos:** Consiste en analizar las licencias que tienen los datos que queremos utilizar. Si no disponen de una licencia que permita su uso, entonces **no** podremos utilizar dichos datos.
- **Análisis de los datos:** Será siempre necesario analizar qué datos hemos conseguido. No podremos utilizar conjuntos de datos en los que desconocemos el significado de algún campo, o algunos campos están incompletos (cuando se

requiera que estén completos) o sean erróneos. Estos datos se encontrarán probablemente en formatos como CSV, JSON, GeoJSON, KML o XML.

- **Estrategia de nombrado:** Nunca debemos olvidar que estamos en un entorno web. Todos los recursos se tienen que poder identificar de forma única y para ello es necesario definir una estrategia de nombrado de recursos.
- **Desarrollo de una ontología:** Para poder representar el conocimiento y habilitar la lectura automática a las máquinas, es necesario definir una ontología que aúne todas las características semánticas de los datos que se van a generar. Será por tanto necesario definir qué conceptos, propiedades y relaciones modelizarán nuestros datos.
- **Transformación de los datos originales:** Haciendo uso de la ontología desarrollada y de herramientas especializadas, se tienen que transformar los datos originales (CSV, XML...) en ficheros RDF útiles para los motores de inferencia.
- **Enlazar con otros datasets:** Una vez se tienen creados los ficheros RDF, hay que enlazar los recursos generados con otros ya existentes, con la intención de obtener toda la información posible de dichos recursos. Esto siempre se hará en la medida de lo posible, ya que no es posible enlazar cualquier recurso. Por ejemplo, será muy fácil enlazar un recurso llamado “Plaza Mayor” ya que se trata de una localización muy conocida. Sin embargo, es posible que una calle pequeña en un pueblo pequeño no haya sido incluida en ningún dataset y no sea por tanto posible enlazarla con nada. En estos casos, obviamente, no hay ninguna ganancia de información.

Todas estas fases deben llevarse a cabo de la forma más cuidada y meditada posible si se quieren obtener resultados adecuados.

2.8. Ontologías

Una ontología¹³ contempla los conceptos, propiedades y relaciones entre conceptos que existen en un dominio concreto. Una ontología, por tanto, debe ser capaz de agrupar todos los conceptos comunes a un ámbito concreto, y describirlos con la profundidad suficiente mediante el uso de propiedades, así como indicar las relaciones que existen entre los conceptos del dominio.

Las ontologías son herramientas útiles para organizar la información de la que se dispone y reducir la complejidad de los problemas. Son ampliamente utilizadas en inteligencia artificial y en la web semántica. Los elementos básicos que toda ontología debe contener son los siguientes:

- **Conceptos o clases:** Son conjuntos de cosas, conceptos, tipos de objetos... que sirven para representar algo del mundo real.
- **Propiedades:** Son atributos, rasgos o características que definen y describen a los conceptos o clases.
- **Relaciones:** Son las formas que tienen de relacionarse individuos de la misma o distintas clases.

Existen varios tipos de ontologías, como son las ontologías de dominio, generales, terminológicas o de modelado de conocimiento, entre otras. Se distinguen unas de otras por los conceptos que tratan de representar y los dominios en los que se aplican.

Las ontologías deben ser codificadas a la hora de su generación y para ello existen varios lenguajes. Uno de los más utilizados es OWL, que se trata de un lenguaje de declaraciones ontológicas, compatible con RDF, ya que todos los elementos que se definen en la ontología son modelados como recursos RDF, identificados por URIs.

Existen numerosas herramientas para desarrollar ontologías, pero en este trabajo se utilizará Protégé, un editor de código abierto para ontologías y *framework* para el desarrollo de sistemas inteligentes desarrollado por la Universidad de Stanford.

Capítulo 3

Análisis de sistemas relevantes

En esta sección se describirán sistemas que van a ser usados en el desarrollo de este proyecto. Por un lado se tratarán aquellos sistemas relativos a la obtención y modificación de los datos originales, así como los sistemas implicados en la generación de los datos enlazados. Por otro lado, se tratarán también aquellos sistemas o *Frameworks* utilizados en el desarrollo de la aplicación construida.

3.1. Fuentes de información

Como se ha indicado anteriormente, es fundamental elegir correctamente los orígenes de información, para poder evaluar sus licencias y también su calidad. En este contexto, se han elegido cuatro fuentes de información distintas que, combinadas, permiten obtener una gran cantidad de información.

3.1.1. Portal de datos abiertos de Madrid

El portal de datos abiertos de Madrid contiene una gran cantidad de datasets, separados en distintas categorías, destinados al uso público. Para cada dataset, se dispone de su sector, fecha de incorporación al catálogo, frecuencia de actualización y los formatos disponibles, así como las **licencias de uso** de cada dataset.

Si bien son varios los formatos que ofrecen (CSV, JSON, XML, RDF, KML, XLSX...) serán los ficheros en formato CSV los que se utilizarán en principio como base de conocimiento del sistema. La aplicación construida tomará cuatro conjuntos de datos de este portal como base de conocimiento: monumentos o edificios de carácter monumental, museos, bibliotecas y templos católicos. Este portal se encuentra en <http://datos.madrid.es/portal/site/egob>.

3.1.2. DBpedia y esDBpedia

DBpedia es un repositorio online abierto y gratuito. Contiene información estructurada obtenida directamente de Wikipedia mediante procesos de mapeo de los *infoboxes* que presenta cada página de Wikipedia. La enciclopedia libre está compuesta por documentos, algo que no ocurre en DBpedia, que se compone de datos estructurados. Esto permite que se puedan realizar consultas con alto nivel de complejidad contra DBpedia, algo que no sería posible realizar contra Wikipedia en sí misma. esDBpedia es igual que DBpedia pero contiene información específica de España.

Estos dos repositorios serán utilizados en este caso como fuentes que enriquecerán los conjuntos de datos obtenidos del portal de datos abiertos con información adicional que originalmente no estaba presente o era errónea o mejorable en dichos *datasets*. Se permite su uso bajo la licencia ***GNU General Public License***.

3.1.3. Open Street Map

Open Street Map (OSM) es un proyecto colaborativo que pretende crear mapas libres y editables¹⁴. Los mapas que utiliza se generan mediante fuentes libres como ortofotografías y toda la información que contiene se distribuye bajo la **licencia abierta ODbL** (Licencia abierta de bases de datos). En julio de 2017 tenía un tamaño de 800 Gigabytes aproximadamente.

Al ser un proyecto colaborativo, cualquier persona puede utilizar las APIs disponibles para editar los mapas añadiendo, modificando o eliminando elementos. Open Street Map utiliza una estructura de datos topológica, que tiene como elementos básicos los siguientes:

- **Nodos:** Son puntos que se establecen en una posición geográfica concreta.
- **Vías:** Son listas ordenadas de nodos que forman una línea (*polilínea* o *polyline*) o un polígono (*polygon*) en el caso de que la lista acabe y termine en el mismo punto.
- **Relaciones:** Son agrupaciones de vías, nodos u relaciones a las cuales se les pueden asignar propiedades comunes. Un claro ejemplo serían todas las vías (y por ende nodos) que forman parte del Camino de Santiago.

- **Etiquetas:** Son conjuntos de propiedades clave-valor que se pueden asignar a nodos, vías o relaciones. Por ejemplo, la etiqueta *name=Asador asado* indica que un elemento (presumiblemente un nodo) tiene el nombre *Asador asado*.

Open Street Map dispone de varias APIs para poder acceder a toda la información que hay en los mapas, ya sea para consultarla, modificarla o por supuesto añadir nueva información. En este contexto, en el presente trabajo **se va a utilizar la API Overpass**, una API que permite realizar operaciones de lectura sobre los mapas. Esta API ofrece un lenguaje específico para hacer consultas, aunque también admite que vengan en formato XML.

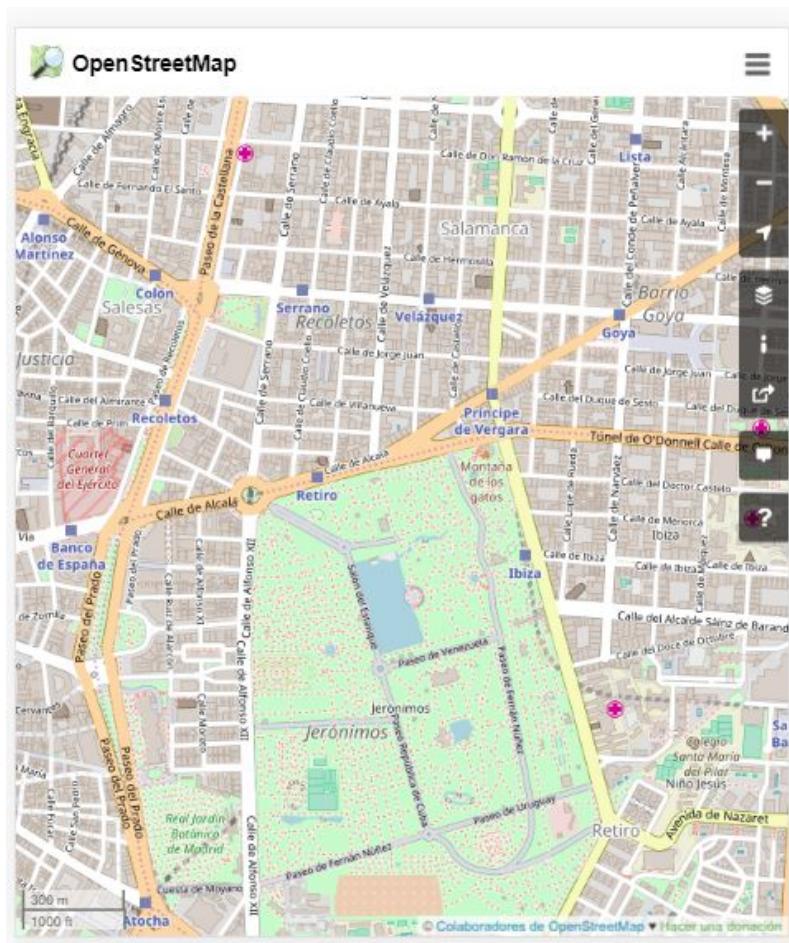


Ilustración 16 - Open Street Map

3.2. Modificación y generación de datos enlazados

Los datos obtenidos del portal de datos abiertos de Madrid serán modificados en función de las necesidades oportunas (campos erróneos o innecesarios) con el objetivo de limpiar los conjuntos de datos y dejarlos preparados para la posterior integración con la ontología que se desarrolle. Para ello, se utilizarán dos herramientas especializadas.

3.2.1. LOD Refine

LOD Refine es una herramienta para limpiar y adaptar datos, así como transformarlos y darles nuevos formatos. Cuenta con una extensión para trabajar con RDF gracias a la cual es posible definir una ontología con la que trabajar y generar ficheros RDF en diversos formatos (TTL, XML), así como enlazar los recursos que se generen con otros existentes.

Esta herramienta cuenta con un lenguaje específico para realizar operaciones de tratamiento de datos, GREL (*Google Refine Expression Language*). Este lenguaje ha sido utilizado para crear nuevas columnas en los conjuntos de datos que son uniones de otras existentes, o columnas con un valor basado en el valor que hubiese en otra columna del dataset.

row	value	value.replace(/.*\(/,'').replace(/\).*/,'')
1.	age 15 and over can read and write (2010 est.)[1]	2010 est.
2.	age 15 and over can read and write (2000 est.)[1]	2000 est.
3.	age 9 and over can read and write (2001 census)[1]	2001 census
4.	age 15 and over can read and write (2006)[3]	2006
5.	age 15 and over can read and write (1980 est.)[1]	1980 est.
6.	age 15 and over can read and write[1]	age 15 and over can read and write[1]
7.	age 15 and over can read and write (2010 est.)[1]	2010 est.

Ilustración 17 - GREL

LOD Refine se ha utilizado para tratar los cuatro conjuntos de datos seleccionados, adaptarlos a la ontología desarrollada y enlazar los recursos generados con otros ya existentes en DBpedia o esDBpedia.

Ilustración 18 - Enlazado de datos con LOD Refine

3.2.2. Protégé

Protégé es un editor de código abierto para ontologías en formato OWL y framework para el desarrollo de sistemas inteligentes desarrollado por la Universidad de Stanford. Esta herramienta se puede descargar desde su propia página web o se puede usar de forma alternativa una versión web sin necesidad de descargar e instalar nada. Protégé es una herramienta que permitirá definir los nombres de las URIs que identificarán los recursos así como todos los elementos (clases, propiedades, relaciones) indicados anteriormente de manera sencilla.

3.3. Desarrollo y despliegue

3.3.1. Spring Framework

La aplicación hace uso de *Spring*, un *framework* de código abierto para el desarrollo de aplicaciones en Java. Gracias a este *framework* se facilita bastante el desarrollo de microservicios ya que:

- **Provee al desarrollador de servidores de aplicaciones embebidos**, como *Tomcat* o *Jetty*.
- **Utiliza archivos de dependencias POM** con las dependencias externas de cada uno de los servicios.
- Facilita la **integración con Maven**.
- Hace **uso de anotaciones** que permiten realizar labores de configuración o inyección de dependencias.

La razón de utilizar *Spring* y no haber utilizado Java EE descansa en la posibilidad de usar un servidor web convencional para el despliegue del sistema, siguiendo las APIs que se establecen en las librerías de *Spring*. Usando Java EE el desarrollo estaría mucho más condicionado por el servidor de aplicaciones utilizado. Al utilizar *Spring*, únicamente se depende de las APIs.

Además, *Spring* proporciona numerosos módulos especializados para cada una de las tareas más comunes del desarrollo de software. Tiene un módulo de acceso a datos, *Spring Data*, un módulo de seguridad, *Spring Security*, un módulo para trabajar con las vistas del modelo MVC, *Thymeleaf*, así como otros módulos como *Spring Boot*, que permite crear de forma sencilla aplicaciones y ponerlas en producción.



Ilustración 19 - Logo de Spring

3.3.2. Amazon Web Services

Amazon Web Services¹⁵ (AWS) es un conjunto de servicios *cloud* que juntos conforman una plataforma de *cloud computing*. Esta plataforma es utilizada por grandes aplicaciones como Dropbox o FourSquare y competidora de Microsoft Azure y Google Cloud Platform. Si bien la lista de servicios que ofrece AWS es inmensa, sólo se explicarán aquellos servicios relacionados íntimamente con el desarrollo y despliegue de la aplicación *Madrid on You*.

- **Amazon EC2:** *Elastic Cloud Computing*. Es el servicio que permite crear instancias virtuales de máquinas junto con balanceadores de carga. Es el servicio más usado de AWS. Este servicio se ha utilizado para desplegar los microservicios que componen la aplicación desarrollada. Dicho despliegue hace que el sistema sea público, y pueda ser accedido desde cualquier dispositivo. Por motivos de seguridad (ataques, abusos en el servicio), la dirección pública para acceder al sistema desplegado no se muestra en esta memoria. Si se quiere obtener, se deberá solicitar mediante un correo electrónico a la dirección jesusperezmelero@gmail.com.
- **Amazon DynamoDB:** Es uno de los servicios que se ofrecen en AWS para utilizar bases de datos distribuidas. En el caso de Dynamo DB **se trata de una base de datos NoSQL**, aunque existe la posibilidad de usar Amazon RDS para bases de datos SQL, pudiendo elegir entre MySQL, PosgreSQL, Oracle y SQL Server.

AWS sigue el modelo de pago *Pay as you go* que quiere decir que todos los meses habrá que pagar tanto dinero como servicios y capacidad de cómputo se haya consumido. En el caso concreto de este trabajo, se ha utilizado una suscripción gratuita de un año a AWS para evitar el pago de tasas.



Ilustración 20 - Logo de AWS

3.4. Otros

3.4.1. Apache JENA

Apache Jena¹⁶ es un *framework* que se utiliza para construir aplicaciones java basadas en ontologías. Gracias a Jena se dispone de APIs para leer, procesar y escribir ontologías RDF y OWL, así como un motor de inferencia compatible con SPARQL para realizar consultas a ficheros RDF. Además cuenta con **Fuseki**, un servidor SPARQL que se puede desplegar en local y permite ejecutar queries contra conjuntos de datos previamente definidos con una interfaz de usuario muy amigable, algo que sin duda agiliza enormemente el desarrollo.

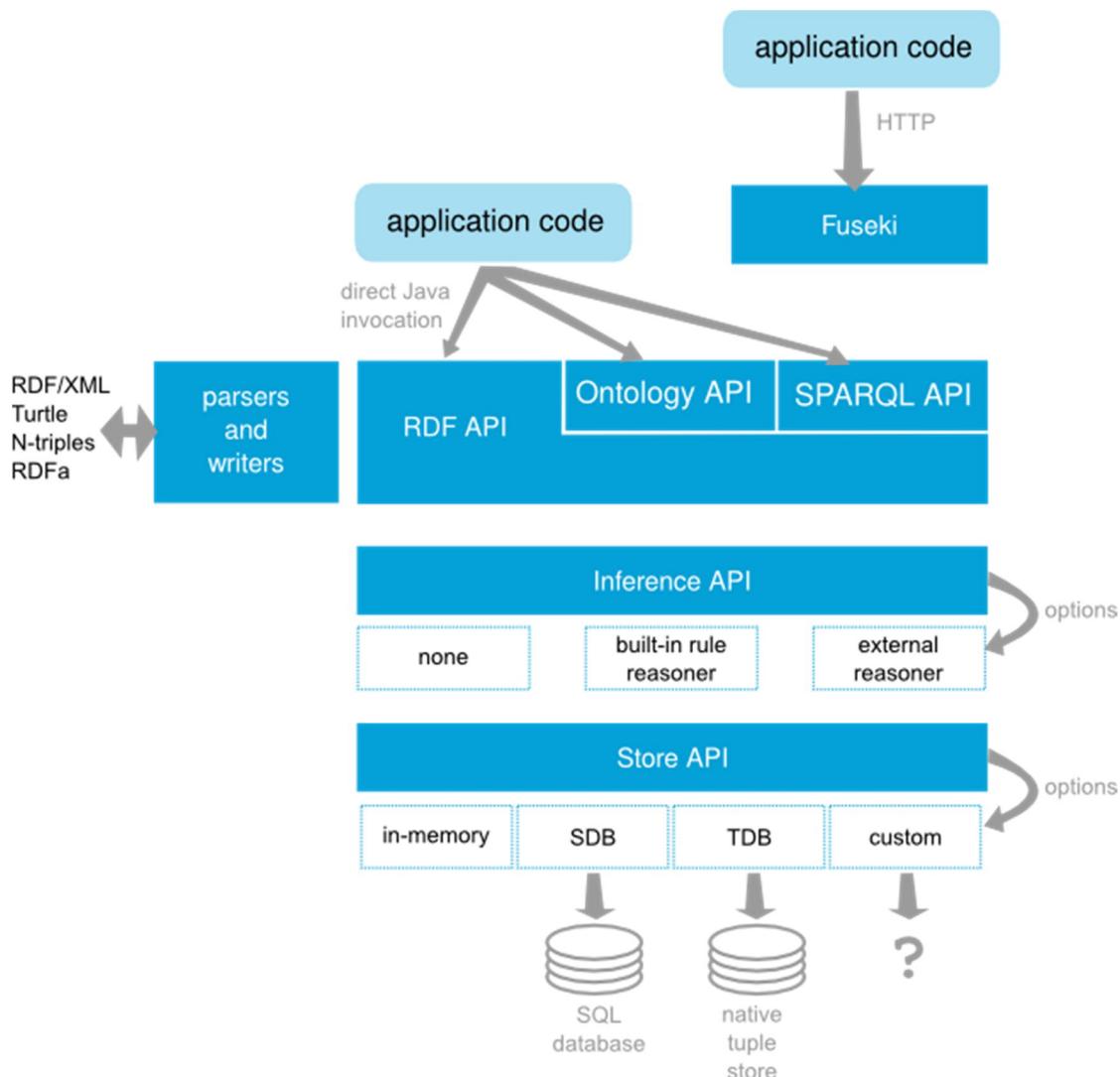


Ilustración 21 - Arquitectura de JENA

Capítulo 4

Especificación de requisitos

Este capítulo contiene la ERS (Especificación de Requisitos Software) para la aplicación *Madrid on You*.

4.1. Propósito

En el proceso de diseño de la aplicación, se han identificado diferentes *Features* o características que la aplicación deberá tener. Para cada característica o *feature* se detallan uno o más requisitos software con su correspondiente numeración. Por tanto, cada requisito podrá identificarse mediante el patrón *Feature (número)*. Por tanto, el requisito 2(1) haría referencia al punto 1 de la *feature* 2.

4.2. Ámbito

El presente proyecto consiste en la creación de una aplicación web dirigida a la generación de rutas aleatorias por la ciudad de Madrid utilizando datos abiertos y enlazando dichos datos con otras fuentes activas de información para obtener una gran cantidad de información sin hacer uso de grandes bases de datos tradicionales.

Esta aplicación permitirá, por tanto, generar rutas aleatorias por uno o varios distritos de Madrid de forma que se tenga un plan para pasar el día. Adicionalmente, la aplicación soportará gestión de usuarios, esto es, inicios de sesión, registros y actualizaciones del perfil de los usuarios.

4.3. Perspectiva del producto

Esta aplicación permite generar rutas turísticas por la ciudad de Madrid, por lo que está pensada especialmente para el uso de turistas o personas que quieran visitar Madrid y no tengan tiempo para planificar una ruta ellos mismos, debido a la falta de conocimiento de la ciudad.

En ocasiones ciertas *apps* pueden sugerir sitios de interés, pero *Madrid on You* no se queda ahí. Además de sugerirte lugares fabulosos, también sugerirá restaurantes, teatros, cines o pubs que estén cerca de los sitios que se podría visitar, de forma que en muy pocos segundos se ha creado un plan completo de visita por una parte de la ciudad.

4.4. Requisitos específicos

A continuación se presenta la lista de las prestaciones (*Features*) que la aplicación deberá satisfacer. Para cada característica o *feature* se detallan uno o más requisitos software, adecuadamente nombrados. El formato del nombre se especifica en cada apartado.

4.5.1. Feature #1 – Inicio de sesión

La aplicación dispondrá de un mecanismo de *login* para poder autenticar al usuario en el sistema. Esta funcionalidad debe estar siempre disponible para cualquier usuario que no esté autenticado previamente.

Los requerimientos de este tipo se codifican con la palabra INI y un número secuencial de tres dígitos.

- **Estado:** Aprobado
 - **Prioridad:** Crítico
 - **Esfuerzo:** Medio
 - **Versión:** 1.0
-
- **INI-001:** El mecanismo de *login* debe fundamentarse al menos en un nombre de usuario y una contraseña.
 - **INI-00:** Si el *login* es correcto, se redirigirá al usuario a la página principal de la aplicación.
 - **INI-003:** Si el *login* es incorrecto, se mostrará un mensaje de error (en texto rojo) indicando que alguno de los parámetros era incorrecto (no se debe especificar cuál).
 - **INI-004:** Las contraseñas deben almacenarse en una base de datos habiendo sido tratadas previamente por una función hash.
 - **INI-005:** Si un usuario ya está autenticado en el sistema, será redirigido a la página principal si intenta volver a realizar un *login*. Esto será así hasta que se realice una operación de *logout*, en cuyo caso dejará de estar autenticado en el sistema.

4.5.2. Feature #2 – Registro en la aplicación

Dado que la aplicación contempla un *login*, será necesario disponer de un formulario de registro de nuevos usuarios que puedan utilizar la aplicación. Esta funcionalidad, al igual que el *login*, deberá estar disponible para todos los usuarios no autenticados en el sistema.

Los requerimientos de este tipo se codifican con la palabra REG y un número secuencial de tres dígitos.

- **Estado:** Aprobado
- **Prioridad:** Crítico
- **Esfuerzo:** Medio
- **Versión:** 1.0

- **REG-001:** El formulario de registro debe contener los campos mínimos para recopilar información útil del usuario, es decir, nombre de usuario, contraseña y e-mail.
- **REG-002:** La contraseña deberá tener al menos 3 caracteres.
- **REG-003:** El nombre de usuario elegido no puede ser idéntico a otro ya existente.
- **REG-004:** Ningún campo del registro puede quedar vacío
- **REG-005:** Si el registro es correcto, se redirigirá al usuario a la página principal de la aplicación.
- **REG-006:** Si el registro es incorrecto, se mostrará un mensaje de error en texto rojo indicando que los parámetros eran incorrectos (sin especificar cuál).

