TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Creación de un Portal de Datos Abiertos y Acceso a un Conjunto de Datos mediante API REST

Curso 2019/2020

Alumno/a:

Samuel Rodríguez Simón

Director/es:

Manuel Torres Gil

Antonio Becerra Terón

Creación de un Portal de Datos Abiertos y Acceso a un Conjunto de Datos mediante API REST

Trabajo Fin de Grado

Alumno: Samuel Rodríguez Simón

Director: Manuel Torres Gil

Codirector: Antonio Becerra Terón

Grado en Ingeniería Informática

Escuela Superior de Ingeniería

Universidad de Almería

Agradecimientos

Gracias a mis tutores, Manolo y Antonio, por guiarme a lo largo de este Trabajo Fin de Grado y resolverme dudas cuando ha sido necesario. También quisiera darles las gracias por su labor docente en las asignaturas que me han impartido a lo largo de este grado.

Gracias a todos los profesores que he tenido a lo largo de mi carrera como estudiante. A los maestros del colegio por enseñarme valores y enseñarme conocimientos básicos. A los profesores del instituto, por ser muy buenos transmisores de conocimiento y ayudarme a sentar una sólida base que fue de gran ayuda para mis estudios universitarios. Y a los profesores que he tenido a lo largo de esta carrera universitaria, por formarme de cara a mi futura carrera profesional, y a orientarla dentro del amplio campo que es la informática.

Gracias a mis padres, por cuidarme a lo largo de mis 22 años de vida, enseñarme valores que me han hecho como soy hoy en día y aconsejarme para tomar un camino del que, seguramente, me haga sentir afortunado y orgulloso. Gracias a mi padre por mostrarme que tengo que trabajar para obtener algo, y a mi madre por su espíritu de lucha y enseñarme que debo de formarme adecuadamente para tener un trabajo digno.

Gracias a mi hermana pequeña, por acompañarme en este camino, compartir ciertas aficiones conmigo y una gran cantidad de momentos en los que ha estado a mi lado.

Gracias a mis amigos, por todos los momentos inolvidables en los que me han acompañado, así como el apoyo ofrecido cuando algo no estaba bien. Estoy muy agradecido por vuestra lealtad y comprensión, así como por señalarme ciertos defectos en los momentos que estos salen a la luz.

Índice

Índice de figuras

Índice de tablas

Índice de códigos

1. Introducción

Generalmente, los gobiernos e instituciones públicas, y gracias a los avances tecnológicos, pueden hacer públicos datos institucionales de una gran variedad de temáticas, y de forma que estos sean fácilmente accesibles a toda la ciudadanía. Este fenómeno es el conocido como Open Data. Aparte de ser una muestra importante de transparencia gubernamental, estos datos pueden permitir a los ciudadanos y a las propias instituciones aprovecharlos y tratarlos para darles ciertas aplicaciones que pueden resultar muy interesantes, dar una interpretación más accesible y ser de gran ayuda a otras personas, como se verá en el capítulo correspondiente.

Para hacer públicos estos datos, se hace necesaria una vía de acceso telemática a estos datos. Esta vía de acceso consiste en una página web en la que se publican los distintos conjuntos de datos, llamada portal de datos abiertos. Estos portales, además de ofrecer los conjuntos de datos, los organiza y permite una búsqueda de conjuntos según distintos parámetros, como palabras clave, el formato en el que están los archivos del conjunto o la institución que los publica. Existen varias alternativas en cuanto a software para desplegar un portal de datos abiertos, por lo que también se realizará una comparación entre estas alternativas.

Estos conjuntos de datos pueden estar almacenados en el propio portal de datos abiertos o que enlacen a otra página en la que se encuentra almacenado el conjunto de datos, siendo la segunda opción la que se explorará en este TFG. Se accederá al conjunto de datos desde el portal de datos abiertos usando una API REST para ello, la cual se desarrollará para el conjunto de datos que se usará. El desarrollo de esta API se reflejará en esta memoria en el capítulo correspondiente.

1.1. Objetivos

Los objetivos principales y secundarios definidos para este TFG son los siguientes:

- Conocer el fenómeno del Open Data.
  - Conocer la importancia de los datos abiertos y sus aplicaciones reales.
  - Definir conceptos relacionados con los datos abiertos.
  - Tratar los formatos usados en los datos abiertos.
- Creación de un portal de datos en abierto.
  - Investigar y comparar distintas alternativas de software para la publicación de datos abiertos (portales de datos abiertos).
  - Justificar la alternativa de portal de datos abiertos escogida.
  - Explicar el uso del portal de datos a desarrollar.
  - Instalación y configuración del software de portal Open Data.
  - Creación de conjuntos de datos con enlaces a otros conjuntos de datos abiertos.
  - Creación de un conjunto de datos enlazado al conjunto de datos a crear, haciendo uso de la API REST a desarrollar.
- Desarrollo de una API REST.
  - Definición del conjunto de datos a crear.
  - Introducir la pila MEAN.
  - Creación y configuración del servidor.
  - Implementación de endpoints para el acceso y publicación de datos.
  - Protección de ciertos endpoints usando tokens.
  - Documentación de la API desarrollada.

Esquemáticamente, el proyecto técnico se podría comprender con el siguiente esquema:

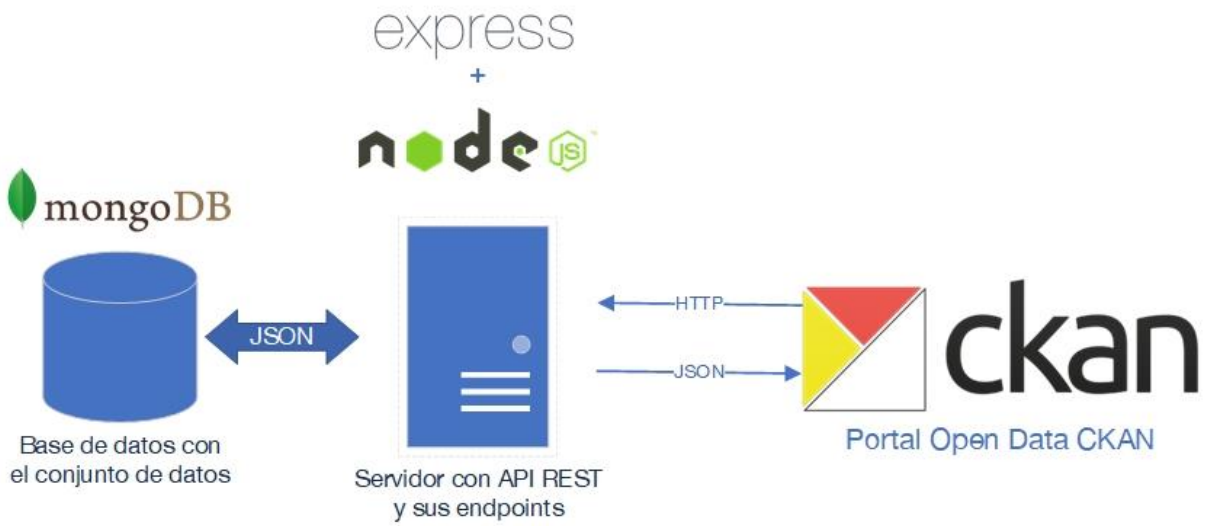

Figura 1.1: Esquema del funcionamiento del proyecto técnico.

1.2. Desarrollo temporal del trabajo por fases

La realización de este trabajo fin de grado se ha realizado en las siguientes fases:

I. Investigación, estudio y realización del anteproyecto.
II. Creación del portal de datos abiertos.
III. Creación de la API REST.
IV. Realización del ejemplo de uso del acceso al dataset desde el portal de datos abiertos.
V. Redacción de la memoria y de la presentación.

En la siguiente figura, se muestra el cronograma de cómo se han acabado desarrollando las fases de este TFG.

MES/SEMANA	Mayo					Junio					Julio				Agosto				Septiembre
FASES	1	2	3	4	5	1	2	3	4	5	1	2	3	4	1	2	3	4	1
I	150 h.																		
II											40 h.								
III																25 h.			
IV																		12 h.	
V															155 h.				

Figura 1.2: Cronograma por fases de la realización de este TFG.

2. Metodologías y Tecnologías Utilizadas. Estado del arte

En este capítulo, introduciremos se introducirá el fenómeno del Open Data, así como las distintas tecnologías disponibles para montar un portal de datos abiertos y crear una API REST para poder interactuar con una base de datos remotas. El estilo de este capítulo será más parecido a un trabajo monográfico que los siguientes episodios en los que se entrará en cuestiones más técnicas y el desarrollo del proyecto.

2.1 Open Data

Esta sección tratará sobre el fenómeno del Open Data, definiendo los conceptos necesarios, mostrando la importancia de los datos abiertos, aplicaciones de estos en la vida real y los formatos digitales en los que estos son publicados.

2.1.1. Concepto de Open Data

Según la Open Knowledge Foundation (OKF), la cual es una de las principales organizaciones en cuanto a datos abiertos, define el término “Open Data” (y sus equivalentes en castellano) como datos que pueden ser utilizados, reutilizados y redistribuidos libremente por cualquier persona, además de encontrarse sujetos a la realización de una atribución al creador y ser compartidos de la misma manera de la cual se han obtenido (Open Knowledge Foundation, 2012).

Gurin (2014) describe el Open Data como: “datos públicos accesibles que la gente, compañías, y organizaciones pueden usar para lanzar nuevos negocios, analizar patrones y tendencias, realizar decisiones motivadas por los datos, y resolver problemas complejos”. Además, establece una diferencia con respecto a otro término, Big Data, el cual, si bien se solapa con el Open Data, define al segundo como “datos con una misión: está diseñado para proveer datos gratuitos, abiertos y transparentes que pueden transformar la manera en la que hacemos negocios, hacemos funcionar el gobierno, y gestionamos todo tipo de transacciones.”, mientras que el Big Data consisten en “procesar conjuntos de datos muy grandes para identificar patrones y conexiones”.