4.5.3. Feature #3 – Actualización del perfil

La aplicación debe permitir que el usuario actualice su información personal, es decir, contraseña y e-mail. El nombre de usuario no se puede modificar.

Los requerimientos de este tipo se codifican con la palabra UPD y un número secuencial de 3 dígitos

- **Estado:** Aprobado
- **Prioridad:** Crítico
- **Esfuerzo:** Medio
- **Versión:** 1.0

- **UPD-001:** Si los datos indicados son correctos, el usuario verá sus datos actualizados.
- **UPD-002:** Si los datos indicados son erróneos, el usuario verá un mensaje de error y los datos no se modificarán, prevaleciendo así los antiguos.

4.5.4. Feature #3 – Atributos

El sistema debe ser intuitivo y usable, de forma que los usuarios puedan aprender a usarlo de forma rápida. Además, deberá cumplir ciertos atributos de fiabilidad, mantenimiento y seguridad.

Los requerimientos de este tipo se codifican con la palabra ATR-UX para los requerimientos relacionados con la usabilidad, ATR-FIA para los requerimientos relacionados con la fiabilidad, ATR-MNG para los requerimientos relacionados con la mantenibilidad y ATR-SEC para los requerimientos relacionados con la seguridad. Todos ellos acompañados de un número secuencial de tres dígitos.

- **Estado:** Aprobado
 - **Prioridad:** Crítico
 - **Esfuerzo:** Medio
 - **Versión:** 1.0
-
- **ATR-UX-001:** Ninguna operación deberá superar las 10 acciones elementales.
 - **ATR-UX-002:** Los mensajes de error deberán ser suficientemente explicativos.
 - **ATR-UX-003:** La aplicación deberá contener un apartado de ayuda.
-
- **ATR-FIA-001:** La aplicación funcionará al menos en los casos de uso que se detallen.
-
- **ATR-MAN-001:** La aplicación deberá estar documentado internamente.
 - **ATR-MAN-002:** La aplicación debe presentar una alta cohesión.
-
- **ATR-MAN-003:** La aplicación debe presentar un bajo acoplamiento.
 - **ATR-SEC-001:** La aplicación será accesible por usuarios registrados.
 - **ATR-SEC-002:** La aplicación no será accesible por usuarios no registrados.

4.5.5. Feature #3 – Rendimiento

Los requerimientos de este tipo se codifican con la palabra PER y un número secuencial de 3 dígitos.

- **Estado:** Aprobado
 - **Prioridad:** Crítico
 - **Esfuerzo:** Alto
 - **Versión:** 1.0
- **PER-001:** La aplicación no sufrirá bloqueos o ralentizaciones, y no debe operar en ningún caso con un tiempo de espera mayor a 20 segundos **en promedio**.

4.5.6. Feature #4 – Generación de rutas

El usuario podrá solicitar un recorrido aleatorio por distintos edificios de carácter monumental o turístico de Madrid. Esta funcionalidad es exclusiva para los usuarios que estén autenticados en el sistema.

Los requerimientos de este tipo se codifican como GEN y un número secuencial de 3 dígitos.

- **Estado:** Aprobado
 - **Prioridad:** Crítico
 - **Esfuerzo:** Alto
 - **Versión:** 1.0
-
- **GEN-001:** El usuario deberá poder elegir si quiere que el recorrido sea por todo Madrid o únicamente por un distrito.
 - **GEN-002:** El usuario deberá poder elegir las horas entre las que se va a realizar el recorrido.
 - **GEN-003:** El usuario deberá poder elegir el número de elementos que contendrá el recorrido.
 - **GEN-004:** La ruta resultante se formará sobre un mapa con cada elemento perteneciente señalado en él.

4.5.7. Feature 5# - Visualización de rutas

El usuario dispondrá de varias opciones de visualización para la ruta que se ha generado. Esta característica es exclusiva para los usuarios que estén autenticados en el sistema.

Los requerimientos de este tipo se codifican como VIS y un número secuencial de 3 dígitos.

- **Estado:** Aprobado
- **Prioridad:** Crítico
- **Esfuerzo:** Alto
- **Versión:** 1.0

- **VIS-001:** Junto con cada edificio perteneciente a la ruta generada, se deberá mostrar una breve descripción del mismo, acompañada de datos útiles para el usuario.
- **VIS-002:** Junto con cada edificio se mostrará, en caso de ser posible, una imagen que se ajuste a la actualidad del edificio.
- **VIS-003:** Junto a cada elemento de la ruta generada, aparecerán diversas localizaciones de puntos de interés como pueden ser cafeterías, restaurantes, teatros, cines o información sobre parkings o paradas de bus y taxi.
- **VIS-004:** Los puntos de interés adicionales deberán asociarse a cada uno de los elementos de la ruta así como visualizarse en el mapa.
- **VIS-005:** Por cada punto de interés adicional se deberá indicar algún tipo de información útil, junto con la distancia aérea a la que se encuentra el elemento del monumento en cuestión.
- **VIS-006:** Todos los elementos que se muestren en el mapa deben poder mostrarse y ocultarse.
- **VIS-007:** El mapa resultante debe tener al menos dos tipos de vistas distintas.
- **VIS-008:** Se deberá poder hacer zoom sobre el mapa.
- **VIS-009:** Se deberá poder enviar por correo electrónico el detalle de la ruta generada.

Capítulo 5

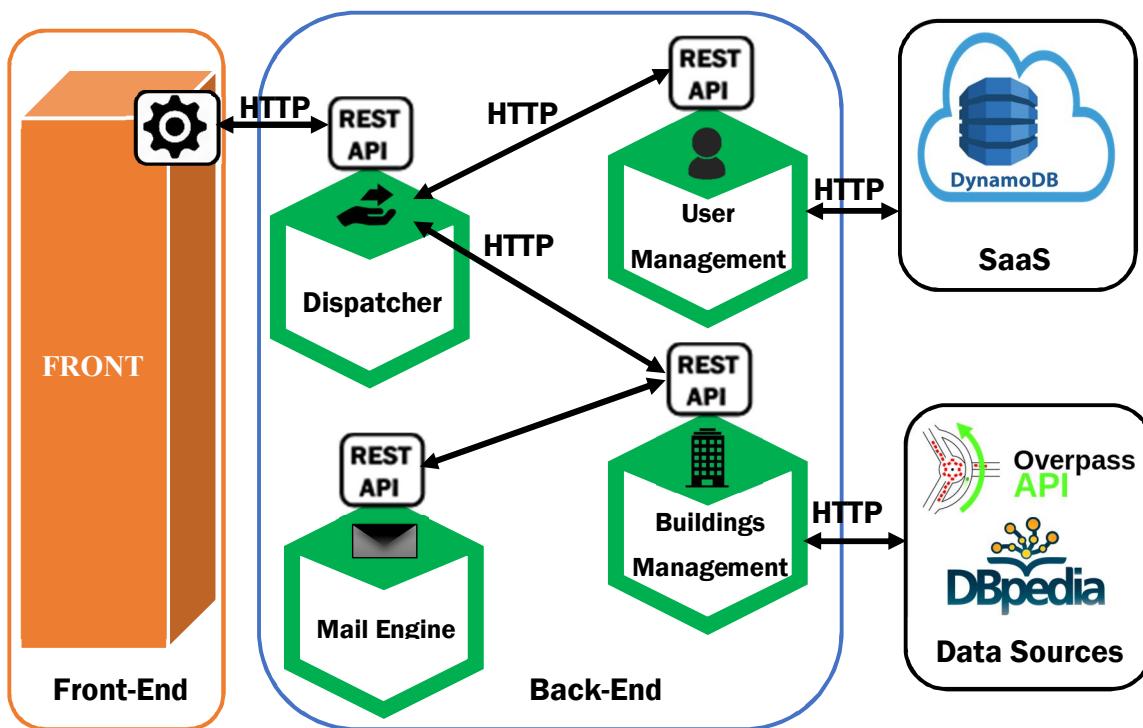
Diseño e implementación

5.1. Introducción

Una vez especificados los requisitos que deben definir la aplicación, se definirá cual es el modelo de arquitectura software elegido para desarrollarla. Se hará presentando en primer lugar la arquitectura elegida y después profundizando en cada uno de los módulos que la componen.

5.2. Arquitectura del sistema

El sistema construido presenta una gran variedad de componentes, que se pueden observar en la siguiente ilustración. En primer lugar, el sistema cuenta con un front-end que seguirá una arquitectura clásica MVC¹⁷ haciendo uso de *Thymeleaf*¹⁸. En cuanto al back-end, seguirá una arquitectura basada en microservicios o MSA¹⁹. Por último, existe una capa SaaS en la que se encuentra Dynamo DB y una capa de acceso a datos externos.



Como se puede observar en la ilustración anterior, la arquitectura se compone por una parte fron-end y una parte back-end, además de los servicios SaaS (Amazon AWS) y fuentes de datos externas al sistema. El back-end está construido siguiendo una filosofía de microservicios. El papel de cada uno de los elementos que componen la arquitectura es el siguiente:

- **Front:** Es el componente que contiene todo el contenido gráfico de la aplicación. Se limita a servir al usuario varias vistas con la información necesaria para que pueda interactuar con la aplicación. Cuando tiene que comunicarse con la parte back, lo hace a través de su controlador, que se comunica con el *dispatcher* únicamente. Es decir, este componente no tiene conocimiento de dónde están desplegados (dirección + puerto) el resto de microservicios excepto el *dispatcher*.
- **Dispatcher:** Este microservicio actúa como puente entre el front y los distintos microservicios pertenecientes al back-end. Cuando recibe una petición http del front, analiza el tipo de esa petición y en función del mismo redirige la petición al microservicio correspondiente para que pueda ser procesada. Cuando el microservicio al que se le redirigió la petición termina su trabajo, retorna una respuesta al dispatcher indicando el resultado de la operación realizada. Dicha respuesta es enviada al front, donde se tratará de la forma correspondiente.
- **User Management:** Este microservicio se encarga de realizar todos los registros y autenticaciones en la aplicación. Para ello, accede a la base de datos NoSQL DynamoDB que almacena todos los datos de los usuarios registrados en el sistema. Este microservicio recibirá peticiones que podrán ser de *login*, de registro o de actualización. Las procesará y emitirá una respuesta al *dispatcher* que a su vez será reenviada al front.
- **Buildings Management:** Este microservicio contiene la lógica principal de la aplicación, ya que es el encargado de generar las rutas aleatorias que son mostradas al usuario en pantalla. Consta de varias partes. En primer lugar, contiene un módulo para tratar ficheros RDF y realizar consultas SPARQL sobre dichos ficheros, pudiendo acceder a DBpedia en el caso de que sea necesario extraer información de allí. En segundo lugar, contiene un módulo para realizar consultas a Open Street Map a través del API Overpass, para obtener puntos de

interés cercanos a las localizaciones que compondrán la ruta. Una vez se ha procesado toda la información y se ha generado la ruta, se devuelve la composición de dicha ruta al *dispatcher*, que a su vez la reenvía al front, donde será mostrada al usuario.

- **Mail Engine:** Este microservicio se encarga de enviar correos electrónicos. Su funcionalidad principal es enviar a los estudiantes que lo deseen un reporte de cómo ha sido generada la ruta, de forma que les ayude en su aprendizaje.
- **SaaS:** Se trata de los recursos de AWS, en este caso de Dynamo DB. Las tablas de Dynamo guardarán información sobre los usuarios registrados en el sistema
- **Data Sources:** Son los servicios externos de los que se recupera información. En este caso, DBpedia, esDBpedia y Open Street Map mediante la API Overpass.

5.3. Front – End

El front-end de la aplicación es la parte encargada de mostrar distintas vistas al usuario para que este pueda relacionarse con la aplicación y así aprovechar sus funcionalidades. Este componente utiliza el patrón de **arquitectura MVC**. Para desarrollar el front-end, se han utilizado las siguientes tecnologías:

- **Thymeleaf:** Thymeleaf es una librería para Java que implementa un motor de plantillas de XML, XHTML y HTML5, idónea para la capa vista del MVC. El objetivo principal de esta librería es permitir la creación de plantillas de una manera elegante y un código bien formateado. Con esta librería se han desarrollado las 5 plantillas que componen el esqueleto del front-end.
- **HTML:** Junto con Thymeleaf, se ha utilizado para componer el esqueleto del front-end.
- **CSS:** Se han utilizado varias hojas de estilo para conseguir combinaciones de colores y estilos que se mantienen en todas las vistas de la aplicación.
- **JavaScript:** Se han desarrollado varios scripts que dotan de cierto comportamiento al front-end.

Además, con el objetivo de que esta aplicación tuviera características de una RIA (*Rich Internet Application*) se ha hecho uso de la metodología AJAX en ciertas partes de la aplicación. De esta forma, en algunas partes de la aplicación no es necesario recargar la página para mostrar una respuesta del servidor, lo que mejora la interactividad, velocidad y usabilidad de la aplicación.

El front-end contiene también un controlador que se encarga de recoger las peticiones de los usuarios y devolver las vistas que *Thymeleaf* selecciona al usuario, así como de lanzar las peticiones HTTP oportunas al back-end (sólo al *dispatcher*) cuando es necesario.

5.3.1. Vistas

De acuerdo con Thymeleaf, la aplicación consta de 5 plantillas o vistas que permiten utilizar la totalidad de la aplicación y una vista adicional para casos de error. Las vistas son las siguientes (nótese que las siguientes ilustraciones se encuentran volteadas para una mejor visualización):

- **Index.html:** Esta plantilla se corresponde con el punto de entrada principal de la aplicación. Muestra una breve descripción del objetivo de la misma, así como varias fotografías de la capital organizadas en un *slider*. En la parte superior derecha, está el botón que permite iniciar sesión o registrarse en la aplicación. Una vez presionado el botón, aparecerá un diálogo modal.

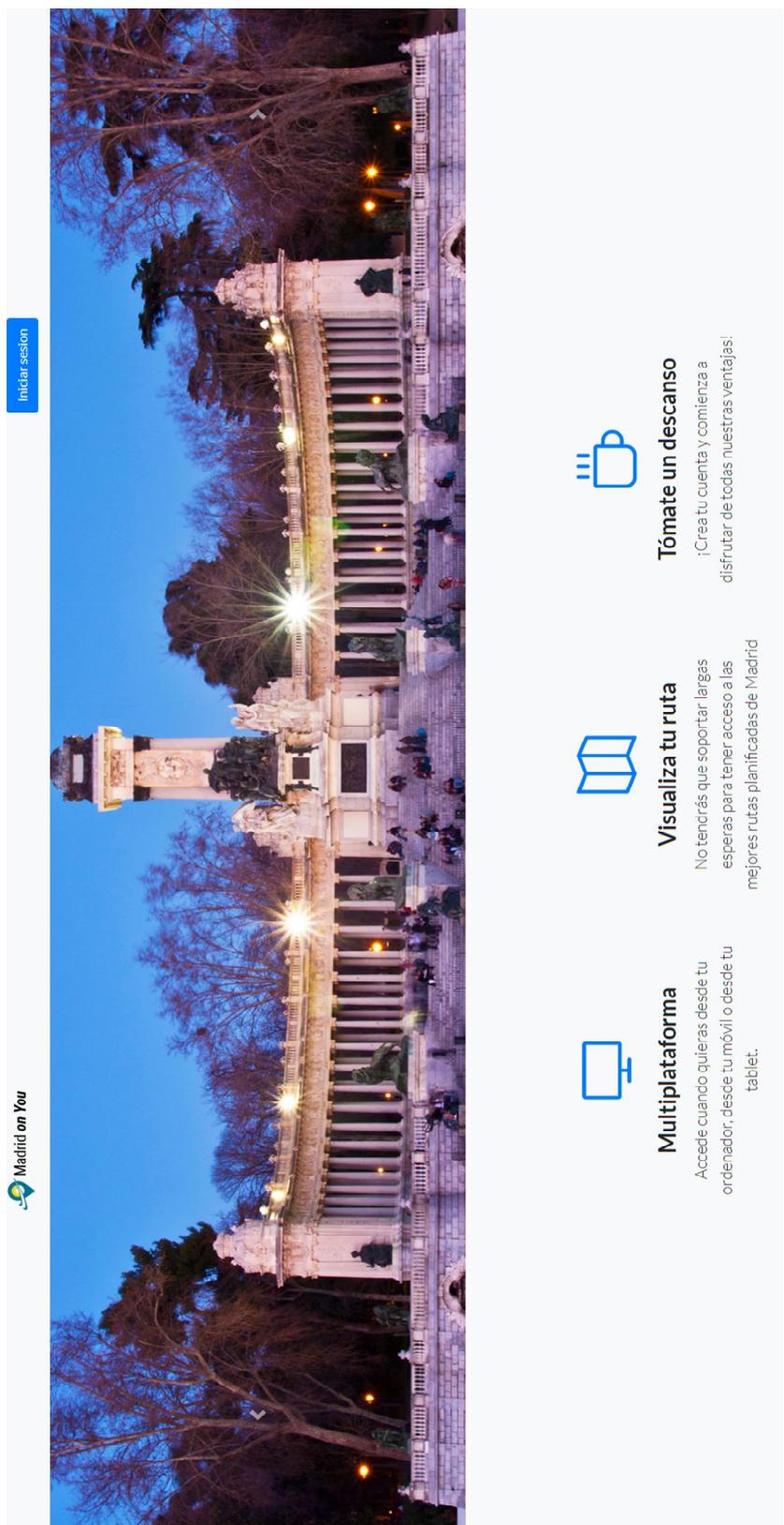


Ilustración 23 - Vista index.html (PC)

- **Home.html:** Esta vista aparecerá una vez el usuario se haya registrado o haya iniciado sesión en la aplicación de forma satisfactoria. Es el panel principal del usuario, donde se muestra actividad reciente relativa a la aplicación. Todo tipo de anuncios o mensajes podrán verse en esta sección a modo de blog.

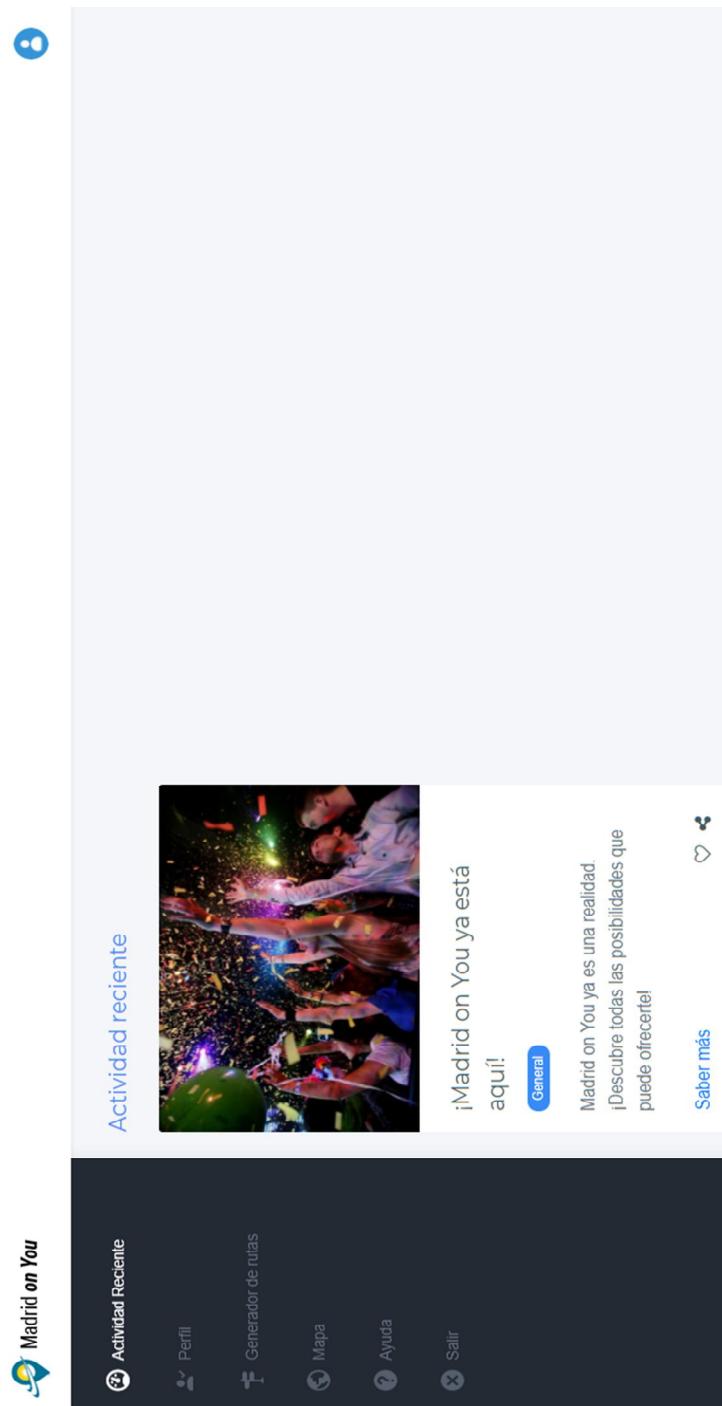


Ilustración 24 - Vista home.html (PC)

- **Profile.html:** Esta vista se corresponde con el perfil del usuario que está registrado, donde verá su foto e información de usuario (nombre de usuario, correo electrónico y contraseña). La longitud de contraseña que puede apreciarse en la siguiente ilustración no tiene por qué corresponderse con la longitud de la contraseña auténtica. Si el usuario quisiera modificar alguno de sus datos básicos, podrá hacerlo escribiendo el nuevo valor deseado en el campo correspondiente y haciendo click en el botón destinado a tal efecto.

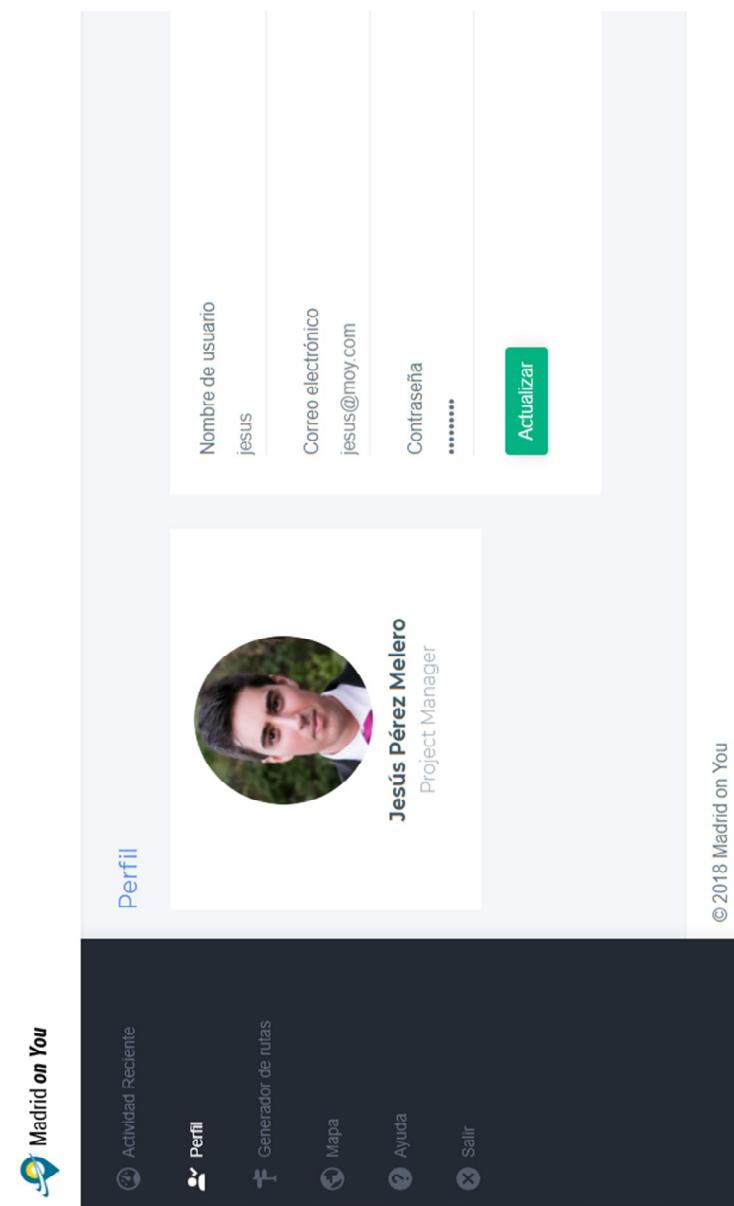


Ilustración 25 - Vista profile.html (PC)

- **Tourist.html:** Esta vista está destinada a la elección de las preferencias de la ruta que va a ser generada para el usuario.

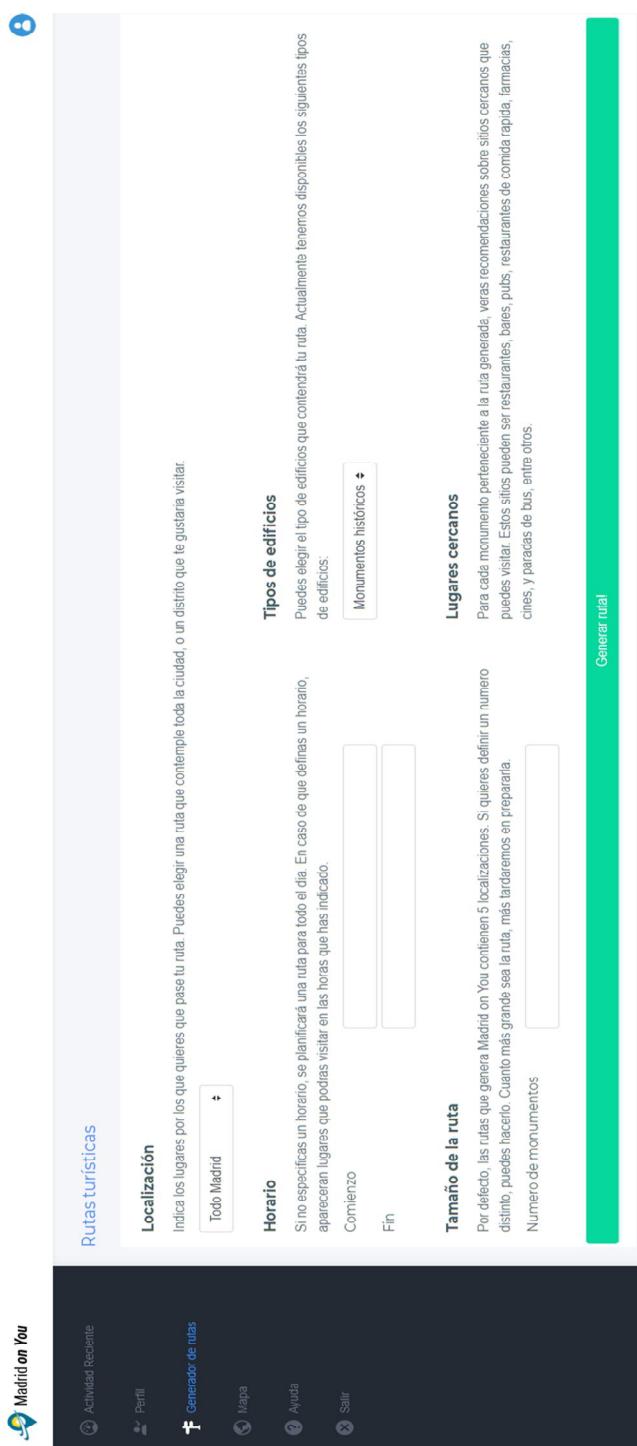


Ilustración 26 - Vista tourist.html (PC)

- **Map-osm.html:** Esta vista es mostrada al usuario en dos ocasiones. Si el usuario hace click en la opción “Mapa” accederá a esta vista, en la cual se mostrará un mapa vacío, que podrá utilizar para las consultas que desee. Esta vista se mostrará también (y de hecho es su principal motivación) cuando el usuario haya definido los parámetros de la ruta que quiere generar. Esta vista mostrará un mapa con los elementos de la ruta que se ha generado así como con los puntos de interés asociados a cada elemento de la misma.

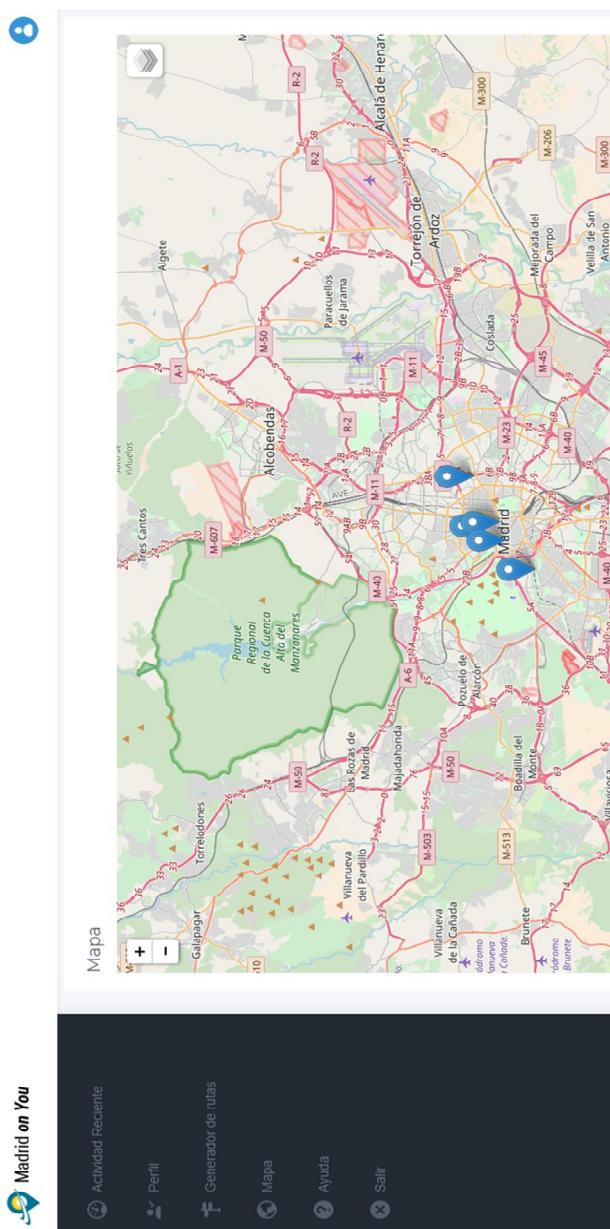


Ilustración 27 - Vista Map-osm.html - Elementos de la ruta (PC)

The screenshot shows a web page with a header featuring a blue circular icon with a white person symbol and the text "Madrid on You". Below the header, there is a title "Ermita de San Isidro" and a large image of a white, domed church building with a cross on top. To the left of the image, there is a sidebar with several icons and their corresponding labels: "Actividad Reciente", "Perfil", "Generador de rutas", "Mapa", "Ayuda", and "Salir". The main content area contains the following information:

Dirección: paseo quince de mayo 62
Distrito: CARABANCHEL
Teléfono: 913 650 841

Descripción: No hay información

Información adicional: Ermita: sábados de 11 a 13 horas (celebración de la Eucaristía a las 12 horas). Fuente del Santo: solo permanece abierta desde la semana anterior a la fiesta de San Isidro hasta la semana posterior. Oficio en rito hispano mozárabe: los días 2 de enero a las 18 horas (comienzo del año Caput anni) y 11 de mayo a las 18,30 horas (aniversario de la dedicación).

[Página web](#)

On the right side of the page, there are two columns of red rectangular buttons, each with a white border:

Column 1 (Top): Cines, Teatros, Paradas de Taxi, Paradas de bus

Column 1 (Bottom): Pubs, Cafeterías, Bares, Aparta-bicis, Parkings

Column 2 (Top): Comida rápida, Restaurantes, Farmacias, Cajeros

Column 2 (Bottom): Pubs, Cafeterías, Bares, Aparta-bicis, Parkings

Ilustración 28 - Vista Map-osm.html - Elemento de la ruta (PC)

Como puede verse en la ilustración, aparecen por defecto los elementos de la ruta señalados en el mapa, el cual tiene dos controles situados en las esquinas superiores izquierda y derecha que permiten modificar el zoom y los elementos que se muestran en el mapa, respectivamente. Para cada elemento, se muestra una lista de posibles puntos de interés que pueden interesar al usuario. Haciendo click en uno de ellos se mostrará la información de dichos puntos en caso de que exista.



Ilustración 29 - Vista Map-osm.html - Farmacias cercanas al elemento de la ruta (PC)

En la ilustración podemos ver las farmacias cercanas al elemento de la ruta seleccionado. Para cada punto de interés (en este caso farmacias) se muestra la distancia aérea desde el elemento de la ruta hasta el punto de interés, así como información adicional relativa al acceso, horarios, etc.

El usuario puede elegir si mostrar o no los puntos de interés asociados a los elementos de la ruta en el mapa. Por defecto no se muestran para evitar una sobrecarga de elementos en el mapa, algo que sería desagradable para el usuario. Si decide mostrar alguno de los tipos de puntos de interés a través del control derecho del mapa, los verá dibujados en el mismo. Un ejemplo puede verse en la siguiente ilustración (teatros cercanos).

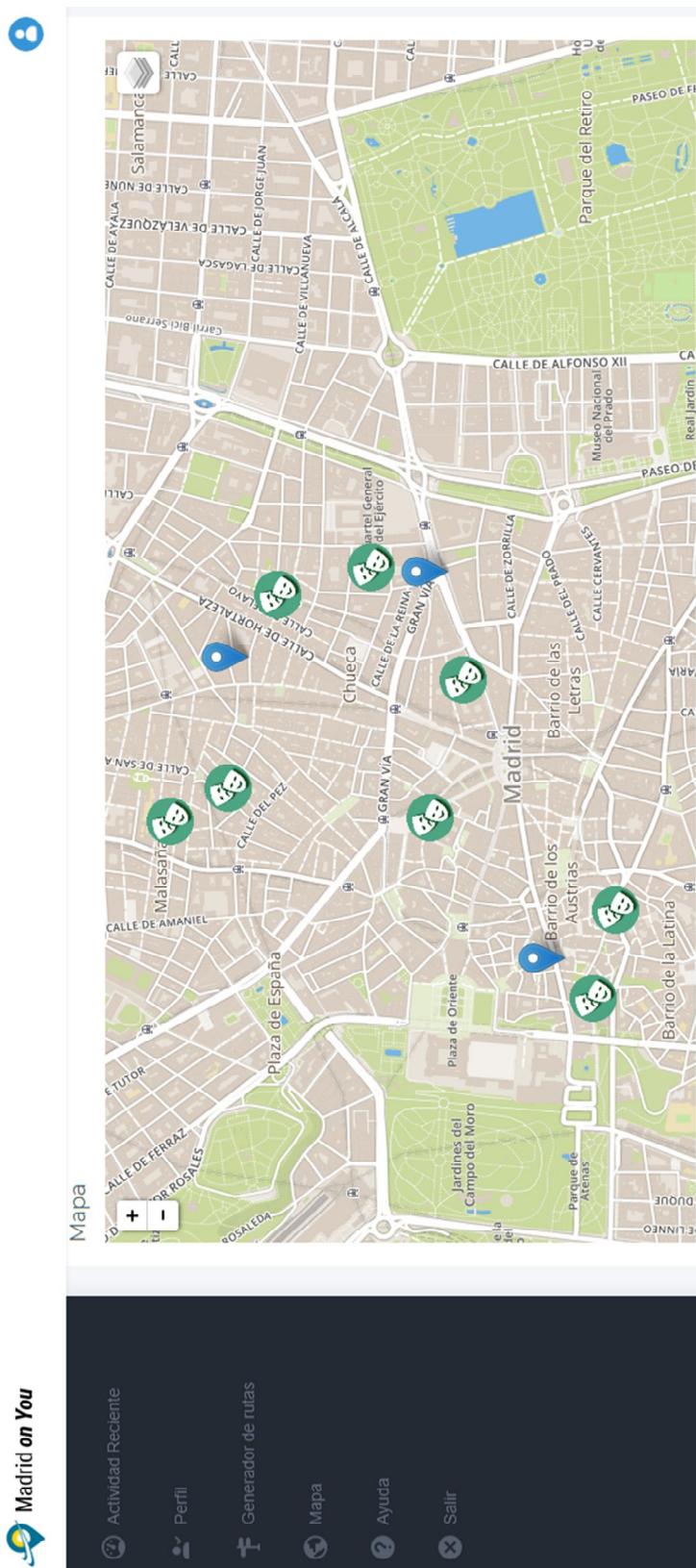


Ilustración 30 - Vista Map-osm.html - Teatros cercanos a varios elementos (PC)

- **Help:** Cuando el usuario haga click en el botón de ayuda aparecerá un PDF en otra pestaña del navegador, en el que podrá ver ayuda con respecto a la utilización del sistema.

5.3.2. JavaScript

Como se ha indicado anteriormente, se ha utilizado código JavaScript para dar comportamiento a ciertas partes del front. Además, se ha utilizado la metodología AJAX²⁰ en algunas de las peticiones que se realizan al back-end. Si bien se ha utilizado código JavaScript (incluyendo funciones de la librería JQuery) para numerosos efectos y animaciones (**código no desarrollado por mí**), únicamente se detallarán aquellos scripts cuyo funcionamiento es vital para el correcto funcionamiento de la aplicación (**desarrollado por mí**).

```
function initmap() {  
  
    ajaxRequest=getXmlHttpRequest();  
    if (ajaxRequest==null)  
    {  
        alert ("This browser does not support HTTP Request");  
        return;  
    }  
    var osmUrl='http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';  
    var osmAttrib='Map data <a href="http://openstreetmap.org">OpenStreetMap</a>  
contributors';  
    var osm = new L.TileLayer(  
        osmUrl,  
        {minZoom: 1, maxZoom: 30, attribution: osmAttrib}  
    );  
  
    askForPlots();  
    return;  
}
```

Ilustración 31 - Función initmap

La función **initmap** se encarga de inicializar el mapa que se muestra en la vista Map-Osm.html. Esta función se encarga de obtener las vistas del mapa (una vista original de OpenStreetMap y Street y Satellite de Mapbox) así como los elementos que se mostrarán en él.

Esto lo realiza la función *askForPlots*, la cual analiza la respuesta recibida por el microservicio *Buildings* y va situando en el mapa cada uno de los puntos incluidos en la ruta generada. Esto se hace, para cada uno de los tipos de elemento que puede haber en el mapa, con un código como este (específico para cines):

```
while (j != obj[i].cinemaAround.length) {  
    var markerd = L.marker([obj[i].cinemaAround[j].lat, obj[i].cinemaAround[j].lon],  
    {icon: cinemaIcon}).addTo(cines);  
    var customPopUpd = "<p><b>" + obj[i].cinemaAround[j].tags.name + "</b> </p>";  
    markerd.bindPopup(customPopUpd);  
    j++;  
}
```

Ilustración 32 – Añadir elementos al mapa (fragmento de *AskForPlots*)

En cuanto al uso de la metodología AJAX, se realizó un estudio para averiguar qué tipo de peticiones podrían beneficiarse de este tipo de metodología. Las peticiones seleccionadas fueron aquellas referentes a los inicios de sesión, registros y actualizaciones de información del perfil. El resto de peticiones corresponden a cambiar de vista, por lo que es necesaria una recarga de la aplicación en el navegador, algo que no ocurre con las otras tres anteriormente citadas.

En las sucesivas páginas se muestra el código de las peticiones AJAX de inicio de sesión, registro y actualización de información del perfil.

```

function fire_ajax_submit_login() {
    var search = {}
    search["username"] = $("#emailLogin").val();
    search["password"] = $("#passwordLogin").val();

    $("#loginButton").prop("disabled", true);

    if (search["username"].length > 3 && search["password"].length > 3)
    {
        $.ajax({
            type: "POST",
            contentType: "application/json",
            url: "/tryLogin",
            data: JSON.stringify(search),
            dataType: 'json',
            cache: false,
            timeout: 600000,
            success: function (data) {
                if (data.status == "Login OK")
                    {window.location.href = "http://localhost:8080/home";}
                else
                    {$('.error').addClass('alert alert-danger').html("Nombre de
usuario o contraseña incorrectos");}
                shakeModal();
            }
            setTimeout( function(){
                $('#loginModal .modal-dialog').removeClass('shake');
            }, 1000 );
            $("#loginButton").prop("disabled", false);
        },
        error: function (e) {
            $('.error').addClass('alert alert-danger').html("Error en la
petición");
            setTimeout( function(){
                $('#loginModal .modal-dialog').removeClass('shake');
            }, 1000 );
            $("#loginButton").prop("disabled", false);
        }
    });
}
else
{
    $("#loginButton").prop("disabled", false);
    shakeModal();
    $('.error').addClass('alert alert-danger').html("<b>Credenciales
incorrectas</b> " + "<br> - El nombre de usuario debe tener más de 3 caracteres. " +
"<br> - La contraseña debe tener mas de 3 caracteres");
    return (false)}
}

```

Ilustración 33 - Código de la petición AJAX de inicio de sesión

```

function fire_ajax_submit_register() {
    var search = {};
    search["username"] = $("#usernameRegister").val();
    search["password"] = $("#passwordRegister").val();
    search["mail"] = $("#emailRegister").val();
    $("#registerButton").prop("disabled", true);
    if (search["username"].length < 3 || search["password"].length < 3)
    {
        $("#registerButton").prop("disabled", false);
        shakeModal();
        $('.error').addClass('alert alert-danger').html("<b>Credenciales incorrectas</b> " +
            "<br> - El nombre de usuario debe tener más de 3 caracteres. " +
            "<br> - La contraseña debe tener mas de 3 caracteres");
        return (false)
    }
    if (search["password"] != $("#password_confirmationRegister").val())
    {
        $("#registerButton").prop("disabled", false);
        shakeModal();
        $('.error').addClass('alert alert-danger').html("Las contraseñas no coinciden");
        return (false)
    }
    if (/^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/.test(search["mail"]))
    {
        $.ajax({
            type: "POST",
            contentType: "application/json",
            url: "/tryRegister",
            data: JSON.stringify(search),
            dataType: 'json',
            cache: false,
            timeout: 600000,
            success: function (data) {
                if (data.status == "Register OK")
                    {window.location.href = "http://localhost:8080/home";}
                else if (data.status == "Ya existe ese nombre de usuario")
                    {$('.error').addClass('alert alert-danger').html("Ya existe ese nombre de
usuario");}
                shakeModal();
            }
            else if (data.status == "Error en los campos de entrada")
                {$('.error').addClass('alert alert-danger').html("Error en los campos de
entrada");shakeModal();}
                setTimeout( function(){
                    $('#loginModal .modal-dialog').removeClass('shake');
                }, 1000 );
                $("#registerButton").prop("disabled", false);
            },
            error: function (e) {
                $('.error').addClass('alert alert-danger').html("Error en la petición");
                setTimeout( function(){
                    $('#loginModal .modal-dialog').removeClass('shake');
                }, 1000 );
                $("#registerButton").prop("disabled", false);
            }
        });
    }
    else
    {
        $("#registerButton").prop("disabled", false);
        shakeModal();
        $('.error').addClass('alert alert-danger').html("La dirección de correo no es válida.");
        return (false) } }

```

Ilustración 34 - Código de la petición AJAX de registro

```

function fire_ajax_submit_update() {
    var search = {};
    search["username"] = $("#usernameUpdate").val();
    search["password"] = $("#passwordUpdate").val();
    search["mail"] = $("#emailUpdate").val();
    $("#updateButton").prop("disabled", true);
    if (search["username"].length < 3 || search["password"].length < 3)
    {
        $("#updateButton").prop("disabled", false);
        $('.error').addClass('alert alert-danger').html("<b>Credenciales incorrectas</b> " +
            "<br> - El nombre de usuario debe tener más de 3 caracteres. " +
            "<br> - La contraseña debe tener mas de 3 caracteres");
        return (false)
    }
    if (/^[\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/.test(search["mail"]))
    {
        $.ajax({
            type: "POST",
            contentType: "application/json",
            url: "/tryUpdate",
            data: JSON.stringify(search),
            dataType: 'json',
            cache: false,
            timeout: 600000,
            success: function (data) {
                if (data.status == "Update OK")
                {
                    window.location.href = "http://localhost:8080/home";
                }
                else if (data.status == "Ya existe ese nombre de usuario")
                {
                    $('.error').addClass('alert alert-danger').html("Ya existe
ese nombre de usuario");
                    shakeModal();
                }
                else if (data.status == "Error en los campos de entrada")
                {
                    $('.error').addClass('alert alert-danger').html("Error en los
campos de entrada");
                    shakeModal();
                }
                $("#updateButton").prop("disabled", false);
            },
            error: function (e) {
                $('.error').addClass('alert alert-danger').html("Error en la
petición");
                $("#updateButton").prop("disabled", false);
            }
        });
    }
    else
    {
        $("#updateButton").prop("disabled", false);
        $('.error').addClass('alert alert-danger').html("La dirección de correo no es
válida.");
        return (false)}
    }
}

```

Ilustración 35 - Código de la petición AJAX de actualización

5.3.3. Hojas de estilo CSS

Si bien se ha desarrollado parte del código CSS, **la mayoría del mismo ha sido obtenida de plantillas y otros recursos en internet, tomando siempre especial cuidado de que tuvieran las licencias libres adecuadas para su uso.** No obstante, más allá de simplemente obtener las hojas de estilo, ha sido necesario modificarlas para que se ajustasen a lo que de verdad se quería obtener.

Uno de los aspectos que ha sido necesario implementar ha sido el diseño responsive. Actualmente, los usuarios no navegan únicamente por un tipo de dispositivo. Pueden utilizar un ordenador cuando estén en sus casas o trabajos, o dispositivos con una resolución menor, como pueden ser teléfonos o tabletas, cuando están viajando o moviéndose. Es por eso que en este proyecto no se ha querido dejar escapar la oportunidad de diseñar una aplicación *responsive*, que sea capaz de responder a los tres grandes tipos de dispositivos: ordenadores, tabletas y móviles.

El diseño *responsive* trata de redimensionar, colocar y adaptar los distintos elementos que contiene la aplicación web en función del tamaño de la pantalla desde la cual se está visualizando el sistema. Para poder realizar este tipo de diseños, es necesario utilizar *media queries* que son parte de CSS versión 3. Un buen diseño *responsive* se caracteriza pues por los siguientes aspectos:

- **Layouts e imágenes fluidos y adaptados** a la pantalla del dispositivo en cuestión.
- Implica una **reducción del tiempo de desarrollo**.
- **Evita duplicidades**.
- **Aumenta la viralidad** de los contenidos, ya que permite que se compartan de una manera más veloz y natural.

Si bien las ilustraciones anteriores muestran cómo luce la aplicación para ordenadores, se mostrarán ahora ejemplos de cómo se ve la aplicación en tabletas y móviles.