Estos datos abiertos son públicos y están disponibles para todos con fines de uso personal o comercial (Gurin, 2014), publicados de forma estructurada siguiendo cierto método, usando una licencia abierta para garantizar el libre acceso, y que pueden ser conjuntos de datos sobre cualquier administración pública, de investigación científica (en particular, de la financiada públicamente) o de compañías privadas (datos que se pueden publicar con los incentivos y protecciones de privacidad adecuadas) (Monino & Sedkaoui, 2016).

La definición de “datos abiertos” que hemos realizado antes deriva de la definición de “conocimiento abierto”, que es en lo que se convierten los datos abiertos tras su publicación y tratamiento (Open Knowledge Foundation, 2012). Según la página Open Definition, también perteneciente a la OKF (2014), definen el conocimiento abierto de esta manera: “El conocimiento es abierto si cualquiera es libre para acceder a él, usarlo, modificarlo y compartirlo bajo condiciones que, como mucho, preserven su autoría y su apertura”.

2.1.2. Principios del Open Data

De acuerdo con la página opengovdata.org, estos son los ocho principios por los que se deben regir los datos abiertos:

1. **Completos:** Todos los datos públicos se hacen disponibles. Los datos públicos son datos que no están sujetos a privacidad válida, seguridad o limitaciones por privilegios.

2. **Primarios:** Los datos están tal y como se han recogido de la fuente, con el nivel de granularidad más alto posible, de forma que no se publican en forma agregada o modificada.
3. **Puntuales:** Los datos se hacen disponibles tan rápido como sea necesario para preservar el valor de los datos.
4. **Accesibles:** Los datos están disponibles para el rango más amplio de usuarios para el más amplio rango de propósitos.
5. **Procesables por máquina:** Los datos están estructurados razonadamente para permitir un procesamiento automatizado.
6. **No discriminatorios:** Los datos están disponibles para cualquier persona, sin requisito de registro.
7. **Sin propietario:** Los datos están disponibles en un formato sobre el cual no haya entidad que tenga su control exclusivo.
8. **Licencia libre:** Los datos no están sujetos a regulaciones sobre copyright, patentes, marcas o secretos comerciales. Pueden permitirse restricciones de privacidad, seguridad y privilegios razonables.

Además, disponen de otros siete principios adicionales:

1. **En la red y gratuitos:** La información no es significantemente pública si no está disponible en Internet sin coste, o al menos no más que un coste marginal de reproducción. También se deben de encontrar fácilmente.
2. **Permanentes:** Los datos deben de hacerse disponibles en una localización de Internet estable indefinidamente y en un formato estable de datos por el mayor tiempo posible.
3. **De confianza:** Según la Recommendation on Open Government de la ACM (febrero de 2009), “El contenido publicado debe de estar firmado digitalmente o incluir un testimonio de la fecha de publicación/creación, autenticidad e integridad.” Las firmas digitales ayudan al público a validar la fuente de los datos que encuentren, de forma que puedan confiar en que esos datos no hayan sido modificados desde que se publicaron. Dado que el origen es para los datos originalmente publicados, esta no es una razón para prevenir al público de modificar los documentos del gobierno.
4. **Presunción de Sinceridad:** la presunción de sinceridad se apoya en leyes como la de Libertad de Información [estadounidense], procedimientos que incluyen la gestión de registros y herramientas como los catálogos de datos.
5. **Documentados:** La documentación sobre el formato y significado de los datos ayudan a hacer los datos útiles.
6. **Seguros para Abrir:** Según la Recommendation on Open Government de la ACM (febrero de 2009), “Los cuerpos de gobierno que publican datos en línea siempre deben buscar publicar usando formatos de datos que no incluyan contenido ejecutable.” El contenido ejecutable en los documentos genera un riesgo de seguridad a los usuarios de los datos porque el contenido ejecutable puede ser algún programa maligno (virus, gusanos, etc.).
7. **Diseñados con Contribución Pública:** El público está en la mejor posición de determinar qué tecnologías de la información serían las más adecuadas para las aplicaciones que el público tenga intención de crear para él mismo. La contribución de público es, en consecuencia, crucial para diseminar la información de forma que tenga valor.

Para contrastar con otra fuente, tenemos los seis principios que propone la Open Data Charter:

1. **Abiertos por defecto:** Esto representa un movimiento real en cómo el gobierno opera y cómo interactúa con los ciudadanos. Por ahora a menudo que tenemos que pedir a los funcionarios la información específica que queremos. Este principio cambia radicalmente esto y dice que tiene que haber presunción de la publicación de todo.
2. **Puntuales y Completos:** Los datos abiertos solo son de valor si siguen siendo relevantes, Tener la información publicada rápidamente y de manera completa es crucial para su potencial de éxito. En medida de los posible los gobiernos deben de proveer los datos en su forma original y no modificada.
3. **Accesibles y Usables:** Asegurar que esos datos los puede leer una máquina y que se puedan encontrar fácilmente harán que los datos vayan más lejos. Los portales son una forma de conseguir esto. Pero también es importante pensar en la experiencia de usuario de aquellos que acceden a los datos, incluyendo los formatos en los que se provee esa información. Los datos deben de estar libres de cargos, bajo una licencia abierta, por ejemplo, esos desarrollados por Creative Commons.
4. **Comparables e Interoperables:** Los datos tienen un efecto multiplicador. Cuanto mayor sea la calidad de los conjuntos de datos a los que tenemos acceso y más fácil sea la comunicación otros, más valor potencial se puede extraer de ellos. Los estándares de datos comúnmente acordados juegan un papel crucial en hacer que esto pase.
5. **Para un Gobernanza Mejorada y Compromiso Ciudadano:** los datos en abierto tienen la capacidad de dejar que los ciudadanos (y otros en el gobierno) tengan una mejor idea de los que los funcionarios y políticos están haciendo. Esta transparencia pude mejorar los servicios públicos y ayudar a que los gobiernos se consideren responsables.
6. **Para un Desarrollo Inclusivo e Innovación:** Finalmente, los datos abiertos pueden ayudar a estimular un desarrollo económico inclusivo. Por ejemplo, un mayor acceso a los datos puede hacer que la ganadería sea más eficiente, o pueden usarse para afrontar el cambio climático. Finalmente, a menudo pensamos en el Open Data como simplemente mejorar el rendimiento del gobierno, pero hay un universo entero ahí fuera de empresarios que ganan dinero gracias a los datos abiertos.

2.1.3. La importancia de los datos abiertos

Hoy en día, y con el número inmenso de aplicaciones que los datos abiertos tienen, se hace imprescindible tener acceso a información pública y usarla en beneficio de la sociedad. Según el Open Data Handbook de la OKF, la importancia de los datos abiertos radica, especialmente, en los gobiernos, tanto nacionales como regionales como locales, y la transparencia que estos muestran a la ciudadanía. Muchos individuos y organizaciones recopilan una gran variedad y volumen de datos para la realización de ciertas tareas, haciendo hincapié en los gobiernos, ya que estos recopilan una gran cantidad y variedad de datos los cuales, por ley, deben de ser públicos y estar al alcance de cualquier persona.

Para Monino y Sedkaoui (2016), la filosofía detrás de este movimiento está centrada en el ciudadano, viendo una mejor imagen de las instituciones públicas gracias al acceso libre a sus datos. Estos datos abiertos deben de ayudar a enriquecer el debate democrático, estimular la vida pública y contribuir a la mejora de los servicios públicos. Además, añaden: “El Open Data es un proceso político cuyo mensaje está construido entorno a la transparencia en la innovación y el desarrollo de la acción pública”.

Actualmente, existen diversos movimientos que exigen una mayor transparencia democrática por parte de los gobiernos, incluyendo una mayor apertura de la información gubernamental como uno de sus fines. De esa manera, se busca vincular a ciudadano como un agente de fiscalización (Open Knowledge Foundation, 2012).

Al parecer, estos movimientos están consiguiendo su objetivo de reclamar una mayor apertura de la información, haciendo ver a líderes mundiales tan importantes como el anterior presidente de los EE. UU. Barack Obama la importancia de los datos abiertos, mencionando estas palabras, las cuales están plasmadas en el libro de Gurin (2014): "Los datos abiertos van a ayudar a lanzar más startups. Van a ayudar a lanzar más negocios. […] Van a ayudar a más empresarios a que se les ocurran nuevos productos y servicios que todavía no hemos imaginado."

Según Gurin (2014), estos datos abiertos se están convirtiendo en un activo muy valioso en la economía, ayudando a pequeños empresarios, a inversores para analizar riesgos, a dueños de compañías para comprender pistas sutiles sobre la reputación de su marca y establecer estrategias basadas en datos, y a empresarios para ofrecer nuevos servicios que beneficien al público, creando startups que pueden facturar millones de dólares y con capacidad de crecer mucho más.

Este autor incluso se atreve a comparar el potencial que tiene el Open Data con el que tenía la Web en 1994, cuando esta todavía estaba en pañales, con tecnologías rudimentarias en comparación con las actuales, así como con una importante fracción menos de información de la ingente cantidad que hay hoy en día.

Los datos abiertos también están teniendo un impacto en cuanto a "darle poder a los ciudadanos para tomar control de sus vidas y pedir cambios habilitando una toma de decisiones más informada y nuevas formas de movilización social", así como para resolver varios grandes problemas públicos, ayudando a ciudadanos y legisladores a buscar la manera de solventar o reducir el impacto de ciertas crisis (Young & Verhulst, 2016).

De acuerdo con los últimos datos que ofrece el Portal Europeo de Datos, el valor del mercado de los datos abiertos en 2019 se establece en 184,45 miles de millones de euros, con un pronóstico de que ascienda a 199,51-334,2 miles de millones de euros para 2025. Además, este sector genera un poco más de un millón de empleados actualmente, pudiendo llegar a ascender hasta los casi dos millones en solo cinco años. También se habla de mejoras de eficiencia y bajadas de costes en sectores tan diversos como el sanitario, el transporte, la producción de electricidad y los servicios públicos gracias a los datos abiertos, los cuales también tienen potencial de mejorar sectores como la agricultura, el comercio y el inmobiliario (Huyer & van Knippenberg, 2020).

En este documento, también se habla de siete conclusiones clave, las cuales son (Huyer & van Knippenberg, 2020):

1. "La especificación e implementación de conjuntos de datos de alto valor como parte de la nueva Directiva de Datos Abiertos es una oportunidad prometedora para abordar las demandas de calidad y cantidad de los datos abiertos."
2. "Es importante abordar las demandas en calidad y cantidad, aunque no son suficientes como para alcanzar el mayor potencial de los datos abiertos."
3. "Los usuarios de datos abiertos reutilizados tienen que ser conscientes y capaces de entender y aprovechar el potencial."
4. "La creación de valor de los datos abiertos forma parte de un reto más amplio de la transformación de destreza y proceso; un proceso prolongado cuyos cambios e impacto no son siempre fácil de observar y medir."
5. "Las iniciativas específicas de un sector y la colaboración entre el sector público y privado fomentan la creación de valor."
6. "Combinar los datos abiertos con datos personales, compartidos, o de origen popular es vital para el cumplimiento de un crecimiento en el mercado de los datos abiertos."

7. “Para diferentes retos, debemos de explorar y mejorar múltiples enfoques de la reutilización de los datos que sean éticos, sostenibles, y adecuados a su propósito.”

Sin embargo, la Open Knowledge Foundation (2012) advierte que los datos abiertos no deben de tener ninguna restricción en cuanto a su reutilización por terceros, ya que supondría una limitación en la capacidad que tienen de crear valor. Un ejemplo muy oscuro de ello es el programa PRISM de la NSA (National Security Agency) norteamericana, el cual para Gurin (2014) es el ejemplo principal de la antítesis a los datos abiertos y del lado oscuro del Big Data, ya que con este programa se recolectaron una enorme cantidad datos personales de millones de estadounidenses sin su consentimiento, ni siquiera con su conocimiento, la cual se ocultó del público hasta la filtración por parte de Edward Snowden en 2013, aunque Gurin no considera como “datos abiertos” a aquellos filtrados del programa PRISM, ya que no fue la propia NSA la que publicó los datos.

2.1.3. Aplicaciones de los datos abiertos

Los datos abiertos, como se ha visto en el apartado anterior, tienen una gran importancia en un amplio abanico de áreas. Una de estas áreas es la creación de valor de múltiples productos y servicios, además de la creación de otros nuevos que pueden ser de gran ayuda a la ciudadanía, y mejorar ya existentes. Dentro de los datos abiertos, se destaca el papel importante que juegan los gobiernos en su publicación, ya que generalmente son los que poseen los portales de datos abiertos. Hoy en día, estos son cruciales para mostrar ser gobiernos transparentes y que rinden cuentas a sus ciudadanos, así como para hacer los servicios que ofrecen más eficientes. En el terreno científico, estos también son importantes, ya que facilitan la innovación y pueden ayudar a alcanzar ciertas conclusiones gracias a la ciencia de datos. Estos son unas cuantas áreas que aparecen en una larga lista en el Open Data Handbook (Open Knowledge Foundation, 2012).

Como ejemplos de aplicación, la OKF, en su Open Data Handbook (2012) (aunque también se incluirán ejemplos de otras publicaciones debidamente referenciados), destaca los siguientes:

- Transparencia:
  - Proyectos como el *Where does my money go?* británico muestran cómo están gastando esos gobiernos el dinero recaudado de los impuestos.
  - En Canadá, gracias al Open Data, el Gobierno ahorró 3,2 millones de dólares en términos de detectar evasión fiscal usando donaciones de caridad.
  - La creación de portales de datos, como el *Data.gov* estadounidense o el *Data.gov.uk* del Reino Unido, están poniendo al servicio de la ciudadanía miles de conjuntos de datos del gobierno gratuitamente (Gurin, 2014).
  - Muchos gobiernos alrededor del mundo están creando portales de transparencia, como es el ejemplo de Brasil, cuyo portal es una de las principales herramientas anticorrupción del país (Young & Verhulst, 2016).
- Toma de decisiones y participación ciudadana
  - En Países Bajos, el servicio *vervuilingsalarm.nl* avisa a una persona con una notificación si la calidad del aire en su zona rebasará cierto umbral previamente definido.
  - Servicios como *Mapumental* (Reino Unido) y *Mapnificient* (Alemania) permiten encontrar lugares donde vivir, tomando en cuenta parámetros como la duración del viaje hasta el trabajo, precios de viviendas y el atractivo de la zona.
    - También se podría mencionar el *New York City Business Atlas*, el cual ayuda a los pequeños de negocios dónde abrir una tienda, buscando de esta forma paliar una brecha de información de investigación de mercados entre los grandes y pequeños negocios (Young & Verhulst, 2016);

- Las técnicas en *análisis de sentimientos* recolectan información de Twitter, blogs y otras fuentes públicas, y usan análisis de texto para convertir esta información en datos abiertos útiles para las compañías en tomas de decisiones estratégicas y de márketing (Gurin, 2014).
- Ciertas páginas webs ayudan a consumidores a tomar decisiones sobre la mejor alternativa a escoger entre las que hay en el mercado, sea en seguros de salud, préstamos, tarjetas de crédito o educación. Hay nuevos negocios que están desarrollando webs y apps para este fin, dando a los consumidores los datos necesarios para tomar decisiones importantes y complejas (Gurin, 2014). Ejemplos (Young & Verhulst, 2016):
  - *A Tu Servicio*, un comparador de seguros de salud en Uruguay.
  - La plataforma *Mejora Tu Escuela*, con información sobre los colegios mexicanos.

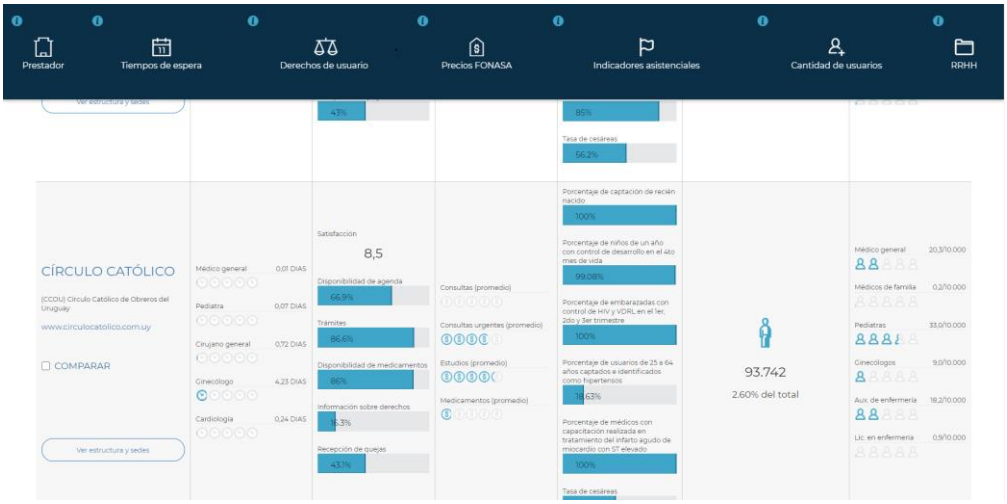

Figura 2.1: A Tu Servicio (Fuente: atuservicio.uy).

- Economía
  - El sitio danés *husetsweb.dk* ayuda a encontrar maneras de mejorar la eficiente energética de un hogar, incluyendo planificación financiera e información sobre constructores especializados.
  - Gracias a sitios web con datos sobre empresas, los inversores pueden encontrar datos de compañías muy prometedoras minimizando riesgos, obteniendo datos de desde startups innovadoras a compañías globales (Gurin, 2014).
  - Las empresas se están volviendo más transparentes sobre sus operaciones lanzando datos abiertos para atraer nuevos inversores, reclutar personal más eficientemente y mejorar su imagen corporativa (Gurin, 2014).
- Creación de nuevos productos y servicios
  - Los empresarios están construyendo nuevos negocios que generan varios millones de dólares de beneficio, gracias a los datos publicados por agencias meteorológicas estatales y datos de GPS (Gurin, 2014).
  - Podrían verse pronto nuevos negocios que usen datos abiertos de salud, así como de otros sectores como finanzas, energía y educación (Gurin, 2014).
- Mejora de productos y servicios existentes
  - Google Translate usa enormes volúmenes de datos de documentos de la Unión Europea de cualquier idioma para entrenar sus algoritmos de traducción y mejorar la calidad de las traducciones.
- Eficiencia en servicios públicos
  - El Ministerio de Educación neerlandés abrió sus datos y, desde entonces, reciben menos preguntas, lo cual conllevó a un menor volumen de trabajo y los

costos, así como en la dificultad de las preguntas que reciben, ya que las importantes quedan claramente respondidas.

- Ciencia y salud
  - Nuevas combinaciones de datos pueden ayudar a descubrir nuevas ideas y conocimientos.
    - Ya en el siglo XIX, el Dr. Snow estableció una correlación entre la contaminación del agua y el contagio del cólera en Londres. Esto llevó a la construcción de un sistema de alcantarillado gracias al cual mejoró enormemente la salud de los londinenses.
  - Los investigadores están acelerando el ritmo de sus nuevos descubrimientos gracias a la publicación de datos en abierto, incluso en el mundo secreto de la investigación de medicamentos, compartiendo datos en redes con expertos y aficionados para que puedan trabajar con los datos y se logren nuevos avances (Gurin, 2014).
  - Hay varios proyectos en cuanto a salud en los que se ha hecho una aplicación de los datos abiertos para ayudar a comprender a la ciudadanía el impacto de cierta enfermedad. Ejemplos de esto son:
    - El proyecto *Battling Ebola* de Sierra Leona ayudó a mejorar significantemente la calidad y la accesibilidad de la información durante la epidemia de ébola en África Occidental del 2014 (Young & Verhulst, 2016).
    - En Singapur se comenzó a compartir información y se creó un mapa de brotes de dengue en el 2005, ayudando a reducir el impacto de una epidemia de esta enfermedad en el 2013 (Young & Verhulst, 2016).
    - Actualmente, durante la pandemia de COVID-19, se han creado varias iniciativas, tanto por parte de gobiernos como otras instituciones y periódicos, creando portales en los que se muestran las cifras de contagiados actualizadas y las gráficas de cómo el número de contagiados o fallecidos evoluciona en cierto territorio, así como mapas interactivos que muestran los contagios por regiones o los distintos brotes que se están generando. Estos portales y mapas usan los datos abiertos de distintos gobiernos nacionales y regionales, así como organizaciones internacionales. Un buen ejemplo de aplicación de los datos abiertos sobre la COVID-19 es el mapa creado por la Universidad John Hopkins estadounidense o el portal de la Junta de Andalucía con información dinámica sobre la expansión de este coronavirus en sus provincias y localidades.