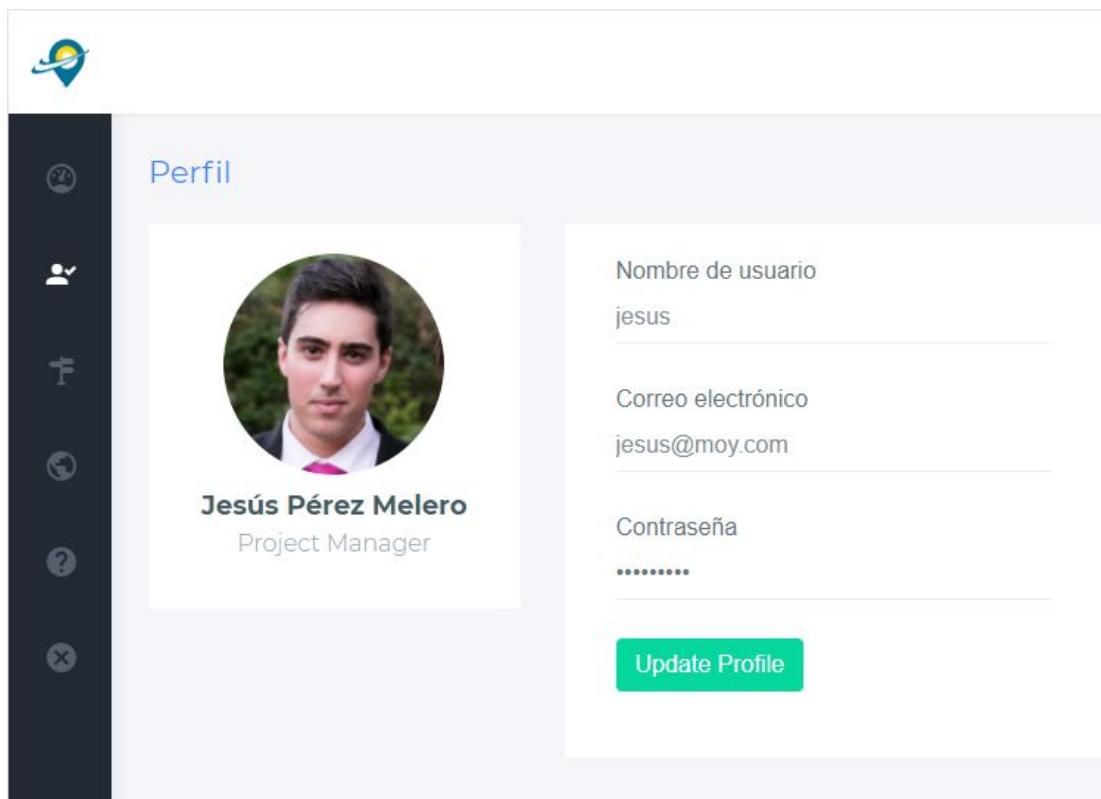


Ilustración 36 - Vista de perfil (Tableta)

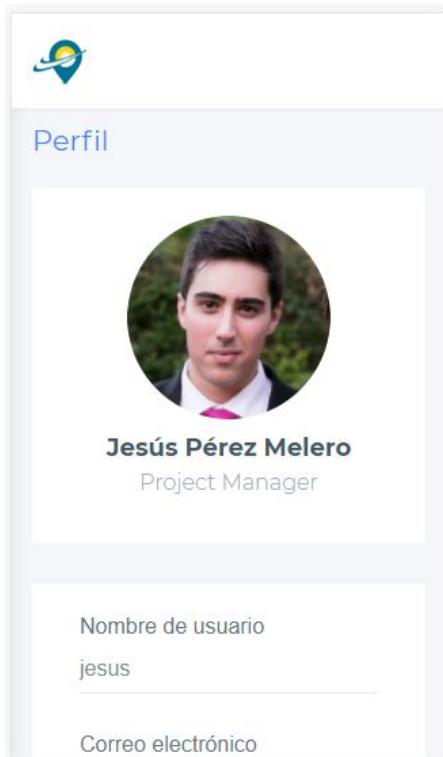


Ilustración 37 - Vista de perfil (Móvil)

Rutas turísticas

Localización
Indica los lugares por los que quieras que pase tu ruta. Puedes elegir una ruta que contemple toda la ciudad, o un distrito que te gustaría visitar.

Todo Madrid

Horario
Si no especificas un horario, se planificará una ruta para todo el día. En caso de que definas un horario, aparecerán lugares que podrás visitar en las horas que has indicado.

Comienzo

Fin

Tipos de edificios
Puedes elegir el tipo de edificios que contendrá tu ruta. Actualmente tenemos disponibles los siguientes tipos de edificios:

Monumentos históricos

Ilustración 38 - Vista de definición de ruta (Tableta)

Tipos de edificios
Puedes elegir el tipo de edificios que contendrá tu ruta. Actualmente tenemos disponibles los siguientes tipos de edificios:

Monumentos históricos

Tamaño de la ruta
Por defecto, las rutas que genera Madrid on You contienen 5 localizaciones. Si quieres definir un número distinto, puedes hacerlo. Cuanto más grande sea la ruta, más tardaremos en prepararla.

Numero de monumentos

Ilustración 39 - Vista de definición de ruta (Móvil)

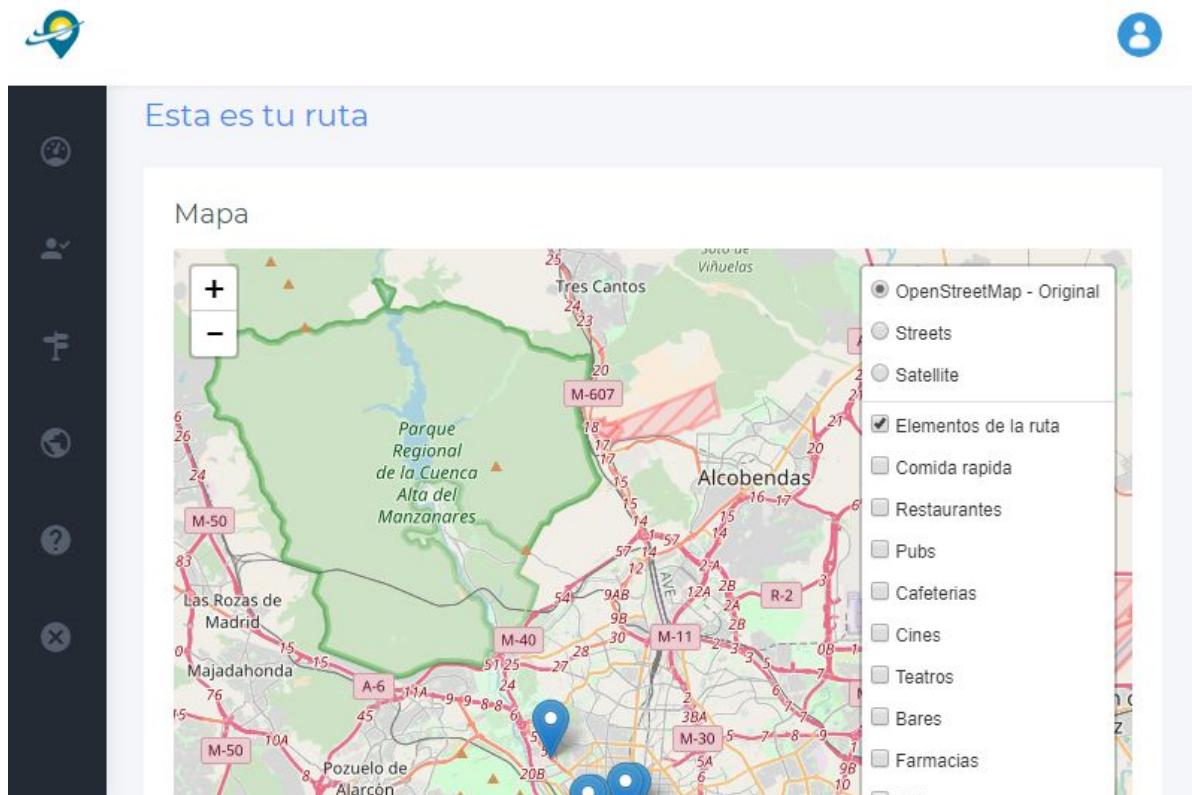


Ilustración 40 - Mapa con los elementos de la ruta (Tableta)

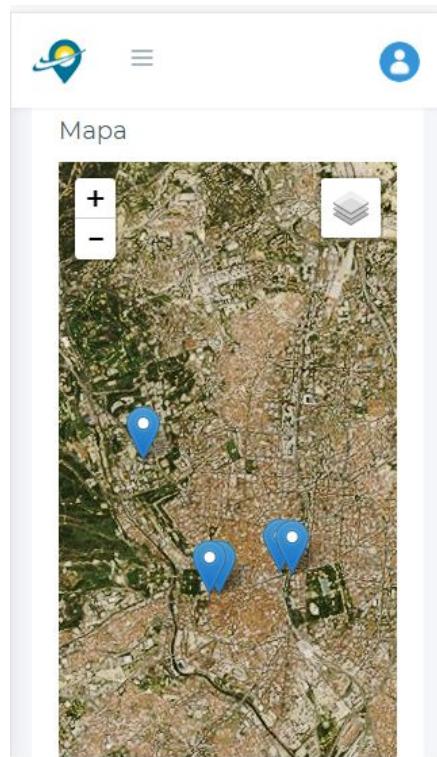


Ilustración 41 - Mapa con los elementos de la ruta (Móvil)

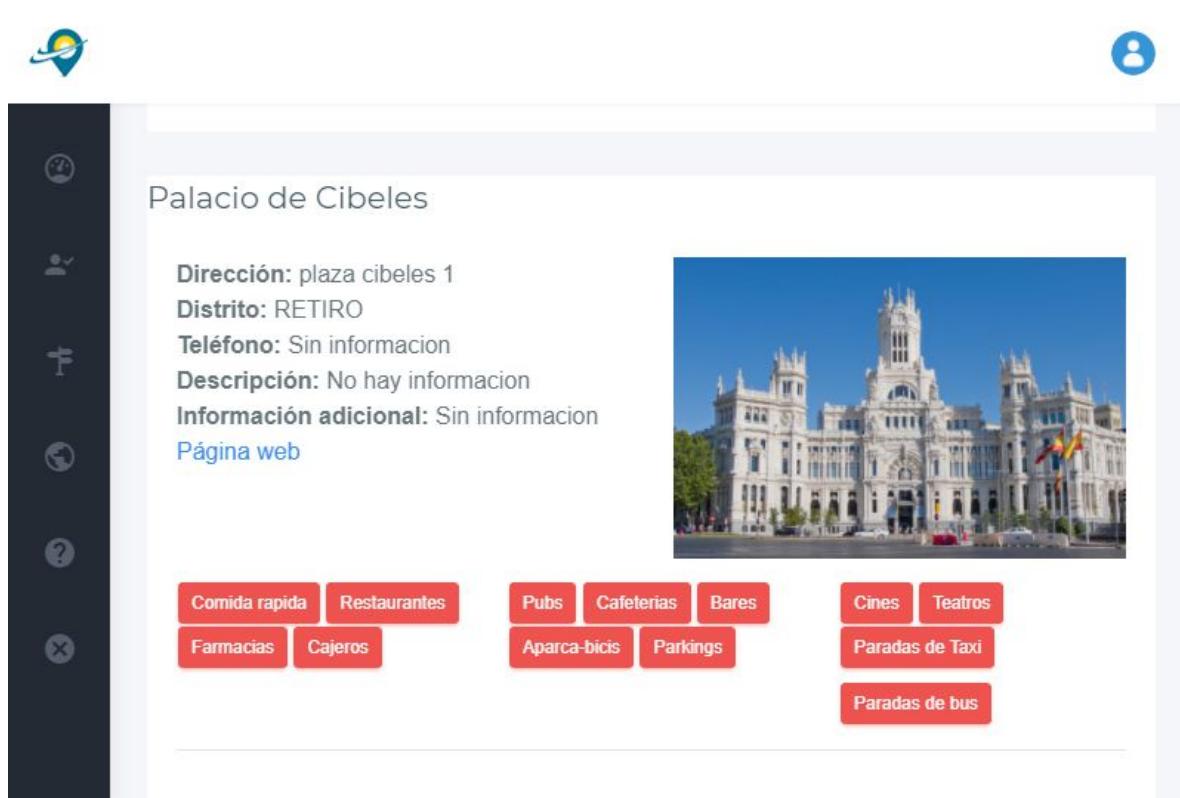


Ilustración 42 - Elemento de la ruta (Tableta)



Ilustración 43 - Elemento de la ruta (Móvil)

5.4. Back – End

El back-end de la aplicación sigue una arquitectura basada en **microservicios** o MSA (*MicroServices Architecture*). Esto quiere decir que el back está compuesto por un conjunto de pequeños servicios que se comunican entre sí mediante APIs bien definidas con peticiones HTTP. Cada uno de los microservicios se encarga de dar soporte a un área concreta del sistema. A continuación se detallan cada uno de los elementos que componen el back-end.

5.4.1. User Management

Este servicio realiza las funcionalidades oportunas para registrar y, por tanto dar de alta un nuevo usuario, y autorizar o denegar un *login* en la aplicación. Para ello se comunica con la capa SaaS, donde se encuentra una tabla de Dynamo DB que contiene toda la información de los usuarios de la aplicación.

Dentro de la carpeta main se encuentran los dos paquetes que componen el proyecto Eclipse de este microservicio. El paquete principal contiene la clase *LoginApplication* que sirve para arrancar el micro.

En el paquete *api* encontramos la API del microservicio y el controlador que implementa dicha interfaz.

Como parte del desarrollo, se han desarrollado varios test unitarios que sirven para probar el correcto funcionamiento de este microservicio. Serán vistos en detalle en una sección sucesiva.

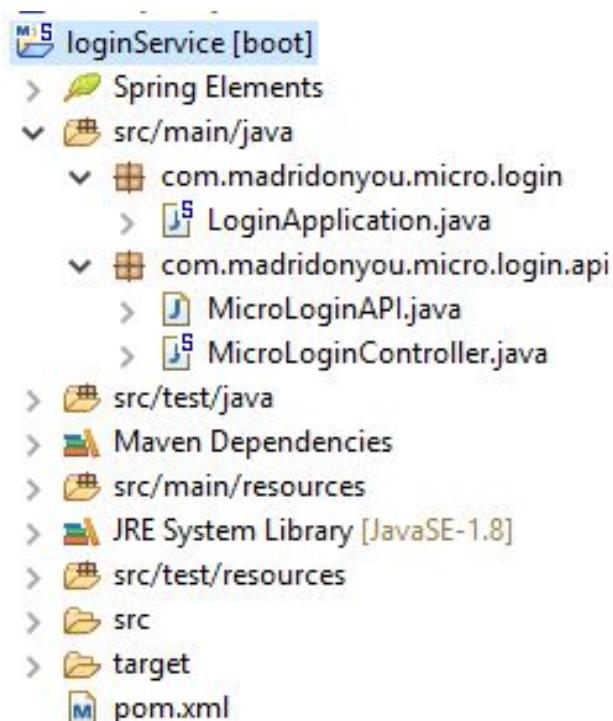


Ilustración 44 - Estructura del Micro de User Management

La API de este microservicio contiene las siguientes operaciones:

Mapping	Tipo	Entrada	Salida
/login	POST	Application/json	Application/json
/register	POST	Application/json	Application/json

Tabla 3 - API de User Management

Cada uno de los métodos de la interfaz espera una entrada concreta. En el caso del login, la entrada está formada únicamente por nombre de usuario y contraseña. En el caso del registro, está formada por lo anterior más un e-mail.

```
{
  "username": "jesus",
  "password": "testt"
}
```

Ilustración 45 - Input para login

```
{
  "username": "jesus",
  "password": "testt",
  "mail": "jesusperezmelero
@google.com"
}
```

Ilustración 46 - Input para register

A continuación se muestran los diagramas UML asociados a cada clase presente en este microservicio, así como un UML que agrupa todas las clases. Nótese que en algunos casos se omitirán los nombres de ciertos métodos (getters & setters) debido al tamaño de la imagen, pero es algo que no oscurece ni dificulta su comprensión.

<p><<Java Class>></p> <p>LoginInput</p> <p>com.madridonyou.micro.domain.inputs</p> <ul style="list-style-type: none"> ▫ username: String ▫ password: String ▫ additionalProperties: Map<String, Object> <ul style="list-style-type: none"> ● LoginInput() ● getUsername(): String ● setUsername(String): void ● getPassword(): String ● setPassword(String): void ● getAdditionalProperties(): Map<String, Object> ● setAdditionalProperty(String, Object): void ● getProperties(): Iterator<String> 	<p><<Java Class>></p> <p>RegisterInput</p> <p>com.madridonyou.micro.domain.inputs</p> <ul style="list-style-type: none"> ▫ username: String ▫ password: String ▫ mail: String ▫ additionalProperties: Map<String, Object> <ul style="list-style-type: none"> ● RegisterInput() ● getUsername(): String ● setUsername(String): void ● getPassword(): String ● setPassword(String): void ● getMail(): String ● setMail(String): void ● getAdditionalProperties(): Map<String, Object> ● setAdditionalProperty(String, Object): void ● getProperties(): Iterator<String>
<p><<Java Class>></p> <p>LoginOutput</p> <p>com.madridonyou.micro.domain.outputs</p> <ul style="list-style-type: none"> ▫ status: String ▫ additionalProperties: Map<String, Object> <ul style="list-style-type: none"> ● LoginOutput() ● getStatus(): String ● setStatus(String): void ● getAdditionalProperties(): Map<String, Object> ● setAdditionalProperty(String, Object): void 	<p><<Java Class>></p> <p>RegisterOutput</p> <p>com.madridonyou.micro.domain.outputs</p> <ul style="list-style-type: none"> ▫ status: String ▫ additionalProperties: Map<String, Object> <ul style="list-style-type: none"> ● RegisterOutput() ● getStatus(): String ● setStatus(String): void ● getAdditionalProperties(): Map<String, Object> ● setAdditionalProperty(String, Object): void

Ilustración 47 - Clases utilizadas en User Management

Las clases utilizadas por este microservicio como entradas y salidas se muestran en la ilustración. Tienen los atributos y métodos necesarios para efectuar un login, registro o actualización (se usan los mismos objetos que en el registro) en la aplicación.

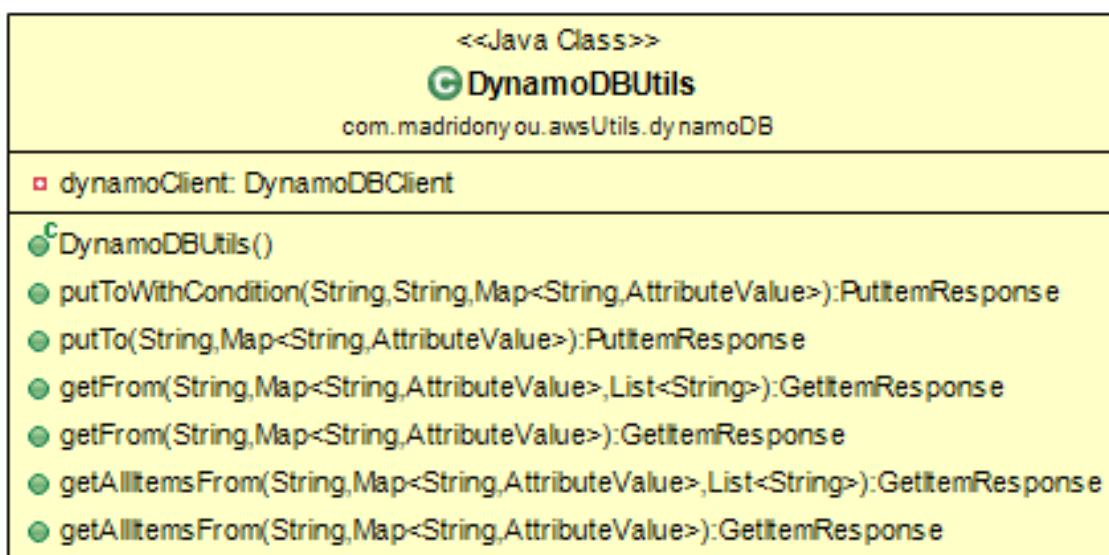


Ilustración 48 - UML de la clase que interactúa con Dynamo DB

Para interactuar con Dynamo DB se utiliza la clase que se muestra en la ilustración. Dispone de un cliente que se encarga de ejecutar las acciones necesarias para recuperar o insertar información en la tabla asociada a los usuarios, donde la clave es el ID de usuario. A continuación se muestra una captura de dicha tabla:

Scan: [Table] moy_users: userId ▲				
	Scan	[Table] moy_users: userId	+ Add filter	Start search
	userId	name	password	username
<input type="checkbox"/>	101016374	jesus	3556498	jesus

Ilustración 49 - Tabla en Dynamo DB

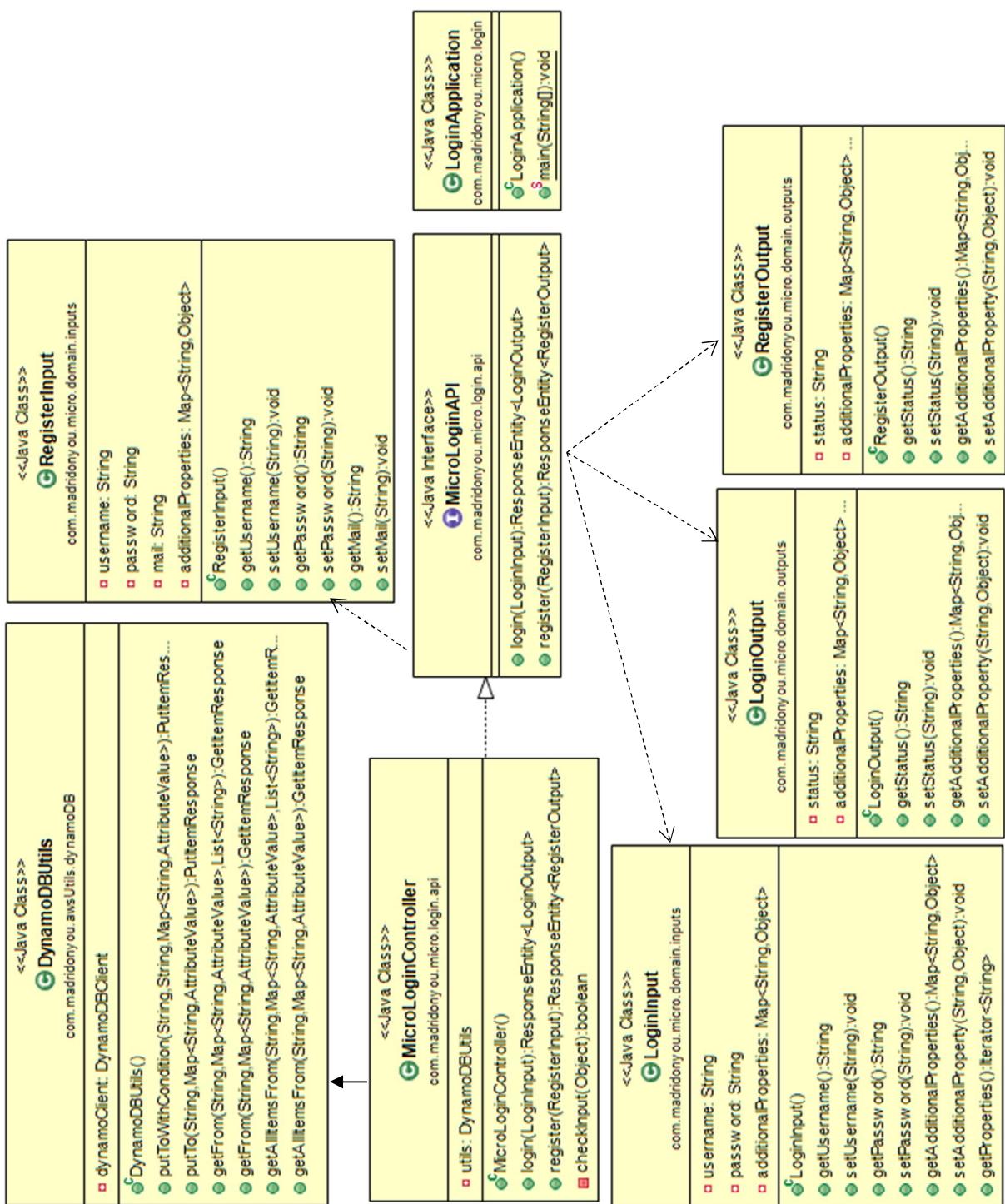


Ilustración 50 - Diagrama UML de clases de User Management

5.4.2. Buildings Management

Este servicio realiza las funcionalidades oportunas para generar las rutas que serán mostradas a los usuarios. Hace uso de un módulo SPARQL que realiza queries contra los ficheros RDF generados para obtener información de los recursos, ya sea únicamente utilizando los RDF o accediendo a DBpedia y esDBpedia para enriquecer la información, y un módulo que envía peticiones a la API Overpass para obtener información de Open Street Map.