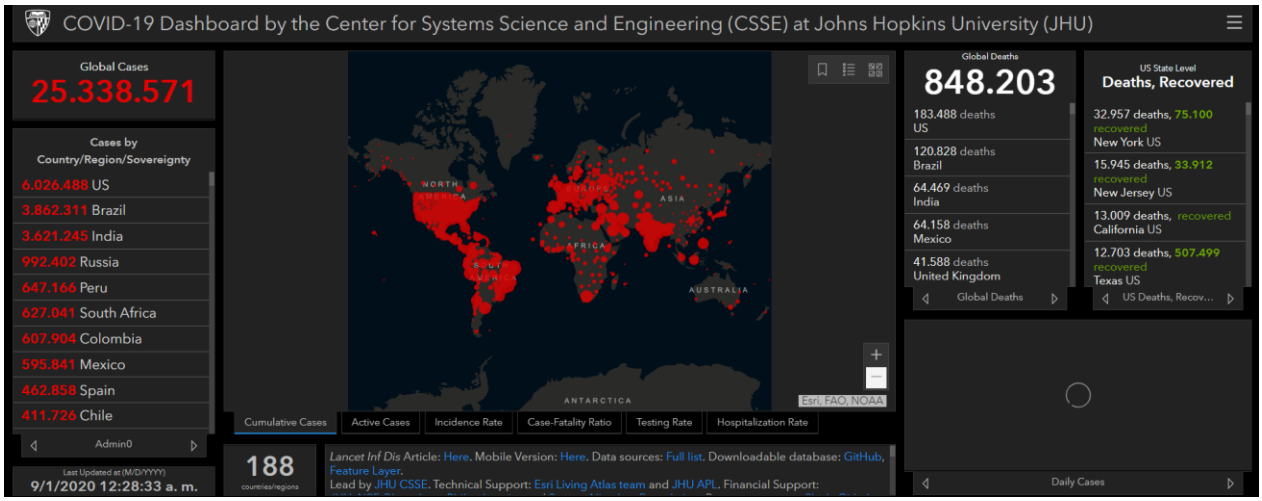

Figura 2.2: Mapa de seguimiento de la expansión de la COVID-19 de la Universidad John Hopkins (Fuente: coronavirus.jhu.edu).

2.1.4. Formatos y almacenamiento de datos abiertos

Estos datos abiertos pueden publicarse en una gran cantidad de formatos. Un portal de datos abiertos, al fin y al cabo, no deja de parecerse en cierta manera a una página dedicada a la descarga de archivos, o a una dedicada a enlazar hacia el contenido.

Los formatos más comunes en los que los gobiernos publican datos abiertos son CSV, JSON (también en su variante geográfica, GeoJSON), HTML (los cuales normalmente son enlaces a otras páginas gubernamentales como el INE), XLS(X) (Excel), PDF, XML y RDF, aunque los formatos en los que publican no se limitan a estos, ya que también se pueden encontrar documentos en formatos tan diversos como ZIP, JPG, DOC(X) o ASCII.

Lo ideal es que estos formatos faciliten su reutilización, ya que, como se ha visto anteriormente, uno de los fines de los datos abiertos es que estos se puedan reutilizar. Es por eso por lo que algunos de los formatos mencionados (como PDF, ZIP, JPG…) no son los ideales para publicar un conjunto de datos en abierto, debido a que es muy difícil realizar un tratamiento adecuado de los datos que contienen. Como bien dice Antonio Almansa Morales (2017), “Si los formatos utilizados no permiten el tratamiento, acceso, reutilización y facilidad de lectura automática por máquina, el open data está condenado a fracasar”.

Otro detalle importante es que estos datos abiertos estén en un formato sujeto a licencias abiertas (por ejemplo, Creative Commons) que permitan su reutilización, para que su reutilización no se vea limitada por restricciones legales u económicas. Además, lo ideal sería que esos datos se dispongan en distintos formatos para que las personas que reutilicen esos datos puedan tratarlos independientemente del software que usen (Almansa Morales, 2017).

En el libro de Monino y Sedkaoui (2016), se citan las siguientes palabras de Tim Berners-Lee, fundador de la World Wide Web, el cual también está muy implicado en el mundo del Open Data: “Si compartimos los datos en Internet – datos públicos, datos científicos, datos de los ciudadanos, cualesquiera – entonces otras personas serán capaces de desarrollar maravillosas creaciones con esos datos las cuales jamás podíamos haber imaginado”.

Tim Berners-Lee también estableció, en el 2010, un conjunto de criterios para medir la calidad de los datos en una escala de cero a cinco estrellas, en lo que se entiende que cero estrellas sería que esos datos no fuesen públicos. La página 5stardata.info sería un complemento a la tabla que se muestra a continuación:

N.º de estrellas	**Descripción**	**Beneficios**
★	Hacer disponibles los datos en Internet en cualquier formato sin restricciones por licencia.	Los usuarios pueden ver, imprimir y guardar los datos, asimismo como seleccionarlos manualmente en un sistema.
★★	Publicar los datos en un formato estructurado (por ejemplo, un archivo Excel en lugar de un archivo escaneado).	Los datos se pueden procesar automáticamente, visualizar y transformar a otros formatos.
★★★	Usar un formato de código abierto (por ejemplo, CSV en lugar de Excel).	Los datos pueden ser manipulados independientemente de su formato y de cualquier software.
★★★★	Usar URIs para identificar elementos para hacer clic en ellos.	Los datos se pueden enlazar, marcar y reutilizar.

★★★★★	Enlazar los datos a datos de otros para darles contexto.	Los patrones en los datos pueden ser identificados automáticamente y es posible descubrir dinámicamente datos complementarios relacionados con los datos originales

Tabla 2.1: Open Data en cinco etapas (Monino & Sedkaoui, 2016).

2.1.5. Legislación sobre Open Data en España

La legislación en España sobre los datos en abierto está todavía en pañales, limitándose a establecer ciertas restricciones en la apertura de datos.

Si bien los principios del Open Data asumen que la información pública tiene que abrirse, no siempre se van a publicar los datos en abierto debido a ciertas razones que aparecen en el artículo 3 de la Ley 37/2007, de 16 de noviembre, sobre Reutilización de la Información del Sector Público tras la modificación realizada en su texto por la Ley 18/2015, de 9 de julio (Almansa Morales, 2017). Esa información sería, principalmente:

a) Los documentos sobre los que existan prohibiciones o limitaciones del derecho de acceso según lo previsto en las normas que regulan este derecho.
b) Documentos reservados, secretos y confidenciales (sobre seguridad del Estado, defensa nacional, los obtenidos de la administración tributaria y de la Seguridad Social, secretos comerciales…).
c) Documentos para cuyo acceso se requiere un derecho o interés legítimo.
d) Documentos en manos de las instituciones públicas ajenas a la función de servicio público.
e) Documentos sobre los que exista una propiedad intelectual o industrial por parte de terceros.
f) Documentos conservados por las entidades que ofrecen servicios esenciales de radiodifusión y televisión y sus filiales.
g) Documentos producidos o conservados por parte de instituciones educativas o de investigación.
h) Documentos producidos o conservados por instituciones culturales que no sean bibliotecas, incluidas las universitarias, museos y archivos.
i) Partes de documentos que solo incluyan logotipos, divisas e insignias.
j) Documentos a los que no se tiene acceso o se tiene acceso limitado por motivos de protección de datos.
k) Documentos elaborados por entidades del sector público empresarial y fundacional en el ejercicio de las funciones atribuidas legalmente y los de carácter comercial, industrial o mercantil elaborado en ejecución del objeto social previsto en sus estatutos.
l) Estudios realizados por el sector público en colaboración con el privado.

El artículo 4 de esa ley establece que no se puede abrir la información que resulte en una violación de la protección de datos de carácter personal, es decir, que se publiquen datos que una persona no ha consentido su difusión pública.

En resumen, la información que se excluiría de ser abierta por las instituciones públicas puede ser por motivos de intereses públicos, por intereses privados de terceros y por motivos de intereses colectivos (Almansa Morales, 2017).

Esta ley también habla, en otros artículos, de formatos disponibles para la reutilización, prohibición de derechos exclusivos, licencias y tarifas, en línea a los conceptos y principios anteriores.

La Ley 37/2007, de 16 de noviembre, sobre Reutilización de la Información del Sector Público tras la modificación realizada en su texto por la Ley 18/2015, de 9 de julio es la principal ley en el marco legal sobre los datos abiertos. Su equivalente europeo sería la Directiva 2003/98/CE del Parlamento Europeo y del Consejo, de 17 de noviembre de 2003, relativa a la reutilización de la información del sector público. A nivel autonómico está la Ley 1/2014, de 24 de junio, de Transparencia Pública de Andalucía, una ley de transparencia que profundiza también en los datos en abierto dentro de los organismos públicos andaluces. También habría que mencionar el Real Decreto 1495/2011, de 24 de octubre, por el que se desarrolla la Ley 37/2007, de 16 de noviembre, sobre reutilización de la información del sector público, para el ámbito del sector público estatal, aunque hay un borrador de un nuevo Real Decreto referente a ello; y un convenio de colaboración entre la Administración General del Estado y [la entidad pública empresarial] Red.es de 2019 a 2023.

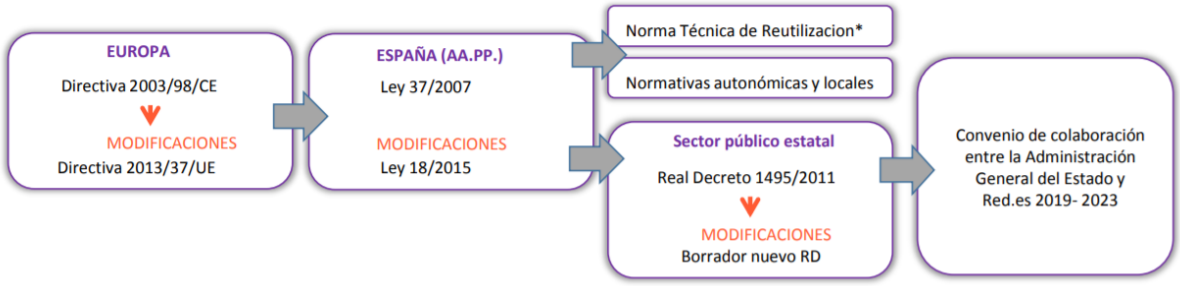

Figura 2.3: Marco legislativo sobre el Open Data en España (Iniciativa Aporta, 2019).

2.2. Portales de datos abiertos

En esta sección, tras haber introducido en el anterior el fenómeno del Open Data, se hará un enfoque en los lugares en los que, generalmente, se publican estos datos, definiendo las funciones que tiene, las distintas alternativas que hay disponibles para crear un portal de datos abiertos y la elección de una de ellas para desarrollar el proyecto, así como la instalación, configuración y creación de conjuntos de datos en el portal de datos abierto a usar.

2.2.1. ¿Qué es un portal de datos abiertos? ¿Cómo es?

Como se ha dicho anteriormente, son sitios web, normalmente pertenecientes a entidades públicas, en los que se publican conjuntos de datos para que la ciudadanía los pueda reutilizar y desarrollar con ellos aplicaciones que exploten el valor potencial de esos datos.

Este se compone de varias secciones, siendo la más interesante el catálogo de datos, donde se encuentran los distintos conjuntos de datos que están publicados en la web, y estos conjuntos de datos, en los distintos formatos en los que se pueden descargar. Pero esta no es la única sección que suelen tener, ya que se pueden consultar las fichas de las distintas organizaciones a las que pertenecen los conjuntos de datos que se publican, así como se pueden encontrar diferentes secciones como noticias, información para desarrolladores, información sobre ciertos temas específicos, información sobre la estrategia de datos abiertos del gobierno y distintos añadidos que las organizaciones dueñas de los portales de datos vean oportunas.

Estos también, y según qué portal de datos y qué software usa, además de permitir la navegación por el catálogo de datos haciendo uso de un navegador web, también pueden

permitir la navegación y descarga de conjuntos de datos haciendo uso de una API integrada en el software que usa el portal de datos.

Hay muchos ejemplos de portales de datos abiertos, como el data.gov estadounidense, data.gov.uk del Reino Unido, datos.gob.es del Gobierno de España, el europeandataportal.eu de la Unión Europea, el portal de datos abiertos de la Junta de Andalucía (juntadeandalucia.es/datosabiertos/portal.html), el madrileño datos.madrid.es, data.worldbank.org del Banco Mundial, entre otros.

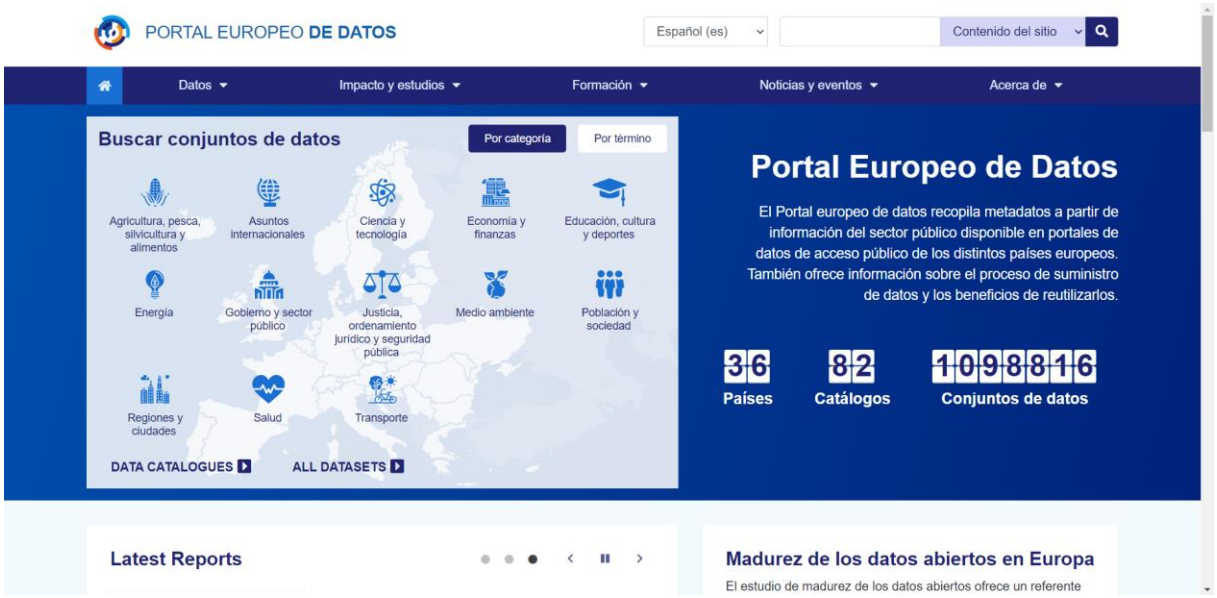

Figura 2.4: Página principal del Portal Europeo de Datos (Fuente: europeandataportal.eu/es)

2.2.2. Alternativas de software para la creación de portales de datos abiertos

Dentro del software existente para portales de datos en abierto, nos encontramos las siguientes soluciones para poder crear un portal de datos abiertos:

- CKAN *(DKAN, DataPress)*
- OpenDataSoft
- Socrata
- Esri ArcGIS Open Data
- Junar
- Soluciones de desarrollo propio

2.2.2.1. CKAN

CKAN (Comprehensive Knowledge Archive Network) es un software de código abierto para portales de datos abiertos desarrollado y mantenido por la Open Knowledge Foundation, y uno de los más usados alrededor del mundo para este cometido. Su cometido es ayudar a la gestión y publicación de colecciones de datos, y es usado por gobiernos nacionales y locales, instituciones de investigación y otras organizaciones que recogen muchos datos (CKAN, 2017). Actualmente se encuentran en su versión 2.9.

Este permite a los usuarios, una vez subidos los datos, usar las funciones de búsqueda para encontrar los datos que ellos necesiten, y previsualizar los datos en forma de mapas, tablas o grafos (CKAN, 2017).

Como se ha dicho anteriormente, es un software Open Source (bajo la licencia GNU GPL v3.0) con una comunidad activa de contribuyentes que desarrollan y mantienen su tecnología

central, además de ser modificado y extendido por una gran comunidad de desarrolladores (CKAN, 2017), la cual ha creado más de 200 extensiones[1] para CKAN.

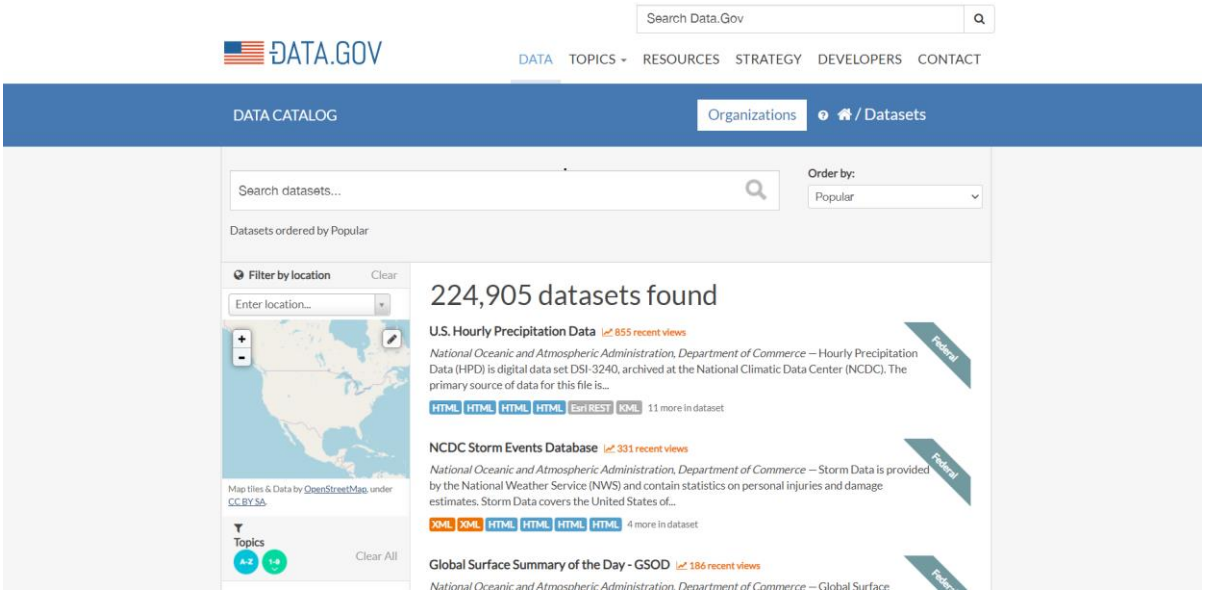

Figura 2.5: Catálogo de datos de data.gov, el cual usa DataPress (Fuente: catalog.data.gov/dataset).

Sin embargo, suele ser necesario un equipo técnico especializado para implementar y mantener una instancia de CKAN, ofreciendo la propia OKF un servicio en el que se encargan de realizar esas labores (McGreggor, 2014).