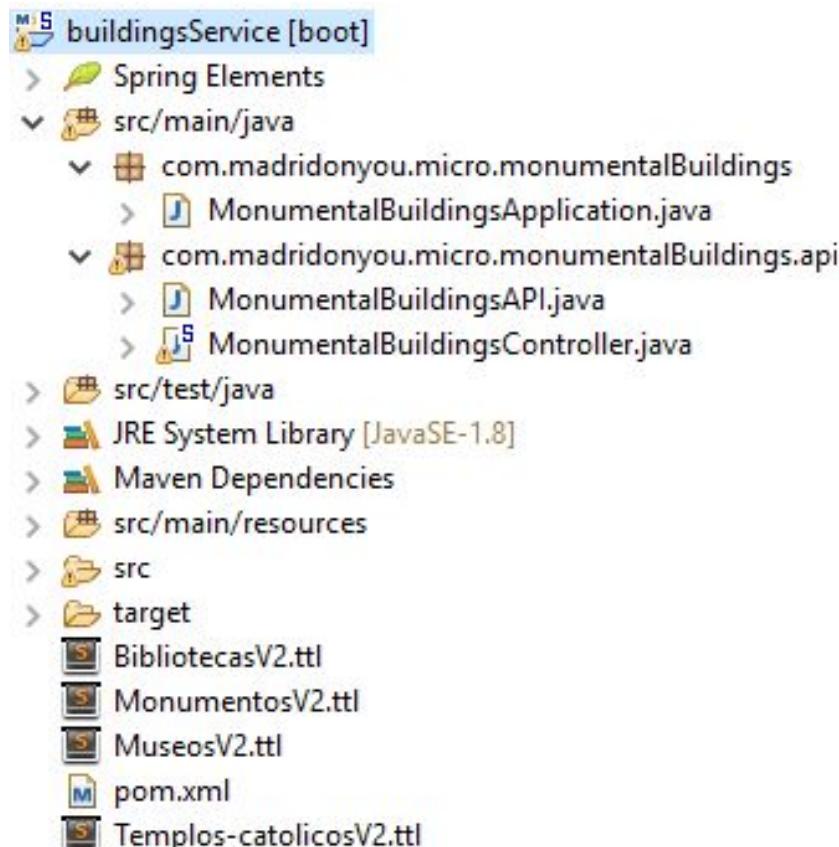


Ilustración 51 - Estructura del micro de Buildings Management

Dentro de la carpeta main se encuentran los dos paquetes que componen el proyecto Eclipse de este microservicio. El paquete principal contiene la clase *MonumentalBuildingsApplication* que sirve para arrancar el micro.

En el paquete *api* encontramos la API del microservicio y el controlador que implementa dicha interfaz. Al igual que en el caso anterior, se han desarrollado varios tests.

La API de este microservicio contiene las siguientes operaciones:

Mapping	Tipo	Entrada	Salida
/random	POST	Application/json	Application/json

Tabla 4 – API de Buildings Management

La entrada que espera el método *random* está compuesta por los parámetros necesarios para componer una ruta: el distrito (para generar una ruta por un distrito concreto o por todo Madrid), las horas de inicio y final, el tamaño (número de elementos que contendrá la ruta) y la categoría (tipos de edificios que contendrá la ruta).

```
{
    "district": "CENTRO",
    "horaIni": "17:50",
    "horaFin": "22:00",
    "size": 5,
    "categoría": "HistoricBuilding"
}
```

Ilustración 52 - Input para random

A continuación se muestran los diagramas UML asociados a cada clase presente en este microservicio, así como un UML que agrupa todas las clases. Nótese que en algunos casos se omitirán los nombres de ciertos métodos (getters & setters) por motivos de espacio, pero es algo que no oscurece ni dificulta su comprensión.

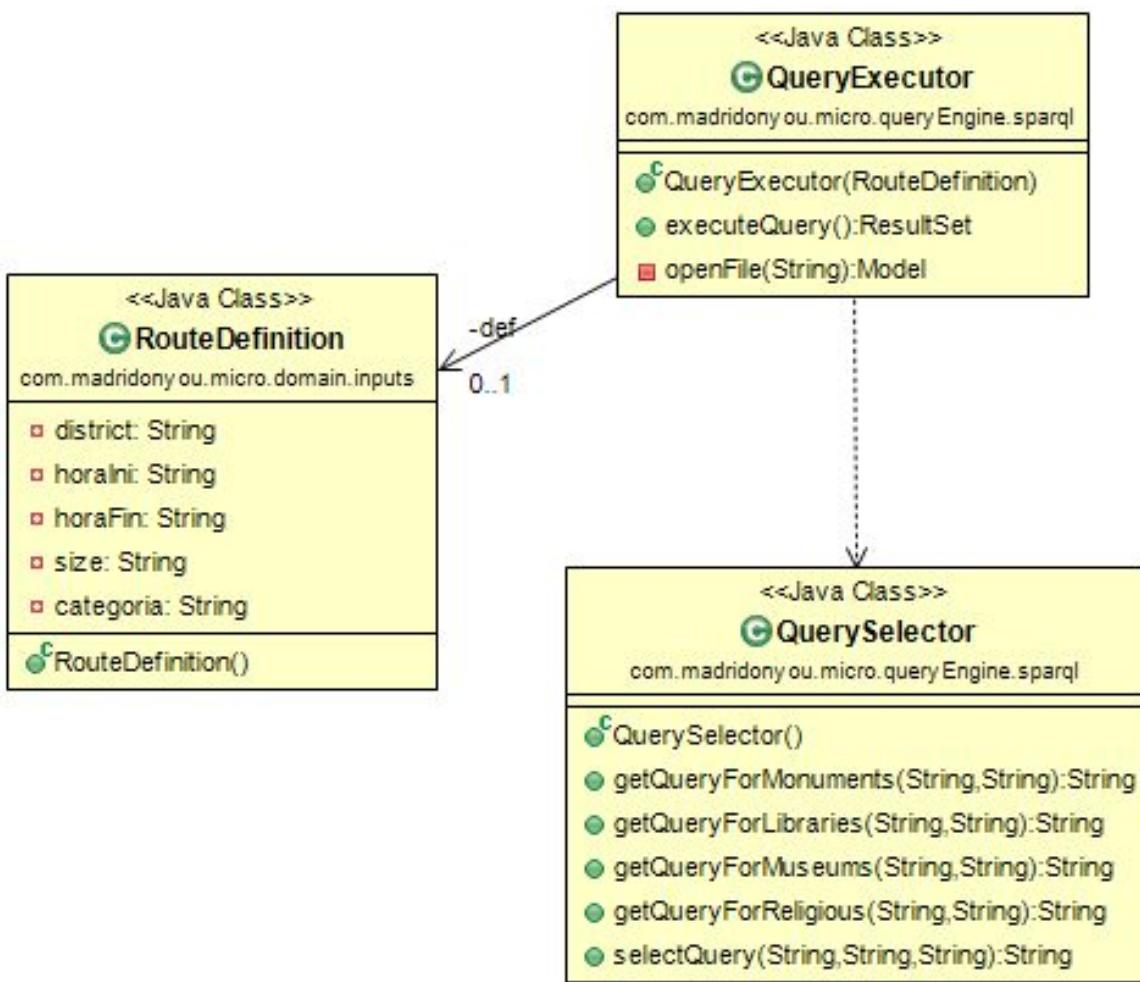


Ilustración 53 - UML de las clases que realizan queries contra los RDF y DBpedia

Para poder realizar consultas SPARQL a los RDF generados será necesario hacer uso de las clases que se muestran en la ilustración. Dichas clases permitirán, a través de la definición de la ruta, seleccionar y ejecutar la query correspondiente en cada caso.

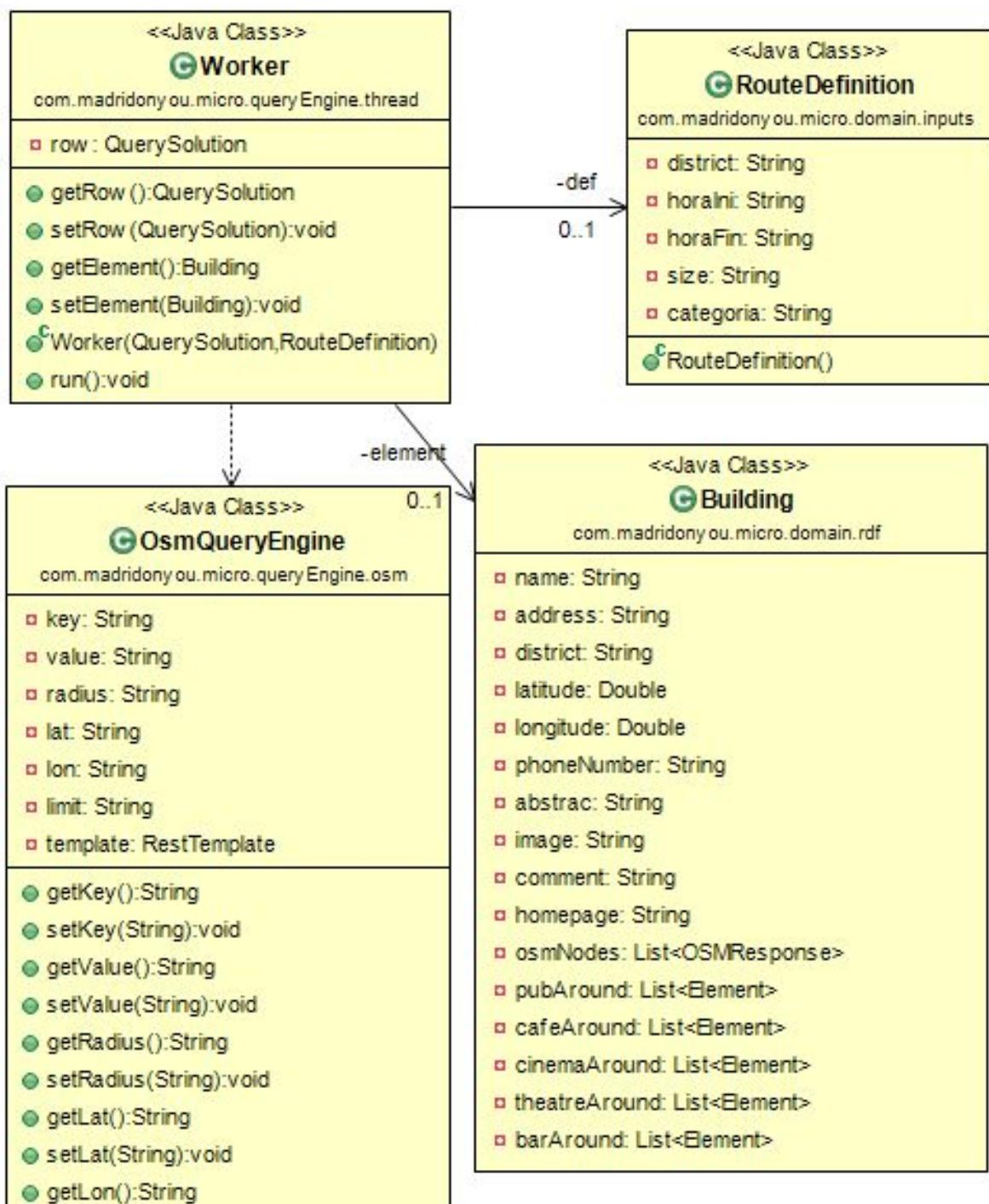


Ilustración 54 - UML de las clases que interactúan con OSM

Para interactuar con OSM se utiliza una clase Thread que lanza peticiones a la API Overpass y encapsula en objetos Java las respuestas recibidas. El detalle de las clases necesarias para el mapeo a objetos Java se encuentra en la siguiente ilustración.

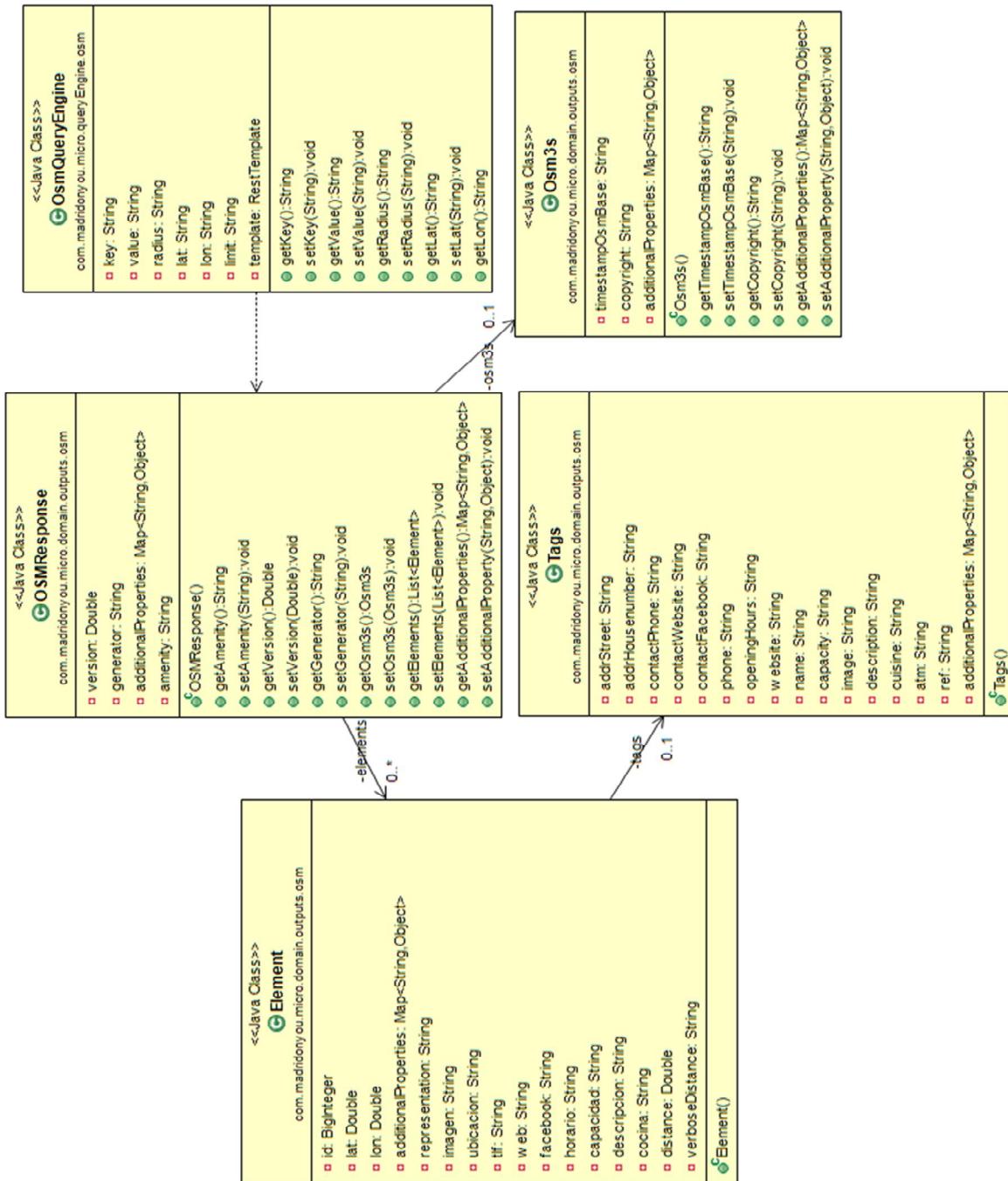


Ilustración 55 - UML de las clases necesarias para mapear la respuesta de OSM

La respuesta que se recibe de OSM debe ser mapeada en varias clases Java para poder ser utilizada posteriormente. En la ilustración se muestran las clases necesarias para ello.

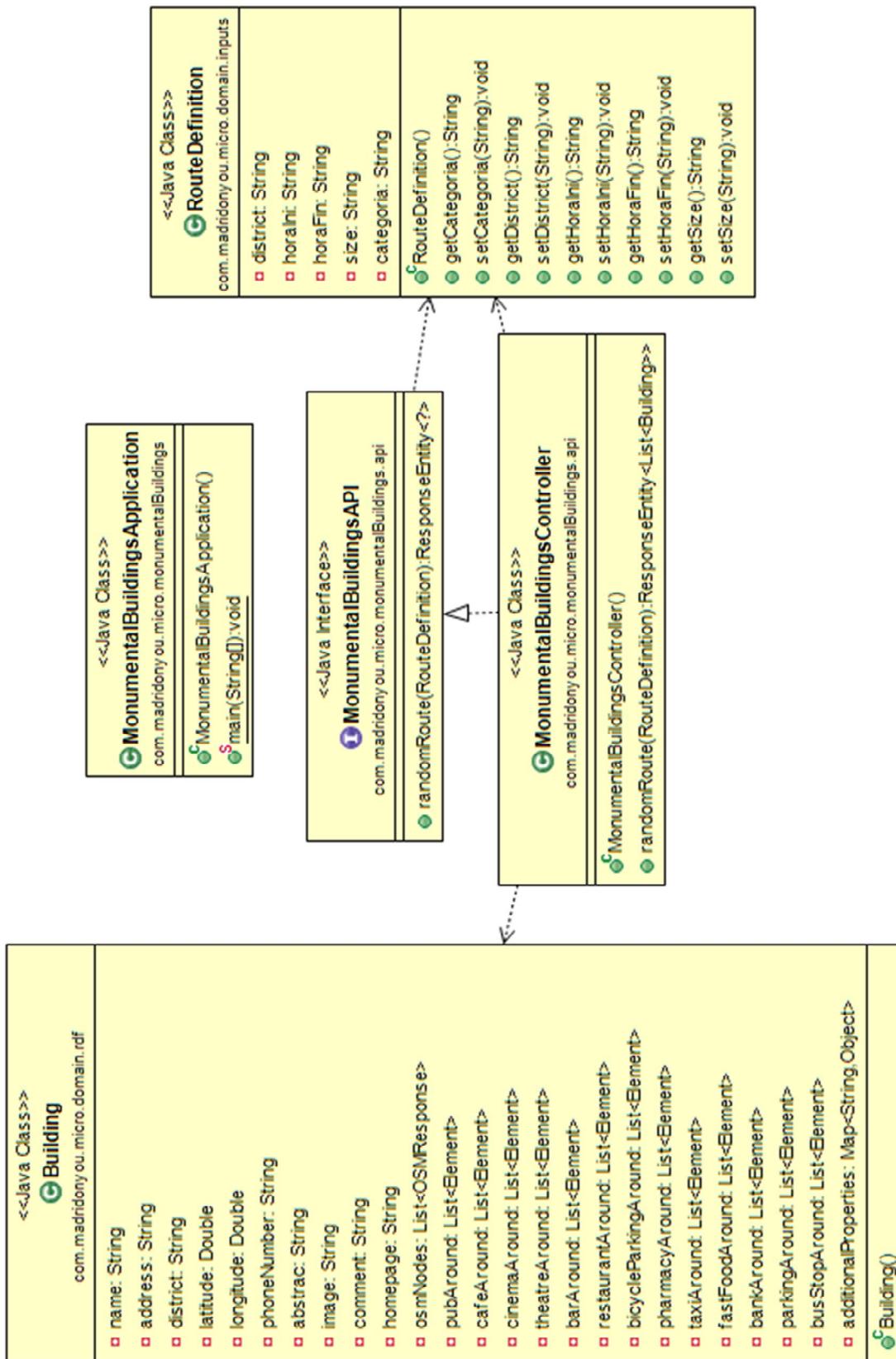


Ilustración 56 - Diagrama UML de clases de Buildings Management

Como se ha indicado anteriormente, este microservicio se encarga de consultar los ficheros RDF generados mediante consultas SPARQL. Un ejemplo de consulta a realizar es (se omiten los prefijos):

```

SELECT DISTINCT ?name ?address ?district ?latitude ?longitude
?phoneNumber ?image ?url ?abstract WHERE {

?a a "categoria" .
?a dbo:name ?name .
?a dbo:address ?address .
?a geo:lat ?latitude .
?a geo:long ?longitude .
?a dbp:phoneNumber ?phoneNumber .
?a dbp:url ?url .

OPTIONAL {
?a owl:sameAs ?uri
OPTIONAL {
SERVICE <http://dbpedia.org/sparql> {
?uri foaf:depiction ?image .
OPTIONAL {
?uri dbo:abstract ?abstract .
FILTER langMatches(lang(?abstract), 'es')
}
}
}
OPTIONAL {
SERVICE <http://es.dbpedia.org/sparql> {
?uri foaf:depiction ?image
}
}
OPTIONAL {
SERVICE <http://es.dbpedia.org/sparql> {
?uri dbo:wikiPageRedirects ?another .
?another foaf:depiction ?image .
}
}
}

?a dbo:district ?location .
?location dbo:name "district" .

}
ORDER BY RAND()
LIMIT "size"

```

Ilustración 57 - Consulta SPARQL

Los tres campos señalados en rojo en la consulta corresponden a los valores que reciba el servicio en su entrada. De este modo, las queries que se ejecutan siguen el mismo patrón pero varían en cuanto a la categoría, distrito y tamaño de las respuestas.

Este microservicio también se comunica con la API Overpass para obtener información de OpenStreetMap, mediante una petición HTTP con el script que contiene la query encapsulado en un mensaje XML.

```
<osm-script>

<query type="node">
    <around radius="150" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="pub"/>
</query>
<print limit="5"/>

<query type="node">
    <around radius="500" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="cafe"/>
</query>
<print limit="10"/>

<query type="node">
    <around radius="500" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="cinema"/>
</query>
<print limit="5"/>

<query type="node">
    <around radius="500" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="theatre"/>
</query>
<print limit="5"/>

<query type="node">
    <around radius="150" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="bar"/>
</query>
<print limit="5"/>

<query type="node">
    <around radius="250" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="restaurant"/>
</query>
<print limit="6"/>

<query type="node">
    <around radius="500" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="bicycle_parking"/>
</query>
<print limit="5"/>

<query type="node">
    <around radius="500" lat="40.4188346862793" lon="-3.6948211193084717"/>
    <has-kv k="amenity" v="pharmacy"/>
</query>
<print limit="5"/>

</osm-script>
```

Ilustración 58 - Script XML para la API Overpass

El script consta de varias queries en las que se obtiene información próxima a una localización concreta. En el script de ejemplo las coordenadas 40.4188346862793 y -3.6948211193084717 corresponden al Banco de España. Cada query sirve para obtener elementos próximos de distinta índole. En el ejemplo podemos ver que se solicitan como máximo 5 nodos de OSM que sean pubs, en un radio de 150 metros, o 10 nodos como máximo que correspondan a cafeterías.

Cada uno de los nodos contenidos en la respuesta a esta petición tiene propiedades adicionales como nombre, imagen, descripción, números de teléfono, páginas de Facebook, comentarios... de forma que únicamente con la latitud y la longitud del elemento se obtienen cantidades ingentes de información.

Sin embargo no todo es perfecto. La API Overpass tiene una limitación en el número de peticiones por segundo que pueden realizarse. **Sólo se permite realizar una petición por segundo**, de lo contrario las peticiones sucesivas serán extremadamente lentas y puede darse el caso de que se reciba un error 429 (*Too Many Requests*).

Por tanto, **OSM es la mayor fuente de datos de la que dispone la aplicación construida, pero también es un cuello de botella** a tener en cuenta ya que para generar una ruta de 5 elementos será necesario esperar como mínimo 5 segundos para que las peticiones a OSM se lleven a cabo. A este tiempo habrá que sumarle el que se tarde en procesar peticiones y las comunicaciones entre microservicios.