Dentro de este, hay varias alternativas, las cuales usan un CMS diferente, como DataPress (integración de WordPress en CKAN) o DKAN (el cual incluye las características de CKAN en Drupal). Estos CMS permiten la creación de nuevas páginas a parte de las de conjuntos de datos, organizaciones y grupos que permite originalmente CKAN. Por ejemplo, data.gov usa DataPress, ya que con la herramienta cmsdetect.com se ha detectado el uso de WordPress, mientras que un ejemplo de portal DKAN puede ser datos.gob.es.

Junto a los dos anteriores, podemos citar como ejemplos de usos de CKAN también data.gov.uk, el Portal de Datos Europeo u opendata.swiss. Características

Las páginas de CKAN (sin tener en cuenta su integración a otros CMS) son las siguientes:

- **Página principal:** en esta página, se pueden exhibir ciertos conjuntos de datos y grupos de estos. También incluye una barra de búsqueda y la opción de colocar una imagen.
- **Conjuntos de datos:** permite la búsqueda de los distintos conjuntos de datos que existen en el portal de datos abiertos. Pueden usarse parámetros como formatos, organizaciones, idiomas, etiquetas… Cada conjunto de datos puede contener varios recursos, los cuales pueden contener el conjunto de datos en distintos formatos o divisiones según cierta granularidad, además de sus metadatos.
    - **Recursos:** es el conjunto de datos en sí. Estos se asocian a un archivo, el cual está enlazado, pudiendo estar almacenado en la propia instancia de CKAN (habiendo sido subido haciendo uso de la extensión FileStore) o ser un recurso remoto almacenado en otro servidor. También contiene los metadatos y, según el formato o las extensiones habilitadas, opciones de previsualización.

[1] Cifra extraída de extensions.ckan.org.

- **Organizaciones:** CKAN permite organizar los conjuntos de datos según las organizaciones que los hayan publicado. Los portales de datos que usan CKAN suelen contener conjuntos de datos pertenecientes a otras organizaciones (por ejemplo, que el portal de datos del gobierno nacional tenga como organizaciones gobiernos regionales o locales).
- **Grupos:** los usuarios pueden crear grupos en los que añaden conjuntos de datos que puedan ver interesantes.

Figura 2.6: Página principal de datos.gob.es, el cual usa DKAN (Fuente: datos.gob.es).

Las funcionalidades incluidas en el núcleo de CKAN son las siguientes:

- **CKAN Action API:** es una API potente, de estilo RPC (llamadas a procedimientos remotos) que expone las funcionalidades principales de CKAN a clientes que consumen la API.
- **DataStore:** extensión que provee una base de datos *ad hoc* para el almacenamiento de datos estructurados de los recursos de CKAN.
- **Extensiones:** CKAN permite a los usuarios que lo vayan a implementar qué funcionalidades escoger para su portal de datos, así como el desarrollo de extensiones, el cual viene documentado en una guía de creación de extensiones (docs.ckan.org/en/latest/extensions/index.html).
- **Federación:** gracias a la extensión de harvesting disponible para CKAN, la cual permite obtener los metadatos de un conjunto de datos de otro portal, CKAN puede ser usado para crear una red de portales de datos abiertos que compartan datos entre ellos.
- **FileStore:** extensión que permite a los usuarios subir archivos e imágenes.
- **Geoespacial:** CKAN contiene funciones geoespaciales avanzadas, cubriendo la previsualización de datos, búsqueda y descubrimiento.
- **Metadatos:** CKAN provee a cada conjunto de datos de un conjunto de metadatos muy rico.
- **Publicación y gestión de datos:** Una interfaz web intuitiva permite a los editores y a los conservadores registrar, actualizar y refinar conjuntos de datos fácilmente.
- **Búsqueda y descubrimiento:** CKAN provee una rica experiencia de búsqueda que permite una rápida búsqueda de palabras clave al estilo de Google así como filtrar por etiquetas y buscar entre conjuntos de datos relacionados.

- **Temas:** Se puede personalizar el aspecto de un portal de datos CKAN siguiendo una guía de creación de temas (docs.ckan.org/en/latest/theming/index.html):
- **Visualización:** la herramienta de previsualización de datos de CKAN alberga características potentes para previsualizar los datos almacenados en el DataStore.

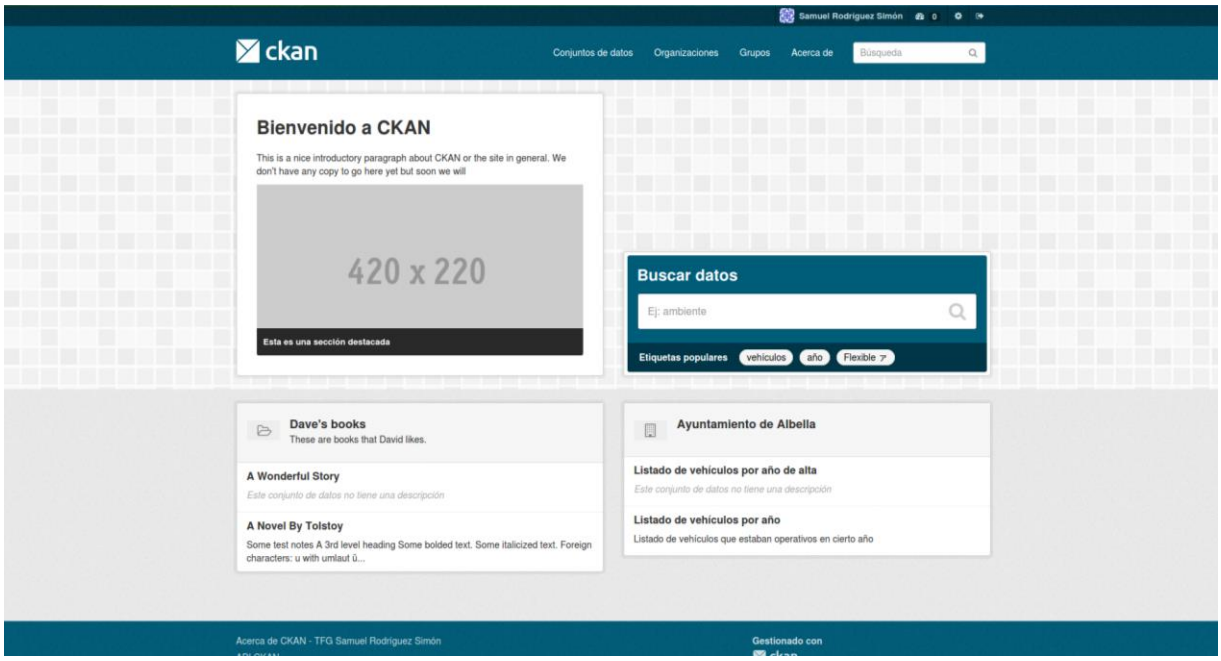

Figura 2.7: Página principal de un portal de datos de CKAN por defecto.

2.2.2.1.1. Tecnología y arquitectura

CKAN usa para su frontend JavaScript, mientras que su backend está desarrollado en Python, teniendo una arquitectura modular para añadir posteriormente extensiones las cuales añaden funciones adicionales como scraping o subida de archivos (CKAN, 2017).

De acuerdo con lo que expone Sean Hammond (seanh en GitHub), antiguo desarrollador de este software, la arquitectura de CKAN de menor a mayor nivel sería la siguiente (Hammond, 2012):

- **Base de datos:** CKAN usa una base de datos PostgreSQL con la cual se comunica a través de SQLAlchemy. También se menciona el uso del framework web Pylons para interactuar con SQLAlchemy (CKAN, 2017).
- **Modelo:** es el paquete ckan/model/ y contiene las clases para las entidades almacenadas en la base de datos de CKAN. Obviamente sería el modelo dentro del concepto Modelo-Vista-Controlador (MVC).
- **Dictization** *(ckan/lib/dictization/)***:** contiene las clases que convierten los objetos del modelo en diccionarios y viceversa. El formato JSON usado por la API de CKAN funciona con estas representaciones de diccionario.
- **Lógica** *(ckan/logic/)***:** contiene las funciones para que la interfaz web, la API y la línea de comandos puedan acceder y manipular los datos almacenados en CKAN. Las funciones de validación y autenticación entrarían en esta parte.
- **CKAN Action API:** Expone en medida de lo posible las funciones de CKAN sin depender de la interfaz web o de la línea de comandos.
- **Interfaz web:** aparte del frontend, sus funcionalidades están implementadas por las clases con funciones de controlador. Como resultado a las llamadas que realiza a la capa lógica, se generan páginas generadas de archivos de plantilla Jinja2. Las funciones de búsqueda son llevadas a cabo por Solr (CKAN, 2017).

- **Línea de comandos:** permite hacer múltiples acciones como gestionar conjuntos de datos y usuarios. Está implementado usando Python Paste Script, usando comandos que comienzan por paster.
- **Plugins:** CKAN incluye interfaces para plugins, los cuales se enganchan a CKAN y pueden añadir o alterar funcionalidades.

2.2.2.2. *OpenDataSoft*

OpenDataSoft es una empresa francesa que ofrece software comercial para portales de datos que surgió a finales de 2011, cuando sus tres fundadores estaban trabajando en la creación del portal de datos abiertos francés. En la actualidad, esta compañía trabaja con centenares de clientes como Schneider Electric, el Ayuntamiento de París o el de Bruselas, estando construidos los portales de datos abiertos de estos dos últimos con su software (OpenDataSoft, s.f.). Su principal producto consiste en una plataforma de datos abiertos basada en una solución Software as a Service (SaaS) (Safe Software, 2015), dedicada al lanzamiento de nuevos servicios al mercado, haciendo uso de la computación en la nube y tecnologías Big Data (OpenDataSoft, 2012).

Figura 2.8: Portal de datos abiertos de la ciudad de París (Fuente: opendata.paris.fr).

2.2.2.2.1. Características

Entre sus características, tenemos que esta solución es idónea para trabajar con grandes conjuntos de datos al hacer uso de Elasticsearch, el cual asegura una búsqueda y análisis de datos casi en tiempo real. Además, soporta formatos geoespaciales y la publicación y gestión de datos puede hacerse a través de una dashboard en tiempo real (Safe Software, 2015). También cuenta con una API REST para la búsqueda y acceso a los datos la cual está ya en su versión 2. Esta API solo soporta el método HTTP GET para obtener datos, los cuales son devueltos en formato JSON (OpenDataSoft, s.f.).

Según distintas páginas al respecto en el apartado de "Crear un conjunto de datos" en la guía de usuario de OpenDataSoft (s.f.), mientras los archivos pesen menos de 240 MB, este soporta los siguientes formatos:

- **CSV** *(.csv, .dat, .txt, .tsv)*
- **Hojas de cálculo:** Microsoft Excel *(.xls, .xlsx)* y OpenDocument *(.ods)*
- **JSON**
- **GeoJSON**
- **KML/KMZ**

- **Shapefile** *(.zip)*
- **MapInfo** *(.zip)*
- **OpenStreetMap** *(.osm)*
- **XML**
- **Archivos comprimidos** *(.zip, .bz2, .tar, .gz, .gzip, .tar.gz, .tgz, .tar.bz2)*
- **Imágenes** *(.gif, .png, .jpeg, .jpg, .tiff, .bmp, .svg)*
- **PDF**
- **GTFS** *(.zip)*

A diferencia de CKAN, OpenDataSoft no realiza distinciones entre recurso y conjunto de datos, requiriendo anteriormente la subida de los conjuntos de datos al propio portal (Neumaier & Umbrich, 2016), aunque al parecer ahora se puede enlazar a archivos remotos, según la guía de creación de conjuntos de datos de OpenDataSoft (s.f.), además de permitir la posibilidad de federar datos de otros portales, con la limitación de que los datos no se duplican, solo los metadatos y las visualizaciones.

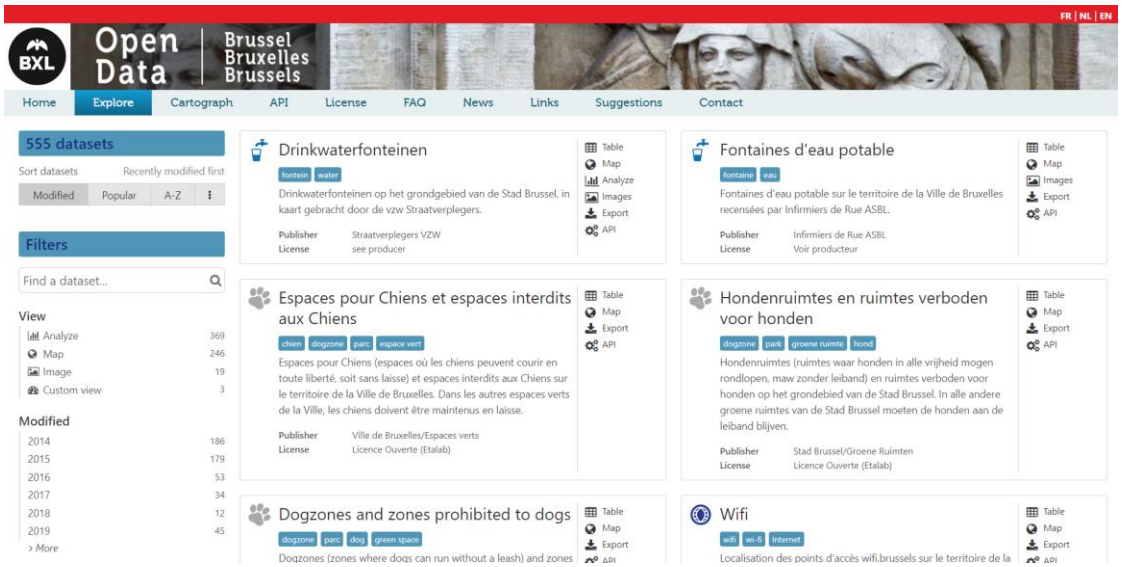

Figura 2.9: Catálogo de datos de Open Data Brussels (Fuente: opendata.brussels.be).

2.2.2.2.2. Tecnología

Entre las tecnologías que esta solución SaaS usa, tenemos que está escrita en Python usando el framework web Django, Hadoop para el procesamiento de datos (OpenDataSoft, 2012), Exalead para realizar las búsquedas y MongoDB como base de datos (OpenDataSoft, 2012), pudiendo usar como hosting de la plataforma diferentes servicios en la nube públicos y/o privados como Amazon Web Services (OpenDataSoft, 2012).

2.2.2.3. *Socrata*

Socrata fue una compañía de software fundada en 2007, cuya sede estaba en Seattle (WA, EE. UU.), que fue adquirida en el 2018 por Tyler Technologies, una compañía de Texas la cual se dedica principalmente al desarrollo de software para el sector público (Lerman, 2018).

El producto de Socrata para el Open Data tiene el nombre comercial de Socrata Open Data & Citizen Engagement Cloud™, el cual, como dice su nombre, se ofrece como un software en la nube, esto es, que albergan los datos en sus propios servidores y proveen acceso a los conjuntos de datos a través de una API (Neumaier & Umbrich, 2016). Al ser un servicio ofrecido por una empresa, no es necesario tener un equipo técnico especializado para su despliegue, al contrario de lo que puede ser necesario para un portal de datos CKAN (McGreggor, 2014).

Ejemplos de portales de datos abiertos Socrata pueden ser el de la Ciudad de Nueva York y el de la Ciudad de Chicago.

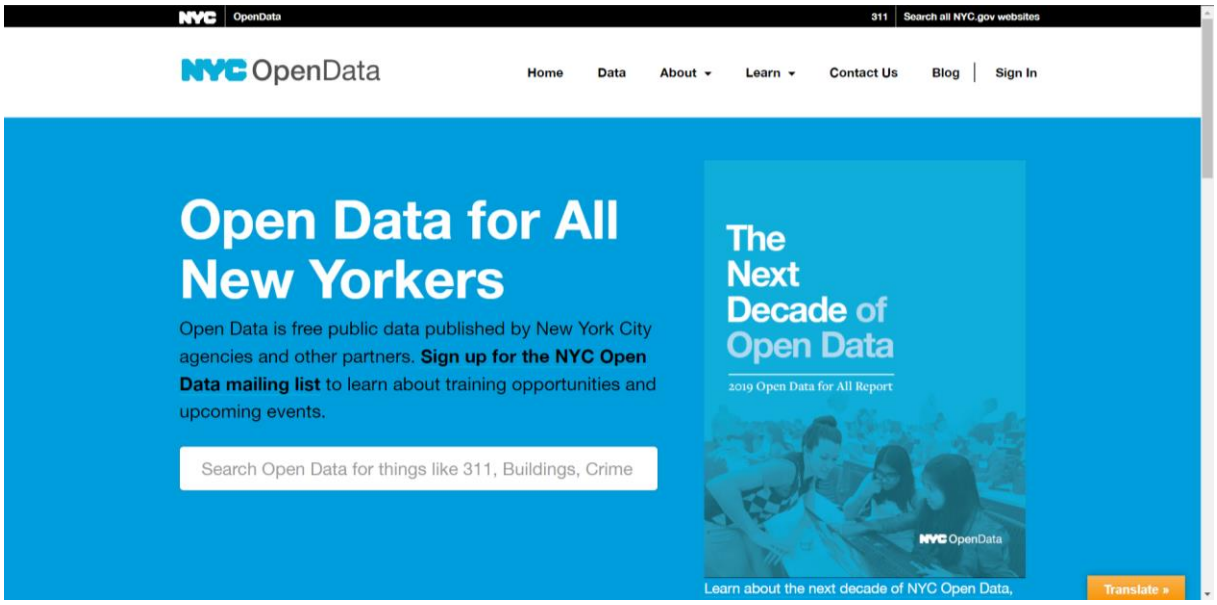

Figura 2.10: Página principal de New York City Open Data, el cual usa Socrata (Fuente: opendata.cityofnewyork.us).

2.2.2.3.1. Características

La plataforma Open Data de Socrata cuenta con las siguientes funciones (Tyler Technologies, s.f.):

- Catálogo de datos abiertos.
- Canal de datos y gestión de metadatos.
- APIs públicas para conjuntos de datos, metadatos y búsqueda. Socrata Open Data API (SODA).
- Mapeo de datos geoespaciales.
- Interoperabilidad entre catálogos mediante data.json.
- Previsualización de datos mediante mapas, gráficos y tablas interactivas.
- Inserción de contenido en páginas web existentes o albergarlo en la página principal del portal.
- Soporte de múltiples formatos como CSV, Excel, JSON, GeoJSON, Shapefile, TSV, KML y KMZ (Safe Software, 2015).
- Facilidad de publicar datos mediante interfaz web, herramienta de escritorio sincronizada o API (Safe Software, 2015).
- Duplicación de conjuntos de datos: publicado y copia funcional (Safe Software, 2015).
- Citizen Connect muestra datos públicos de alto interés sobre un mapa interactivo fácil de usar en teléfonos móviles.
- Las perspectivas ayudan a contar historias en línea – mediante fotos, gráficos insertados, tablas narrativas y más – para conectar los puntos entre las políticas y sus resultados.
- Los paneles de control personalizados resaltan la información clave para los responsables.
- Conexión fácil con otras aplicaciones de la suite de Socrata, como Open Performance, Open Budget, Open Payroll y Open Payments.

2.2.2.4. Otras alternativas

Existen otras soluciones Open Data menos extendidas, pero se mencionarán en este apartado.

2.2.2.4.1. Esri ArcGIS Open Data

Esri es una compañía especializada en Sistemas de Información Geográfica (GIS), cuyo principal producto es ArcGIS, la cual es una suite de aplicaciones en el campo de los GIS. Dentro de esta suite, existe una solución SaaS específica para Open Data, y viene incluida en el paquete ArcGIS Online (Esri, s.f.).

Los portales de datos ArcGIS permiten compartir datos geoespaciales y mapas de otros servicios de ArcGIS, así como datos no geoespaciales como hojas de cálculo, PDFs, tablas (Esri, s.f.), imágenes, CSVs, URLs, documentos Word y otros servicios vía Koop (Safe Software, 2015).