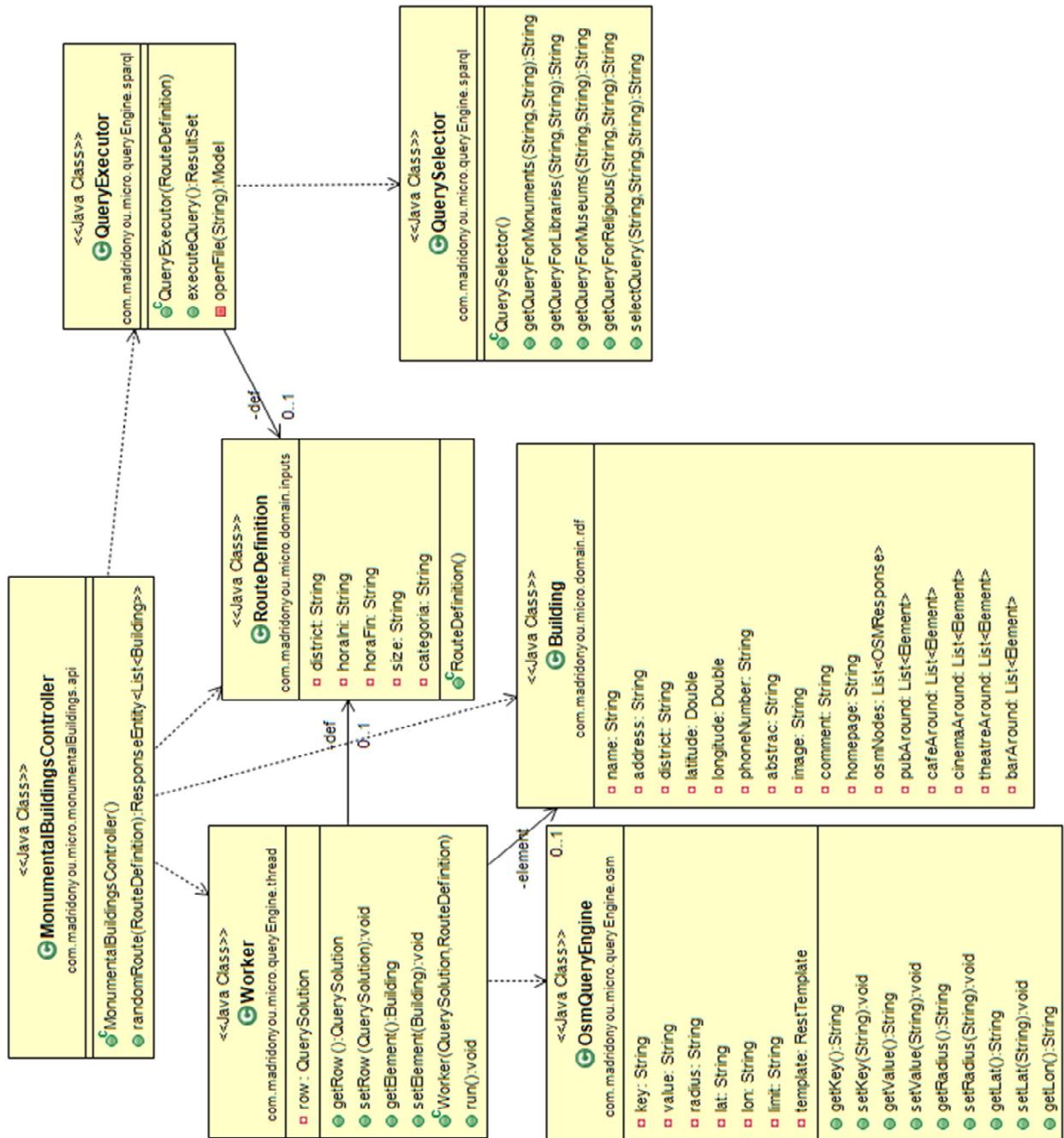


Ilustración 59 - Diagrama UML de clases relativas a integración con DBpedia y OSM

5.4.3. Dispatcher

El *dispatcher* únicamente se encarga de redirigir peticiones del front hacia el back y viceversa. Es por ello que sólo se mostrará la API que ofrece al back para interactuar con el front.

Mapping	Tipo	Entrada	Salida
/forward	POST	Application/json	Application/json

Tabla 5 - API del dispatcher

Cada petición recibida por el dispatcher será analizada y, en función del tipo de petición que sea (ya se han visto en anteriores diagramas los distintos Input's que contempla la aplicación) redirigirá la petición al microservicio correspondiente. La respuesta de dicho microservicio será inmediatamente devuelta al front, sin realizar ningún tipo de comprobación o ajuste, dejando dicha tarea al front, que será el encargado de mostrar o no mostrar cierta información en función del valor que tenga (por ejemplo, imágenes que no se puedan visualizar).

5.4.3. Mail Engine

Este microservicio se encarga de enviar correos electrónicos. Sólo dispone de una única operación, la cual recibe como entrada un JSON con el asunto, destinatario y cuerpo del mensaje.

Mapping	Tipo	Entrada	Salida
/sendMail	POST	Application/json	Application/json

Tabla 6 - Api de Mail Engine

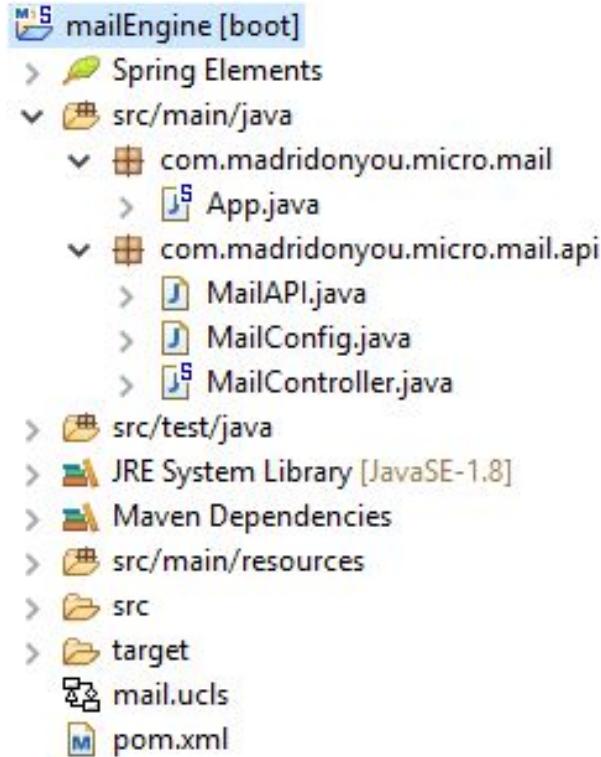


Ilustración 60 - Estructura del proyecto mail Engine

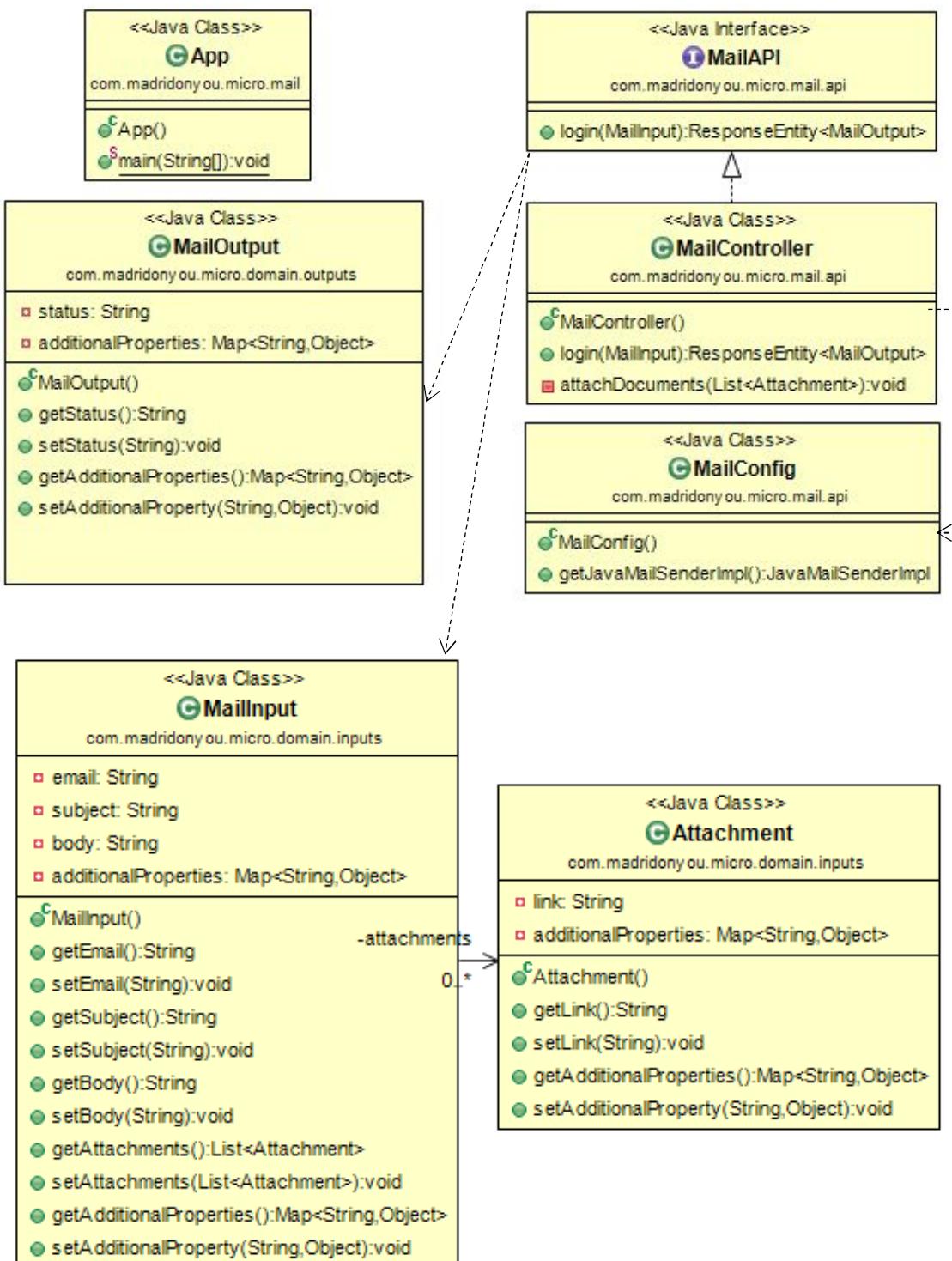


Ilustración 61 - UML de las clases de Mail Engine

5.5. Ontología

Para desarrollar la ontología se ha utilizado Protégé, que como se indicó anteriormente es un editor *open-source* de ontologías y también un *framework* para el desarrollo de sistemas inteligentes.

La ontología se divide principalmente en tres categorías:

- **Clases:** Son en esencia la agrupación de conceptos similares. Por ejemplo, “ordenador” podría ser una clase donde “portátil”, “sobremesa”, “Workstation” serían subclases.
- **Relaciones:** Representan la interacción entre los diferentes conceptos (clases) del dominio.
- **Propiedades:** Representan cualidades atribuibles a las clases.

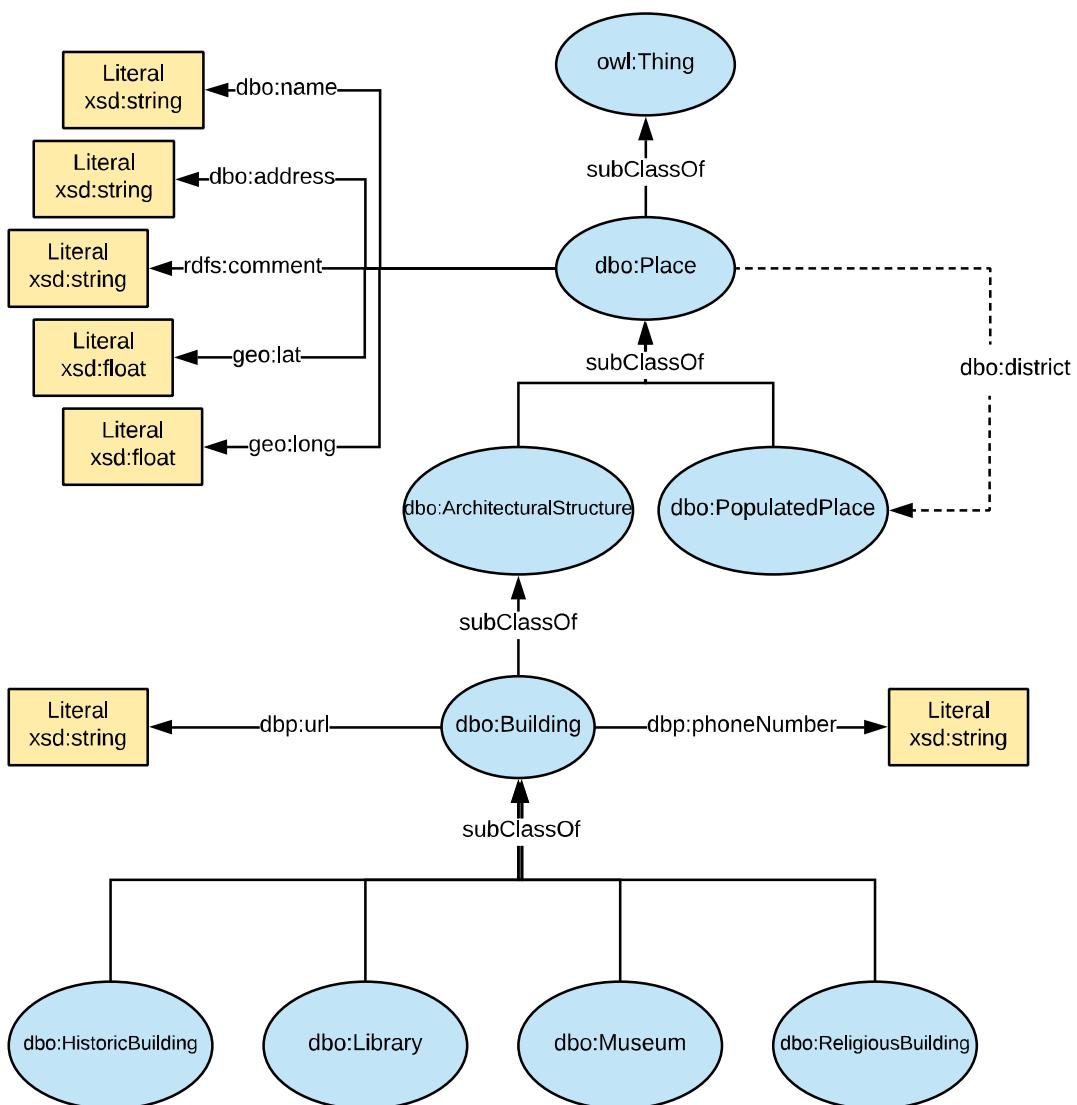


Ilustración 62 - Ontología desarrollada

5.5.1. Clases



Ilustración 63 - Clases de la ontología

La clase *Thing* es la clase padre de cualquier tipo de entidad, por lo que se sitúa en lo alto de la jerarquía de clases. De acuerdo con el problema que se quiere resolver, es adecuado modelizar cualquier tipo de monumento, edificio, parque, etc... como un lugar, que es lo que se representa mediante la clase *Place*. La clase *Place* tiene dos clases hijas, que son *ArchitecturalStructure*, que es necesaria para modelizar todos los monumentos y edificios, y por otro lado, *PopulatedPlace*, que hace referencia a un lugar que es parte de una ciudad y está poblado, lo cual es idóneo para representar los distritos a los cuales pertenecen los elementos de las rutas.

Dentro de *ArchitecturalStructure* podemos observar que solo modelizaremos elementos de la clase *Building*, que son los que aplican en este caso, y que podrán ser de tres tipos, en función del tipo de edificio en cuestión.

5.5.2. Relaciones

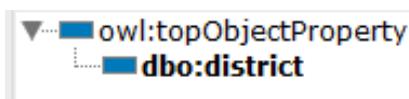


Ilustración 64 - Relaciones de la ontología

En Protégé, se utilizan las *ObjectProperties* para definir relaciones entre las distintas clases que se tienen (las indicadas en la sección anterior en este caso). En la ontología desarrollada únicamente es necesaria una relación, la cual es *district*, que tiene como dominio de la relación un elemento de tipo *Place* y como objetivo un elemento de

tipo *PopulatedPlace*. Esta relación es necesaria para poder representar la relación de pertenencia de una localización a un distrito.

5.5.3. Propiedades

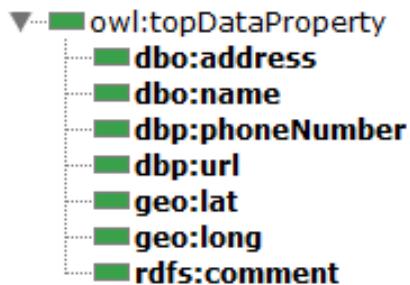


Ilustración 65 - Propiedades de la ontología

En Protégé, se utilizan las *DataProperties* para definir las propiedades aplicables a las distintas clases que componen la ontología. Las propiedades que aplican en este caso concreto son las siguientes:

- **Address:** Cadena que representa la dirección del elemento.
- **Name:** Cadena que representa el nombre del elemento.
- **PhoneNumber:** Cadena que representa el número de teléfono del elemento.
- **Url:** Cadena que representa el enlace web del elemento.
- **Lat:** Coma flotante de doble precisión que representa la latitud del elemento.
- **Long:** Coma flotante de doble precisión que representa la longitud del elemento.
- **Comment:** Cadena que indica algún tipo de comentario o información adicional sobre el elemento.

5.6. Ficheros RDF

Como se ha indicado en el capítulo 3, se han obtenido datos abiertos del portal de datos abiertos de Madrid. Los conjuntos descargados tenían formato CSV, es decir, todos los campos se encuentran separados por algún tipo de delimitador (por defecto la coma). Es habitual que este tipo de conjuntos de datos contengan información que no sea útil para un caso de uso concreto. En caso de ser así, es necesario tratar el dataset para que contenga la información estrictamente útil y necesaria. Para ello se ha utilizado LOD Refine, eliminando partes innecesarias, repetitivas o incorrectas de los datos. Una vez tratados, los conjuntos de datos han sido adaptados a la ontología anterior de forma que pudiesen ser transformados a ficheros RDF sobre los cuales ejecutar consultas con el lenguaje SPARQL.

Cabe destacar que gracias a LOD Refine se realiza una *reconciliación* de nuestros datos contra datos existentes en DBpedia o en esDBpedia. Por ejemplo, si estamos tratando el elemento “Castillo de la Alameda”, LOD Refine será capaz de enlazar nuestro recurso (con una información muy limitada) con su recurso existente de esDBpedia (http://es.dbpedia.org/page/Castillo_de_La_Alameda), obteniendo así muchísima más información.

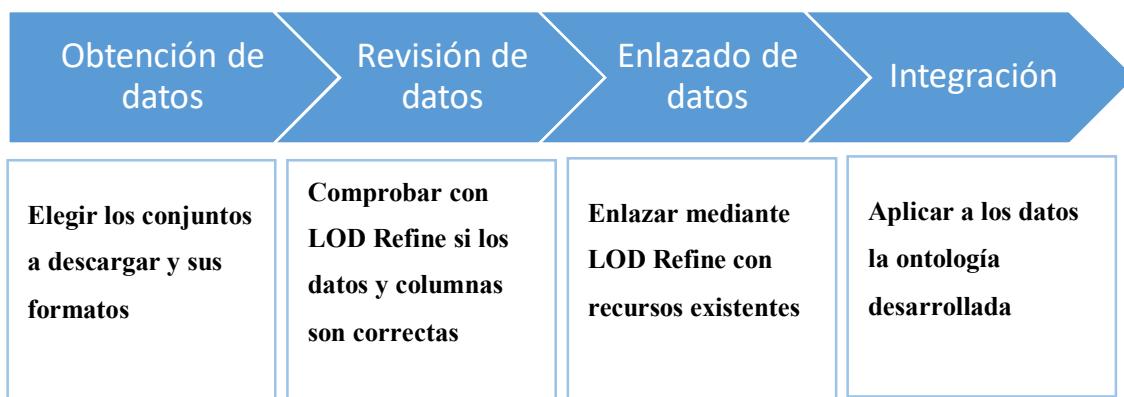


Ilustración 66 - Proceso de generación de ficheros RDF

Un fragmento del RDF de museos es el siguiente:

```
http://madridonyou.com/resources/a2b7382bb596e34d02d24fb197e653d7 a dbo:Museum ;

    dbo:name "Casa Museo Lope de Vega"^^xsd:string ;

    dbo:address "calle cervantes 11"^^xsd:string ;

    dbp:phoneNumber "914 299 216"^^xsd:string ;

    dbp:url
<http://www.madrid.es/sites/v/index.jsp?vgnextchannel=9e4c43db40317010VgnVCM100000dc0ca8c0RCRD&vgnextoid=4eff8c46145e8110VgnVCM2000000c205a0aRCRD> ;

    geo:lat "40.414358466555235"^^xsd:float ;

    geo:long "-3.6974741545860015"^^xsd:float ;

    rdfs:comment "Para la visita es imprescindible reserva anticipada en: casamuseolopedevega@madrid.org o en el telefono: 914 299 216, en horario del museo. De martes a domingo de 10 a 18 horas. Cerrado los lunes, dias 1 y 6 de enero, 1 de mayo, 9 de noviembre y 24, 25 y 31 de diciembre. Este horario puede sufrir alteraciones por actividades extraordinarias que se celebran en este espacio."^^xsd:string .

<http://madridonyou.com/resources/b02b3bdbf5ff7f440752ce88c7e420ca> a
dbo:PopulatedPlace ;

    dbo:name "CENTRO"^^xsd:string ;
```

Capítulo 6

Casos de uso

En este capítulo se muestran los casos de uso que pueden darse en la aplicación, que están íntimamente relacionados con los requerimientos indicados en la ERS del presente trabajo.

6.1. Actores

Los actores serán los usuarios de las operaciones de login, registro, actualización del perfil y generación de rutas, a través de PCs, tabletas o móviles.

6.2. Casos de uso

De acuerdo con lo establecido en los requisitos funcionales de la ERS, se han definido cuatro casos de uso que pueden darse en la ejecución de la aplicación. Aquellos requisitos que son no funcionales quedan fuera de esta clasificación. Los casos de uso que se han definido son:

- **Login en la aplicación**
- **Registro en la aplicación**
- **Actualización de los datos del perfil**
- **Generación de rutas**

Para cada caso de uso se detallarán las precondiciones, postcondiciones, actor o actores involucrados, la descripción del caso de uso y el diagrama de secuencia asociado al flujo de las operaciones que se realicen.

6.2.1. Caso de uso #1: Login en la aplicación

Precondiciones	El usuario tiene acceso al sistema y no está autenticado.
Postcondiciones	El usuario accede a la página principal de la aplicación.
Actor/es involucrado/s	Usuario a través de PC, móvil o Tablet (front)
Descripción	El usuario utilizará sus credenciales para iniciar sesión en la aplicación. Si los datos que introduce son correctos, será redirigido a la página principal de la aplicación. En caso contrario, verá un mensaje de error que le indicará qué ha ido mal.

Tabla 7 - Caso de uso #1

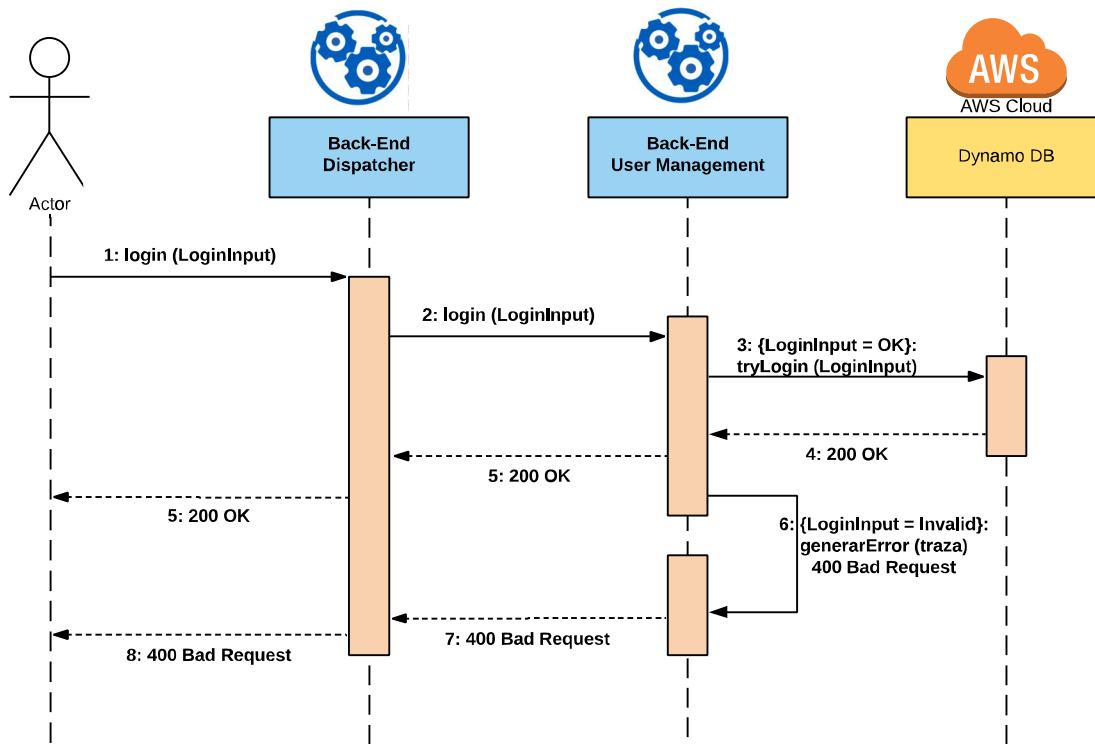


Ilustración 67 - Diagrama de secuencia del caso de uso #1

6.2.2. Caso de uso #2: Registro en la aplicación

Precondiciones	El usuario tiene acceso al sistema y no está autenticado.
Postcondiciones	El usuario accede a la página principal de la aplicación.
Actor/es involucrado/s	Usuario a través de PC, móvil o Tablet (front)
Descripción	El usuario no dispone de credenciales, por lo que tiene que registrarse en la aplicación mediante un formulario de registro. Una vez haya introducido datos aparentemente correctos, se intentará registrarle en la aplicación y no hay errores será redirigido a la página principal de la aplicación. En caso contrario verá un mensaje de error que le indicará qué ha ido mal.

Tabla 8 - Caso de uso #2

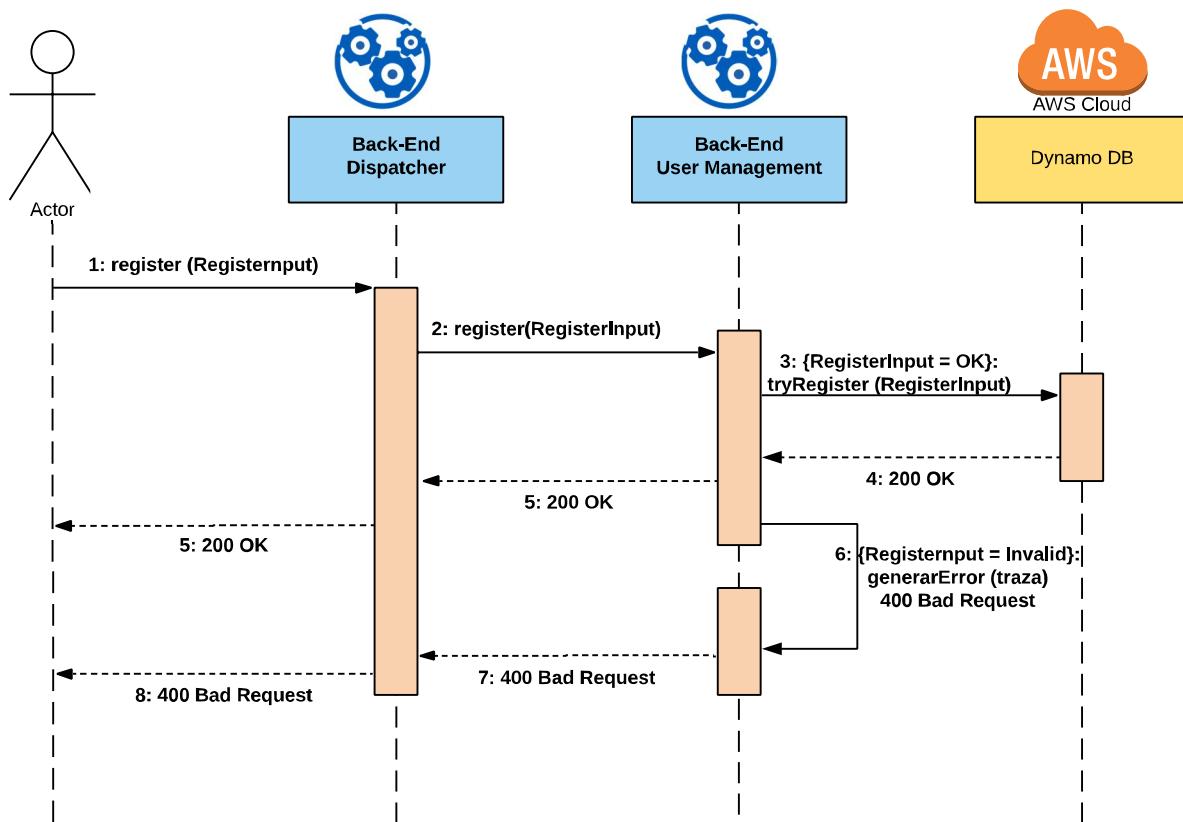


Ilustración 68 - Diagrama de secuencia del caso de uso #2

6.2.3. Caso de uso #3: Actualización del perfil

Precondiciones	El usuario tiene acceso al sistema y está autenticado.
Postcondiciones	El usuario ha modificado sus datos.
Actor/es involucrado/s	Usuario a través de PC, móvil o Tablet (front)
Descripción	El usuario autenticado rellenará los datos del formulario de actualización del perfil y, si todo es correcto, los datos serán actualizados. En caso contrario, se mostrará un mensaje de error indicando que los datos no se pudieron actualizar.

Tabla 9 - Caso de uso #3

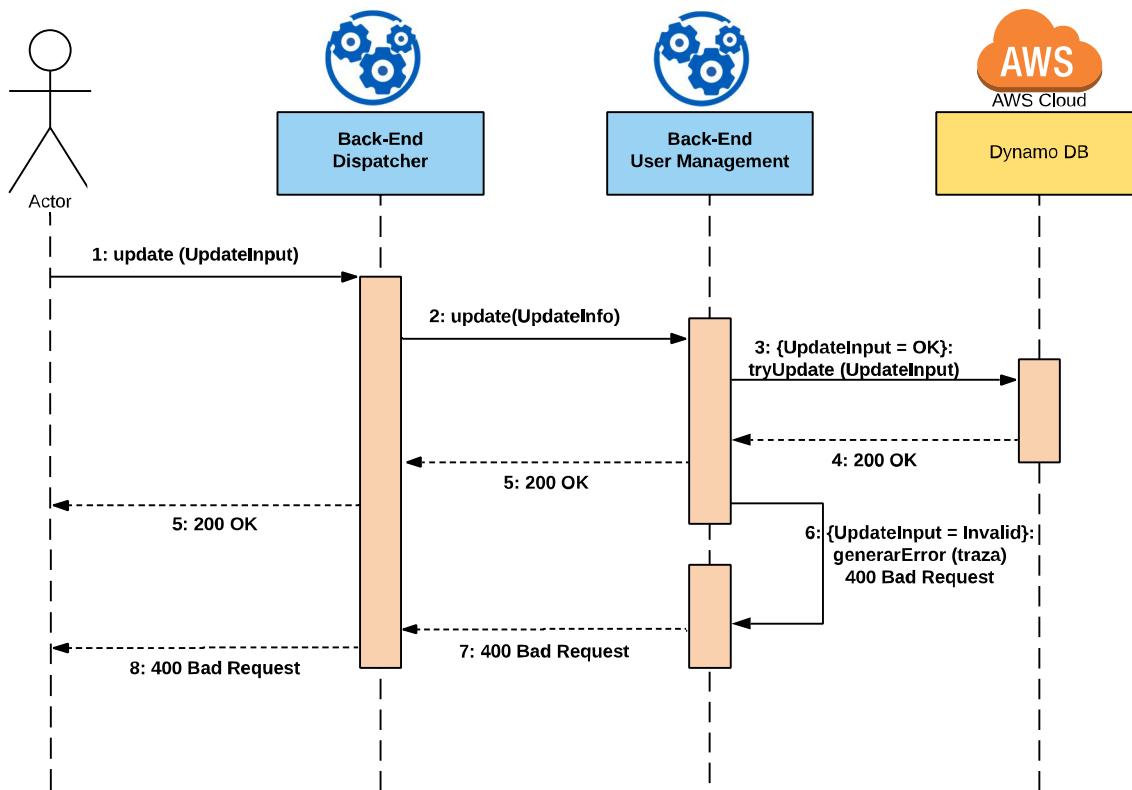


Ilustración 69 - Diagrama de secuencia del caso de uso #3

6.2.4. Caso de uso #4: Generación de rutas

Precondiciones	El usuario tiene acceso al sistema y está autenticado.
Postcondiciones	El usuario recibe una ruta con varios elementos.
Actor/es involucrado/s	Usuario a través de PC, móvil o Tablet (front)
Descripción	El usuario autenticado llenará un formulario que se enviará al back-end, donde se procesará realizando queries y llamadas a la API de OSM. Una vez se haya compuesto la ruta, se enviará al front para que la pinte sobre el mapa.

Tabla 10 - Caso de uso #4

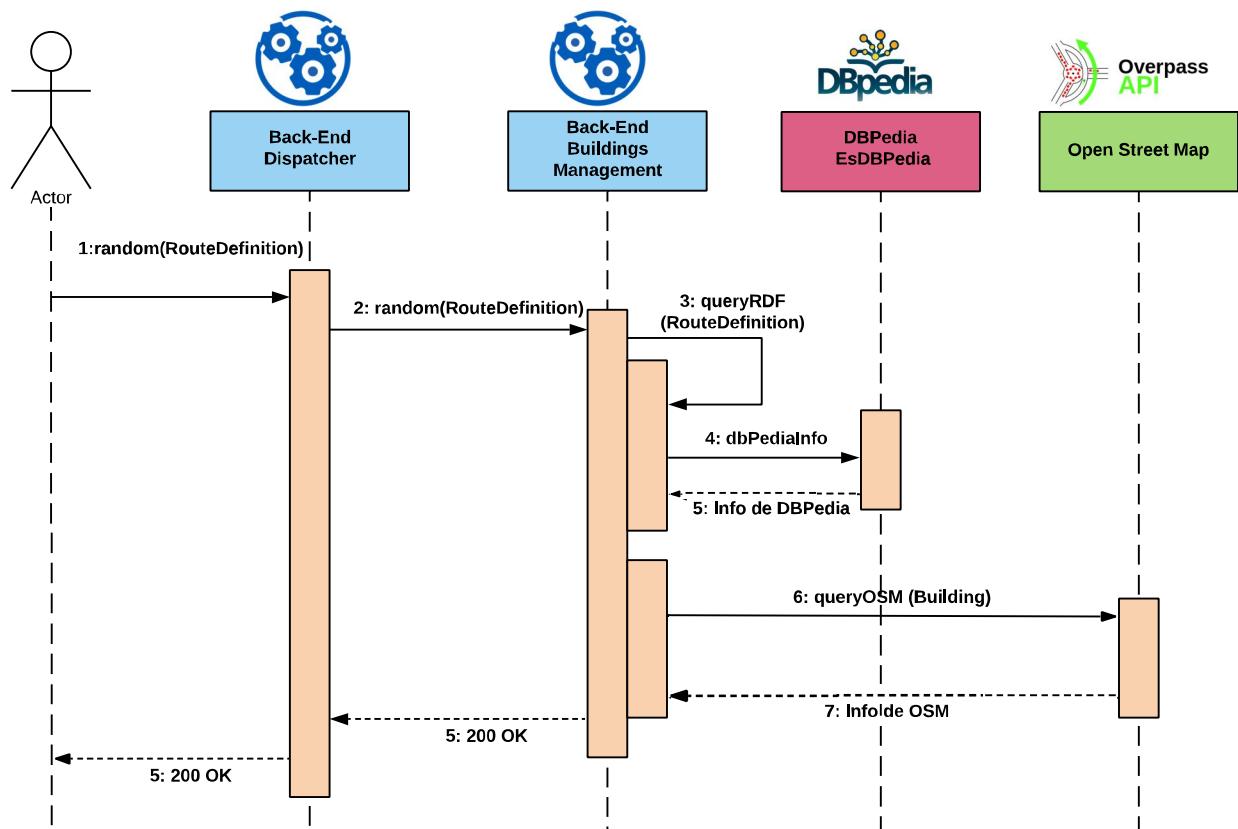


Ilustración 70 - Diagrama de secuencia del caso de uso #4

Capítulo 7

Evaluación del sistema

En este capítulo se muestran las pruebas que a las que se ha sometido el sistema para verificar su funcionamiento y asegurar la calidad del mismo. Para ello, se han realizado pruebas unitarias y pruebas funcionales, de forma que toda la funcionalidad de la aplicación quede cubierta.

7.1. Pruebas unitarias

Para las pruebas unitarias se ha utilizado la herramienta *JUnit* al ser la más apropiada dado que el código fuente de la aplicación está escrito en Java. Se han desarrollado pruebas unitarias sobre los diferentes partes del sistema:

- **Front:** Pruebas que aseguran que el controlador que genera las vistas funciona correctamente.
- **Back – User Management:** Pruebas que aseguran que las funcionalidades de alta, registro y actualización de usuario funcionan correctamente cuando los datos de entrada son correctos y que devuelven los mensajes de error oportunos cuando los datos no son correctos.
- **Back – Buildings Management:** Pruebas que aseguran que la funcionalidad de generar una ruta funciona correctamente para cualquier tipo de entrada que llegue al sistema.
- **Back – Mail Engine:** Pruebas que verifican el envío correcto de e-mails.
- **Back – Dispatcher:** Pruebas que aseguran que la conectividad entre el front y el back es correcta.

Todas las pruebas desarrolladas se encuentran dentro de la carpeta *src/test/java* de los distintos microservicios y deben ejecutarse satisfactoriamente para poder crear una versión dirigida al entorno de producción.

7.2. Pruebas funcionales

Con el objetivo de asegurar la funcionalidad completa del proyecto, más allá del correcto funcionamiento individual de cada una de las piezas que lo componen, se ha realizado un plan de pruebas funcionales, el cual se muestra a continuación, que recoge todas las funcionalidades de la aplicación y evalúa si se llevan a cabo satisfactoriamente o no las acciones que realiza el usuario.

Nótese que todas las pruebas funcionales se han realizado desde la interfaz de usuario construida.

7.2.1. Pruebas funcionales de gestión de usuarios

En esta sección se detallan las pruebas funcionales relativas a la gestión de usuarios del sistema: creación, modificación y login de usuarios. Para cada prueba se muestra su nombre, modulo o microservicio asociado, requerimientos asociados, descripción, datos de entrada, condición/es de éxito y resultado final de la prueba.

Prueba 01	
Nombre de la prueba	TFunc_UserMng_01
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	INI-001: El mecanismo de login debe fundamentarse al menos en un nombre de usuario y una contraseña. INI-002: Si el login es correcto, se redirigirá al usuario a la página principal de la aplicación.
Descripción	El usuario intenta iniciar sesión en la aplicación
Entrada	Datos de usuario correctos
Condición de éxito	El usuario es redirigido a la página <i>home</i>
Resultado	OK

Tabla 11 - Prueba funcional 01

Prueba 02	
Nombre de la prueba	TFunc_UserMng_02
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	<p>INI-001: El mecanismo de login debe fundamentarse al menos en un nombre de usuario y una contraseña.</p> <p>INI-003: Si el login es incorrecto, se mostrará un mensaje de error (en texto rojo) indicando que alguno de los parámetros era incorrecto (no se debe especificar cuál).</p>
Descripción	El usuario intenta iniciar sesión en la aplicación
Entrada	Datos de usuario incorrectos
Condición de éxito	El usuario recibe un mensaje de error
Resultado	OK

Tabla 12 - Prueba funcional 02

Prueba 03	
Nombre de la prueba	TFunc_UserMng_03
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	<p>REG-001: El formulario de registro debe contener los campos mínimos para recopilar información útil del usuario, es decir, nombre de usuario, contraseña y e-mail.</p> <p>REG-002: La contraseña deberá tener al menos 3 caracteres.</p> <p>REG-003: El nombre de usuario elegido no puede ser idéntico a otro ya existente.</p> <p>REG-004: Ningún campo del registro puede quedar vacío.</p> <p>REG-005: Si el registro es correcto, se redirigirá al usuario a la página principal de la aplicación.</p> <p>INI-004: Las contraseñas deben almacenarse en una base de datos habiendo sido tratadas previamente por una función hash.</p>
Descripción	El usuario intenta registrarse en la aplicación
Entrada	Datos de registro correctos
Condición de éxito	El usuario es redirigido a la página <i>home</i> con su nueva cuenta
Resultado	OK

Tabla 13 - Prueba funcional 03

Prueba 04	
Nombre de la prueba	TFunc_UserMng_04
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	<p>REG-001: El formulario de registro debe contener los campos mínimos para recopilar información útil del usuario, es decir, nombre de usuario, contraseña y e-mail.</p> <p>REG-002: La contraseña deberá tener al menos 3 caracteres.</p> <p>REG-003: El nombre de usuario elegido no puede ser idéntico a otro ya existente.</p> <p>REG-004: Ningún campo del registro puede quedar vacío.</p> <p>REG-006: Si el registro es incorrecto, se mostrará un mensaje de error en texto rojo indicando que los parámetros eran incorrectos (sin especificar cuál).</p> <p>INI-004: Las contraseñas deben almacenarse en una base de datos habiendo sido tratadas previamente por una función hash.</p>
Descripción	El usuario intenta registrarse en la aplicación
Entrada	Datos de registro incorrectos
Condición de éxito	El usuario recibe un mensaje de error
Resultado	OK

Tabla 14 - Prueba funcional 04

Prueba 05	
Nombre de la prueba	TFunc_UserMng_05
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	UPD-001: Si los datos indicados son correctos, el usuario verá sus datos actualizados.
Descripción	El usuario intenta actualizar su perfil
Entrada	Datos de perfil correctos
Condición de éxito	El usuario ve actualizados sus datos
Resultado	OK

Tabla 15 - Prueba funcional 05

Prueba 06	
Nombre de la prueba	TFunc_UserMng_06
Modulo / Microservicio	Front-End, Back-End (Micro User Management), SaaS
Requisitos asociados	UPD-002: Si los datos indicados son erróneos, el usuario verá un mensaje de error y los datos no se modificarán, prevaleciendo así los antiguos.
Descripción	El usuario intenta actualizar su perfil
Entrada	Datos de perfil incorrectos
Condición de éxito	El usuario recibe un mensaje de error y los datos del perfil no se modifican
Resultado	OK

Tabla 16 - Prueba funcional 06

7.2.2. Pruebas funcionales de generación de rutas

En esta sección se detallan las pruebas funcionales relativas a la generación de rutas del sistema. Desde su creación, en función de los parámetros que indique el usuario, hasta su visualización final en el mapa, con todos los elementos adicionales en los alrededores de los elementos de la ruta.

Prueba 07	
Nombre de la prueba	TFunc_RouteGen_01
Modulo / Microservicio	Front-End, Back-End (Micro Buildings Management), DBpedia, esDBpedia, OSM
Requisitos asociados	<p>GEN-001: El usuario deberá poder elegir si quiere que el recorrido sea por todo Madrid o únicamente por un distrito.</p> <p>GEN-002: El usuario deberá poder elegir las horas entre las que se va a realizar el recorrido.</p> <p>GEN-003: El usuario deberá poder elegir el número de elementos que contendrá el recorrido.</p> <p>GEN-004: La ruta resultante se formará sobre un mapa con cada elemento perteneciente señalado en él.</p>
Descripción	El usuario intenta crear una ruta por todo Madrid, de 3 elementos, con un horario a su elección
Entrada	Datos para la generación de la ruta
Condición de éxito	El usuario ve un mapa con 3 localizaciones señaladas en él, que pueden pertenecer a cualquier distrito de Madrid.
Resultado	OK

Tabla 17 - Prueba funcional 07

Prueba 08	
Nombre de la prueba	TFunc_RouteGen_02
Modulo / Microservicio	Front-End, Back-End (Micro Buildings Management), DBpedia, esDBpedia, OSM
Requisitos asociados	<p>GEN-001: El usuario deberá poder elegir si quiere que el recorrido sea por todo Madrid o únicamente por un distrito.</p> <p>GEN-002: El usuario deberá poder elegir las horas entre las que se va a realizar el recorrido.</p> <p>GEN-003: El usuario deberá poder elegir el número de elementos que contendrá el recorrido.</p> <p>GEN-004: La ruta resultante se formará sobre un mapa con cada elemento perteneciente señalado en él.</p>
Descripción	El usuario intenta crear una ruta sin personalizar ninguna opción (distrito, horas y número de elementos)
Entrada	Datos para la generación de la ruta
Condición de éxito	El usuario ve un mapa con 5 localizaciones señaladas en él, que pueden pertenecer a cualquier distrito de Madrid.
Resultado	OK

Tabla 18 - Prueba funcional 08

Prueba 09	
Nombre de la prueba	TFunc_RouteGen_03
Modulo / Microservicio	Front-End, Back-End (Micro Buildings Management), DBpedia, esDBpedia, OSM
Requisitos asociados	<p>GEN-001: El usuario deberá poder elegir si quiere que el recorrido sea por todo Madrid o únicamente por un distrito.</p> <p>GEN-002: El usuario deberá poder elegir las horas entre las que se va a realizar el recorrido.</p> <p>GEN-003: El usuario deberá poder elegir el número de elementos que contendrá el recorrido.</p> <p>GEN-004: La ruta resultante se formará sobre un mapa con cada elemento perteneciente señalado en él.</p>
Descripción	El usuario intenta crear una ruta por el distrito CENTRO de Madrid, con 2 elementos
Entrada	Datos para la generación de la ruta
Condición de éxito	El usuario ve un mapa con 2 localizaciones señaladas en él, que pertenecen al distrito CENTRO de Madrid.
Resultado	OK

Tabla 19 - Prueba funcional 09

Prueba 10	
Nombre de la prueba	TFunc_RouteVis_01
Modulo / Microservicio	Front-End, Back-End (Micro Buildings Management), DBpedia, esDBpedia, OSM
Requisitos asociados	<p>VIS-001: Junto con cada edificio perteneciente a la ruta generada, se deberá mostrar una breve descripción del mismo, acompañada de datos útiles para el usuario.</p> <p>VIS-002: Junto con cada edificio se mostrará, en caso de ser posible, una imagen que se ajuste a la actualidad del edificio.</p> <p>VIS-003: Junto a cada elemento de la ruta generada, aparecerán diversas localizaciones de puntos de interés como pueden ser cafeterías, restaurantes, teatros, cines o información sobre parkings o paradas de bus y taxi.</p> <p>VIS-004: Los puntos de interés adicionales deberán asociarse a cada uno de los elementos de la ruta.</p> <p>VIS-005: Por cada punto de interés adicional se deberá indicar algún tipo de información útil, junto con la distancia aérea, y se visualizarán en el mapa.</p>
Descripción	El usuario intenta visualizar información relativa a los elementos de la ruta.
Entrada	Ruta generada en la Prueba 09
Condición de éxito	El usuario verá descrito cada elemento de la ruta, con toda la información disponible que se tenga de ese elemento, junto con localizaciones cercanas a él, que podrá mostrar u ocultar. Para cada elemento cercano, verá su nombre, dirección y la distancia aérea a la que se encuentra del elemento de la ruta.
Resultado	OK

Tabla 20 - Prueba funcional 10

Prueba 11	
Nombre de la prueba	TFunc_RouteVis_02
Modulo / Microservicio	Front-End, Back-End (Micro Buildings Management), DBpedia, esDBpedia, OSM
Requisitos asociados	<p>VIS-006: Todos los elementos que se muestren en el mapa deben poder mostrarse y ocultarse.</p> <p>VIS-007: El mapa resultante debe tener al menos dos tipos de vistas distintas.</p> <p>VIS-008: Se deberá poder hacer zoom sobre el mapa.</p> <p>VIS-009: Se deberá poder enviar por correo electrónico el detalle de la ruta generada.</p>
Descripción	El usuario intenta visualizar información el mapa y recibir un correo electrónico con el detalle de cómo se ha generado la ruta.
Entrada	Ruta generada en la Prueba 09
Condición de éxito	El usuario verá en el mapa los distintos elementos de la ruta generada. Podrá aumentar o disminuir el zoom, así como cambiar la vista del mapa y seleccionar qué elementos quiere que aparezcan en él. Podrá ocultar o mostrar los distintos tipos de elementos que se muestran en el mapa. Además, en su correo electrónico, el usuario recibe un e-mail con el detalle de generación de la ruta.
Resultado	OK

Tabla 21 - Prueba funcional 11

Capítulo 8

Usabilidad

Sin duda hoy en día es fundamental que las aplicaciones y los sistemas que se construyen cuenten con un nivel de usabilidad adecuado y sean sencillos de utilizar. Además, deben tratar de evitar que los usuarios se sientan descontentos al utilizarlos (ya sea por que vayan lentos, porque los colores hagan daño a los ojos, porque los sonidos sean estridentes...).

Cada vez más, la usabilidad y la experiencia de usuario son dos puntos fundamentales en el desarrollo de software, y por ello he querido incluirlos en la realización de mi trabajo fin de grado.

Para evaluar la usabilidad de la aplicación construida se han elaborado una serie de sencillos tests que se repartirán a varios usuarios. En esos tests se informa al usuario de que va a realizar una serie de tareas que permitirán evaluar la usabilidad de la aplicación construida. Se le proporcionan las instrucciones pertinentes para el uso del mismo, y acto seguido se evalúa su interacción con la aplicación.

Un total de 4 personas han sido evaluadas. A continuación se muestra la plantilla que se entregó a los usuarios y los resultados que se obtuvieron en cada una de las evaluaciones.

Madrid on You

Lea con atención

Este pequeño test permite comprobar el grado de usabilidad del producto construido. El producto es una aplicación que permite generar rutas aleatorias por la ciudad de Madrid, mostrando localizaciones de interés ocio-turístico y cultural.

A continuación se enuncian varias tareas que deberá realizar utilizando la aplicación. En caso de duda, no dude en preguntar a la persona que le está supervisando.

Tareas a realizar

Tarea 1: Regístrese en la aplicación. Puede elegir el nombre de usuario y contraseña que desee.

Tarea 2: Una vez registrado, y con la sesión iniciada, verá la pantalla principal de la aplicación. Diríjase a su perfil e intente modificar su contraseña, indicando una nueva.

Tarea 3: Intente generar una ruta aleatoria para el próximo fin de semana, por el distrito centro de Madrid. La ruta debe contener 3 elementos. Puede elegir las horas que desee.

Tarea 4: Intente visualizar los datos asociados a cada elemento de la ruta generada (nombre, dirección...).

Tarea 5: Intente ver, por cada uno de los elementos de la ruta, los puntos de interés cercanos (bares, restaurantes, cines...).

Tarea 6: Vaya al mapa donde se muestra la ruta generada. Intente mostrar en el mapa únicamente los cines (en caso de que haya alguno). Si no hay, intente mostrar únicamente los restaurantes.

8.1. Resultados de la evaluación

Para cada participante se llenó, con ayuda de un compañero, la siguiente tabla de resultados:

Participante nº		Número de participante
Edad		Edad del participante
Tarea	Éxito	Comentarios
1	SI/NO	Comentarios específicos del usuario en la tarea
2	SI/NO	Comentarios específicos del usuario en la tarea
3	SI/NO	Comentarios específicos del usuario en la tarea
4	SI/NO	Comentarios específicos del usuario en la tarea
5	SI/NO	Comentarios específicos del usuario en la tarea
6	SI/NO	Comentarios específicos del usuario en la tarea
Observaciones		Observaciones generales de la prueba (rellena el supervisor)

Tabla 22 - Formulario de resultados de usabilidad

A continuación se muestran los resultados obtenidos por las cuatro personas que realizaron el test.

Participante nº	1	
Edad	24	
Tarea	Éxito	Comentarios
1	SI	“He metido una contraseña de 1 letra porque no sabía que el mínimo eran 3. Podría haberlo sabido antes de darle a enviar.”
2	SI	“Ahora ya sabía que la contraseña tenía que tener mínimo 3 letras”
3	SI	
4	SI	
5	SI	
6	SI	“Quizá hay demasiadas cosas en el bocadillo que se despliega”
Observaciones		El usuario ha sido capaz de realizar todas las tareas. No le ha gustado que el sistema no le avisase (previo envío de formulario) de que el tamaño mínimo de la contraseña son 3 dígitos.

Tabla 23 - Resultados del primer usuario

Participante nº	2	
Edad	38	
Tarea	Éxito	Comentarios
1	SI	
2	SI	
3	SI	
4	SI	
5	SI	“Estaría bien que cuando pico en un botón para ver los restaurantes por ejemplo, el botón ese se pusiera de otro color, para saber que lo tengo pinchado”
6	SI	
Observaciones		El usuario ha sido capaz de realizar todas las tareas. A la hora de ver los diferentes puntos de interés asociados a cada elemento de la ruta, ha visto que, efectivamente, los botones no cambian de color cuando se hace click sobre ellos. Algo que sin duda debe corregirse en futuras versiones.

Tabla 24 - Resultados del segundo usuario

Participante nº	3	
Edad	19	
Tarea	Éxito	Comentarios
1	SI	
2	SI	
3	SI	
4	SI	
5	SI	“Me gusta que estén ordenados de menor a mayor distancia”
6	SI	
Observaciones		El usuario ha sido capaz de realizar todas las tareas. Sorprendentemente no ha comentado ningún aspecto a mejorar. Ha sido el usuario que más rápido ha realizado el test. Esto puede ser debido a que también es el más joven y está acostumbrado a tratar con aplicaciones a diario.

Tabla 25 - Resultados del tercer usuario

Participante nº		4
Edad		47
Tarea	Éxito	Comentarios
1	SI	“No me gusta que el cuadrito haya temblado cuando he metido el mail mal. Me pone nerviosa.”
2	SI	“Estaría bien que se pudiera modificar la imagen. No me gusta la que viene de primeras.”
3	SI	
4	SI	
5	SI	
6	SI	“No me gusta que los dibujitos se solapen cuando hay muchos...”
Observaciones		El usuario ha sido capaz de realizar todas las tareas. A la hora de registrarse ha introducido un e-mail erróneo (sin el @) y la aplicación lo ha rechazado. El diálogo modal que muestra el registro ha temblado (gracias a un <i>shake</i> programado en JS) y eso ha provocado que el usuario se pusiese nervioso. Quizá se podría relajar dicha animación para que no sea tan brusca. Además, el usuario ha indicado que se debería poder modificar la imagen del perfil. En efecto, esta es una funcionalidad que se implementará en versiones sucesivas, ya que no estaba planificada en esta versión. Por último, ha dicho que los iconos relativos a restaurantes se solapaban unos con otros cuando había varios en pocos metros. Efectivamente esto es así, y deberá mejorarse para versiones sucesivas.

Tabla 26 - Resultados del cuarto usuario

Gracias a los resultados de este test de usabilidad será posible planificar una nueva versión de la aplicación en un futuro, que corrija todos los defectos e imperfecciones que esta contiene, en la opinión de los usuarios evaluados.

Es importante destacar que, a pesar de las observaciones que algunos usuarios dan, todos fueron capaces de realizar las tareas propuestas sin necesitar ayuda (cuando cometían un error les bastaba con el mensaje que la aplicación les mostraba para saber qué estaba mal) lo que implica que el sistema cumple unos mínimos exigibles de usabilidad y está listo para ser usado y puesto en producción.

Capítulo 9

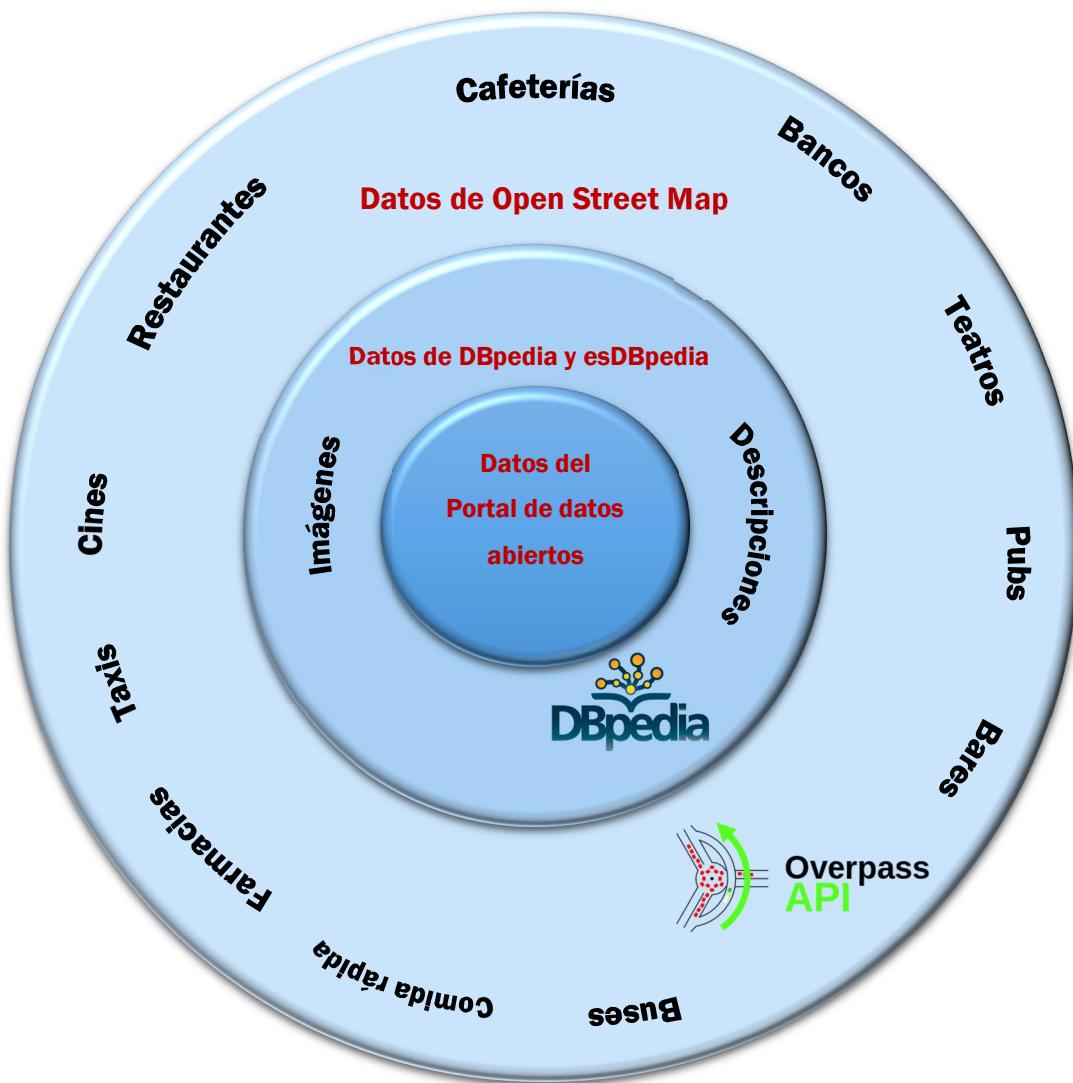
Conclusiones

En este trabajo se ha creado una aplicación **totalmente original** en la cual se toma como base principal de información unos ficheros muy pequeños (el mayor tiene un tamaño de 121 KB) y a la vez devuelve una enorme cantidad de información. **La clave es hacer uso de los datos enlazados.** Gracias a ellos, la aplicación es capaz de absorber una gran cantidad de información, que se muestra al usuario en forma de ruta turística para que en cuestión de segundos tenga planificado un recorrido ocio-cultural por Madrid. Tomando los objetivos planteados en el primer capítulo, se obtienen las siguientes conclusiones:

- Se ha desarrollado un **software que muestra rutas aleatorias a los usuarios** por la ciudad de Madrid, que contienen lugares de interés turístico y de ocio.
- Las rutas se pueden visualizar en un **mapa altamente personalizable**.
- Gracias a los test de usabilidad se puede concluir que **la interfaz de usuario es mejorable pero consigue alcanzar el nivel mínimo exigible y permite que los usuarios puedan interactuar fácilmente** con la aplicación. Además, se ha implementado un diseño responsive.
- **La aplicación podrá ser usada por alumnos** que estén aprendiendo a trabajar con las tecnologías involucradas en este proyecto gracias al envío del PDF con los pasos de generación de la ruta.

Para llevar a cabo el desarrollo de software se ha seguido una metodología de mejora continua llamada PDCA (*Plan-Do-Check-Act o Ciclo de Deming*). A la hora de desarrollar ha sido fundamental contar con una ERS bien definida, así como con los diagramas UML y casos de uso asociados a los diseños de bajo y alto nivel, respectivamente. Sin duda la parte más difícil del desarrollo ha sido la integración con OSM y la correcta visualización de los mapas, debido a mi poca experiencia en front-end, así como realizar la ontología y las queries SPARQL de manera adecuada.

Uno de los objetivos clave de este trabajo era **enriquecer la información** de los conjuntos de datos que se obtenían del portal de datos abiertos de Madrid. Dicha información se ha enriquecido integrándola con datos obtenidos de DBpedia y esDBpedia, así como con datos de Open Street Map. El resultado es una aplicación que, teniendo como base muy poca información, ofrece al usuario una gran cantidad de alternativas para realizar planes por Madrid. Todo ello gracias a los principios de los datos enlazados. La siguiente y última ilustración muestra, en forma de composición, cómo las distintas fuentes de datos contribuyen al enriquecimiento de la información original.



9.1. Concurso de datos abiertos de Madrid

Esta aplicación fue presentada en el **Concurso de Datos Abiertos de la Ciudad de Madrid**²¹, que tuvo lugar el último trimestre del año 2017, resultando candidata a obtener uno de los principales premios de la categoría de *Datos Abiertos*. Desgraciadamente, debido al desfase temporal entre la entrega del concurso (finales de noviembre) y la entrega del presente trabajo (15 de enero) no fue posible presentar una versión totalmente funcional del sistema al concurso.



Ilustración 72 - Presentación de la app en Medialab Prado (Madrid)

9.2 Aplicación a la docencia

Con la intención de ayudar a futuros compañeros he querido que dentro de este trabajo hubiera una parte, aunque pequeña, que pudiese ser utilizada para facilitar el aprendizaje en un área que, bajo mi punto de vista, no es nada fácil. Una vez se ha generado una ruta, existe la opción de enviar las instrucciones de generación de la ruta por correo electrónico (al e-mail que el usuario indicó en el registro), recibiendo así un correo electrónico con los distintos pasos que se han llevado a cabo para generar la ruta. Únicamente se mencionan aquellos pasos relacionados con los ficheros RDF, SPARQL, ontologías y Open Street Map.

La intención de esto es que los alumnos de asignaturas relacionadas con web semántica (preferentemente alumnos de la asignatura *Web Semántica y Linked Data /Web of Linked Data And Semantic Web*) tengan un modelo de referencia del poder que pueden tener los datos enlazados y cómo pueden utilizarse, así como ver consultas SPARQL sobre RDF y consultas federadas en directo. Todo ello para facilitar el aprendizaje de los alumnos, intentando que comprendan desde un primer momento los alcances de estos procedimientos.

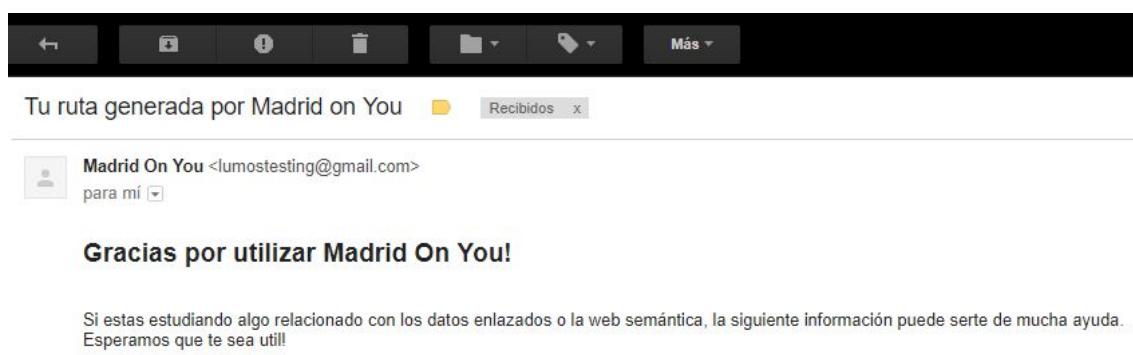


Ilustración 73 - Fragmento del mail enviado a alumnos

Capítulo 12

Líneas futuras

La aplicación que se ha construido en el presente Trabajo Fin de Grado es una primera versión (1.0) y es por ello que, a pesar de ser totalmente funcional y cubrir los objetivos planteados, es fácil identificar puntos a mejorar para futuras versiones. A continuación se indica, para cada uno de los módulos a mejorar, aspectos a tener en cuenta para futuras revisiones.

12.1. Front-End

Las mejoras que se pueden aplicar en el front-end de la aplicación vienen directamente de los resultados del análisis de usabilidad. Los usuarios han identificado ciertos aspectos que podrían mejorarse:

- Se debería mostrar, antes de llenar ningún formulario, los valores válidos para los campos (nombre de usuario, contraseña, e-mail...).
- El desplegable del mapa donde se ve la ruta podría estar mejor organizado.
- Los botones que muestran y ocultan puntos de interés cercanos a un elemento de la ruta deberían cambiar de color cuando están pulsados.
- Los iconos que se muestran en el mapa para restaurantes, bares, pubs... deberían estar mejor diseñados (con sombras y 3D) para que al solaparse en el mapa no den mala impresión.

12.2. Back-End

Las mejoras que se pueden aplicar en el back-end están relacionadas con funcionalidades no implementadas o incompletas:

- Se debería implementar la funcionalidad de cambiar la imagen en el perfil del usuario.
- A la hora de generar las rutas, se podría implementar una “inteligencia” que seleccione elementos aleatorios pero que no estén separados a más de X distancia, ya que podría darse el caso, si elegimos una ruta por todo Madrid, de que los elementos de la ruta estén situados en puntos muy distantes unos de otros, haciendo de este modo difícil (o imposible) que la ruta se complete en un día.
- Si bien se ha obtenido mucha información de fuentes externas como DBpedia u OSM, sería posible extraer aún muchísima más, para que el sistema de una información mucho más completa.
- Se podría implementar una funcionalidad que dibuje en el mapa el recorrido para ir de un elemento a otro de la ruta (como las indicaciones que da un GPS).
- Se podría implementar una funcionalidad que permita obtener la distancia de un elemento a un punto de interés asociado mediante indicaciones GPS y no mediante la distancia aérea, ya que en algunos casos la distancia aérea pasa por atravesar edificios y se muestra una distancia inferior a la real.
- Se debería idear una forma de reducir el impacto en el tiempo de ejecución debido al límite de 1 segundo por petición de la API Overpass. Esto hace que, siendo la aplicación muy rápida en sus operaciones internas, se vea ralentizada por el limitado rendimiento de la API asociada a Open Street Map.
- El mail que se envía a los usuarios podría ser más elegante. Además, se podría usar el microservicio de mail para confirmaciones de registros o actualizaciones del perfil.

12.3. Ontología y RDF

Las mejoras que se pueden aplicar en la ontología y los ficheros RDF tienen que ver con ofrecer aún más alternativas a los usuarios para poder generar rutas más variadas.

- Incluir en la ontología más elementos de los que hay ya, para poder realizar rutas con un número mayor de alternativas.
- Publicar los ficheros RDF en un repositorio abierto para que sean accesibles por alguien que quiera utilizarlos.
- Generar consultas SPARQL para obtener recursos en base al estilo arquitectónico o el año de construcción de los elementos de las rutas, permitiendo así la creación de rutas por edificios barrocos, del siglo XIX, del siglo XX...

Bibliografía

- [1] – “Datos Enlazados.” Wikipedia, Wikimedia Foundation, 29 Sept. 2017, es.wikipedia.org/wiki/Datos_enlazados.
- [2] – “Datos Abiertos.” Wikipedia, Wikimedia Foundation, 14 Jan. 2018, es.wikipedia.org/wiki/Datos_abiertos.
- [3] – “Multimedia y Web 2.0.” Instituto Nacional De Tecnologías Educativas y De Formación Del Profesorado, 14 Dec. 2016, www.ite.educacion.es/formacion/materiales/155/cd/modulo_1_Iniciacionblog/concepto_de_web_20.html.
- [4] – “Tim Berners-Lee.” Wikipedia, Wikimedia Foundation, 14 Jan. 2018, es.wikipedia.org/wiki/Tim_Berners-Lee.
- [5] – Delgado, Hugo. “La Web 3.0 Significado y Origen.” Diseño De Páginas Web, Sitios De Internet y Posicionamiento SEO Akus.net, 5 Sept. 2012, disenowebakus.net/la-web-3.php
- [6] – “The Linking Open Data Cloud Diagram.” The Linking Open Data Cloud Diagram, 22 Aug. 2017, lod-cloud.net/.
- [7] – “Semantic Web.” W3C, 12 Oct. 2016, www.w3.org/standards/semanticweb/.
- [8] – “Resource Description Framework.” Wikipedia, Wikimedia Foundation, 7 Jan. 2018, es.wikipedia.org/wiki/Resource_Description_Framework.
- [9] – W3C Consortium. “SPARQL Query Language.” SPARQL Query Language for RDF, 15 Jan. 2008, www.w3.org/TR/rdf-sparql-query/.
- [10] – Quilitz, Bastian. Leser, Ulf. “Querying distributed RDF Data Sources with SPARQL.” Humboldt-Universität zu Berlin, https://link.springer.com/content/pdf/10.1007/978-3-540-68234-9_39.pdf

- [11] – García Castro, Raúl. Poveda-Villalón, María. Radulovic, Filip. “Methodological guidelines for the generation of Linked Data”. Universidad Politécnica de Madrid.
- [12] – García Castro, Raúl. Radulovic, Filip. “RDF Generation with LOD Refine”. Universidad Politécnica de Madrid.
- [13] – “Ontology (Information Science).” Wikipedia, Wikimedia Foundation, 13 Jan. 2018, [en.wikipedia.org/wiki/Ontology_\(information_science\)](https://en.wikipedia.org/wiki/Ontology_(information_science)).
- [14] – “OpenStreetMap Wiki.” OpenStreetMap Wiki, 21 Nov. 2017, wiki.openstreetmap.org/wiki/Main_Page.
- [15] – “Amazon Web Services.” Wikipedia, Wikimedia Foundation, 14 Jan. 2018, en.wikipedia.org/wiki/Amazon_Web_Services.
- [16] – “Apache Jena -.” Apache Jena, 15 Dec. 2016, jena.apache.org/.
- [17] – “Modelo–Vista–Controlador.” Wikipedia, Wikimedia Foundation, 3 Jan. 2018, es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.
- [18] – “Thymeleaf.” Documentation - Thymeleaf, 17 Apr. 2015, www.thymeleaf.org/documentation.html.
- [19] – Richardson, Chris. “Microservices Pattern: Microservice Architecture Pattern.” Microservices.io, 15 Apr. 2016, microservices.io/patterns/microservices.html.
- [20] – “AJAX.” Wikipedia, Wikimedia Foundation, 9 Jan. 2018, es.wikipedia.org/wiki/AJAX.
- [21] – Medialab Prado. “Premio De Datos Abiertos 2017.” Medialab Prado, 15 Sept. 2017, medialab-prado.es/article/datamad.
- [22] Tom Heath and Christian Bizer. *Linked Data: Evolving the Web into a Global Data Space* (1st edition). Synthesis Lectures on the Semantic Web: Theory and Technology, 1:1, 1-136. Morgan & Claypool. 2011.

ANEXO I: Distribución de código

IMPORTANTE: Todo el código asociado al diseño, desarrollo y depuración de la aplicación realizada estará disponible en **GitHub**, concretamente, en la siguiente dirección:

<https://github.com/JLumos/This-is-Madrid-on-You>

Esto es así ya que el sistema de entrega que ofrece la universidad para archivos distintos a la memoria tiene un tamaño máximo de subida de 10 MB y el desarrollo realizado supera ampliamente ese tamaño (entre imágenes, código, diagramas, etc.)

El hecho de dar a mostrar el código se encuadra mejor dentro de la presentación pública y defensa del presente trabajo que dentro la realización de esta memoria, por lo que todo el código será visible a partir del día **29 de Enero** de 2018.

ANEXO II: Licencia

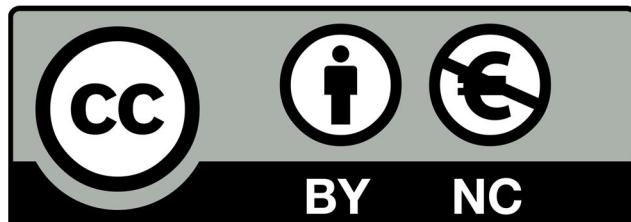
You are free to:

- **Share** — copy and redistribute the material in any medium or format
- **Adapt** — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

Under the following terms:

- **Attribution:** You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
- **Non-commercial:** You may not use the material for commercial purposes.



Este documento esta firmado por

	Firmante	CN=tfgm.fi.upm.es, OU=CCFI, O=Facultad de Informatica - UPM, C=ES
	Fecha/Hora	Tue Jan 16 00:09:05 CET 2018
	Emisor del Certificado	EMAILADDRESS=camanager@fi.upm.es, CN=CA Facultad de Informatica, O=Facultad de Informatica - UPM, C=ES
	Numero de Serie	630
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)