Facilita a los usuarios buscar y descargar conjuntos de datos enteros o subconjuntos filtrados, descubrir datos tanto espaciales como tabulares a través de API (Esri, s.f.).

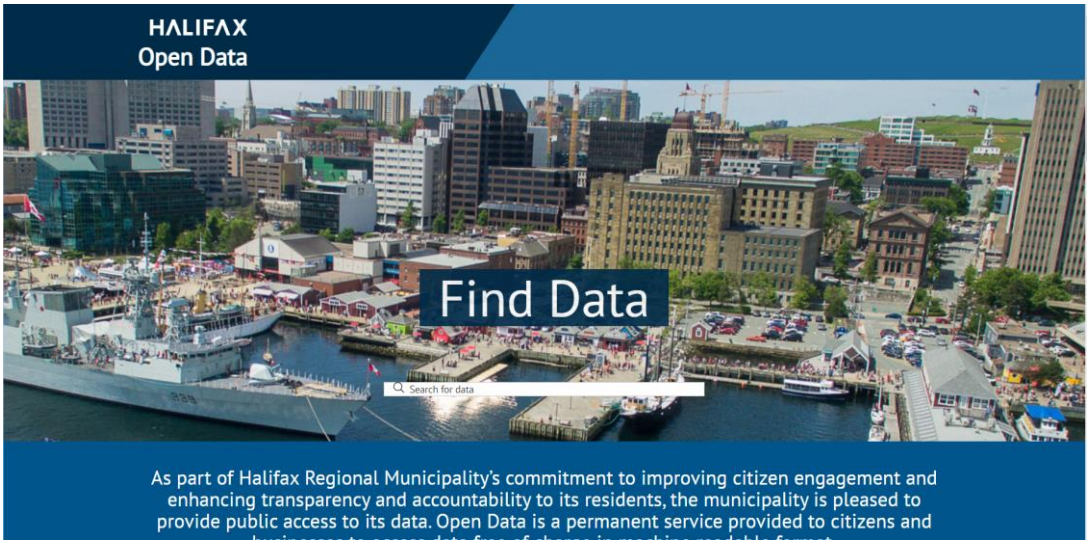

Figura 2.11: Catálogo de datos abiertos de la ciudad de Halifax, Canadá (Fuente: http://catalogue-hrm.opendata.arcgis.com/).

Permite, gracias a una plantilla, tener un portal listo en minutos y editarlo fácilmente, arrastrando y soltando elementos, así como incorporar funciones de ArcGIS Hub para llevar más allá el portal y sus datos abiertos (Esri, s.f.). Sin embargo, esto da un aspecto demasiado parecido a otros portales de datos abiertos montados con este software.

Además, puede mostrar los datos y metadatos en el buscador, y crear diversas gráficas para encontrar patrones en los datos sin necesidad de descargarlos (Safe Software, 2015).

Ejemplos de portales de datos abiertos hechos con ArcGIS son Johns Creek DataHub, el Catálogo de Datos GIS de Maryland o el Halifax Open Data.

2.2.2.4.2. Junar

Es una compañía chilena que ofrece una plataforma de datos SaaS, la cual permite transformar los datos en tablas, visualizaciones y mapas. Además, proporcionan una API para que desarrolladores y empresas puedan reutilizar los datos (Junar, s.f.).

Esta plataforma es usada, por ejemplo, por varias instituciones del Gobierno de Chile, la Ciudad de Palo Alto o la Ciudad de Bahía Blanca.

Figura 2.12: Portal de datos abiertos de la ciudad de Palo Alto, California (Fuente: data.cityofpaloalto.org).

2.2.2.4.3. Desarrollos propios

Es posible desplegar un portal de datos sin depender de ninguna de las alternativas anteriormente planteadas.

Se puede desarrollar un portal de datos abiertos propio haciendo uso de plataformas en la nube como Amazon Web Services (AWS), Microsoft Azure o Google Cloud Platform. Por ejemplo, en AWS, activando servicios de bajo nivel (como S3, EC2 y RDS) y usando una plataforma de integración de datos (como FME Cloud de Safe Software), se puede montar un servicio potente, tolerante a las fallas y escalable el cual es fácil de mantener a un coste un muy alto. Soluciones de este estilo permitirían usarse para almacenar y disponer grandes volúmenes de datos. Es posible construir una interfaz web en lo más alto del servicio de almacenamiento (Safe Software, 2015).

Si no se desea crear un portal de datos abiertos, se pueden compartir esos conjuntos de datos a través de distintos servicios de hosting gratuito como Google Drive, GitHub o algún servidor FTP.

2.3. API REST

Un portal de datos abiertos puede limitarse a ofrecer enlaces a distintos archivos sin dar la posibilidad de realizar consultas. Otra opción podría ser permitir la personalización de alguna forma y poder filtrar a priori los datos a los que se desea acceder. Ahí podemos plantear el uso de una API REST para proporcionar esta funcionalidad.

A continuación, introduciremos este concepto y veremos ciertas alternativas que hay disponibles para su desarrollo.

2.3.1. ¿Qué es una API REST?

Primero, se definirá qué se entiende cuando nos referimos a “API REST”. Para ello, definiremos tanto lo que es una “API” como lo que es “REST”.

Una API (Application Programming Interface, *Interfaz de Programación de Aplicaciones*) se puede definir como un conjunto de funciones y procedimientos ofrecidos normalmente como una librería para permitir a otro software realizar ciertas acciones. Estas funciones permiten

el acceso a ciertas funciones autorizadas para un usuario o una aplicación externa (Pardo, 2019).

Lo siguiente sería dar una definición lo más breve posible de REST. En el libro "Core JavaServer™ Faces" de Geary y Horstmann (2010) se dice que:

> Un estilo arquitectónico llamado REST (REpresentational State Transfer, *[Transferencia de Estado Representacional]*) propone que las aplicaciones web deberían usar HTTP como se concibió originalmente. Las búsquedas deben usar solicitudes GET. Las solicitudes POST, PUT y DELETE deben de ser usadas para la creación, la variación y la eliminación [respectivamente]. (p.89)

Este estilo de arquitectura fue propuesto en una tesis doctoral de Roy Fielding, uno de los principales autores de la especificación HTTP, en el 2000, con la idea de usar HTTP para la comunicación entre máquinas que mecanismos más complejos como CORBA, RPC o SOAP (Elkstein, 2008). Mientras que estos necesitan realizar ciertas operaciones para comunicarse con otra aplicación, una aplicación cliente RESTful (que cumple con los principios de REST) simplemente manda una petición HTTP al servidor y recibe una respuesta a esa petición.

También se suele asociar REST a un estilo de URLs similar al de este ejemplo:

```
http://myserver.com/catalog/item/1729
```

, aunque también sería RESTful una URL para una petición GET con estos parámetros:

```
http://myserver.com/catalog?item=1729
```

Sin embargo, deja de ser RESTful una URL la cual usa un método HTTP (GET, POST, PUT, DELETE…) distinto al cual se tiene intención de usar. Un ejemplo sería usar un método GET para actualizar los datos de una página, para lo cual está el método PUT (Geary & Horstmann, 2010).

Se puede deducir entonces que una API REST es una API basada en el estilo de arquitectura de software REST, es decir, una interfaz entre dos aplicaciones la cual usa, como forma de comunicación entre esas aplicaciones, un mecanismo muy sencillo basado en mandar una petición y recibir una respuesta.

2.3.2. Distintas alternativas para la creación de la API

Vamos a definir las distintas alternativas para ello en cuanto al framework para el backend. El backend consiste en la programación de la lógica del servidor, es decir, cómo el servidor gestiona las peticiones y respuestas que hacen los clientes, así como de la seguridad del acceso a las acciones que realizan ciertas peticiones y respuestas. Un framework ayuda al desarrollo de una aplicación o servicio web.

Las alternativas a explorar son las siguientes:

- Express (+ Node.js). *JavaScript.*
- Django. *Python.*
- Ruby on Rails. *Ruby.*
- Phalcon. *PHP.*
- Laravel. *PHP.*

2.3.2.1. Express

Express es un framework web minimalista y flexible para Node.js (Express, s.f.). Facilita la configuración de un servidor, creando un servidor web para escuchar peticiones relevantes y devolver las respuestas oportunas y definiendo una estructura de directorios, estableciendo

una carpeta para servir archivos estáticos de una forma que no bloquea otros procesos de la aplicación (Holmes, 2019).

Una característica importante de Express es que provee una interfaz simple para dirigir una URL entrante a un trozo de código, pudiendo servir tanto una página HTML estática como una escritura o una lectura de la base de datos, logrando, de esa manera, hacer que el código se pueda escribir más rápido y mantener más fácil (Holmes, 2019).

Las vistas son una forma de crear plantillas para páginas HTML, haciendo mucho más sencilla la creación de estas páginas, usando componentes reusables, así como datos de la aplicación. Express los compila juntos y sirve todo ello en una página HTML (Holmes, 2019).

También soluciona una limitación de Node.js, y es que, al ejecutarse sobre un solo hilo de proceso y no asignar un silo de RAM, no memoriza al usuario de una petición a otra, debido a que ve una serie de peticiones HTTP y este es un protocolo sin estado. Gracias a las sesiones de Express, se puede identificar a usuarios individuales a lo largo del recorrido que hace a través de las páginas (Holmes, 2019).

2.3.2.1.1. La pila MEAN

Express suele usarse como el componente backend de la llamada *pila MEAN*, la cual es una pila de aplicaciones formada por (Holmes, 2019):

- **MongoDB** (base de datos)
- **Express.js** (framework web)
- **Angular.js** (framework front-end)
- **Node.js** (servidor web)

Nótese que MEAN es el acrónimo de las tecnologías en el orden que se han escrito.

Esta pila de aplicaciones se caracteriza por usar simplemente JavaScript para el desarrollo de las aplicaciones, tanto en el frontend (la parte visual de la página web) como para el backend (Holmes, 2019).

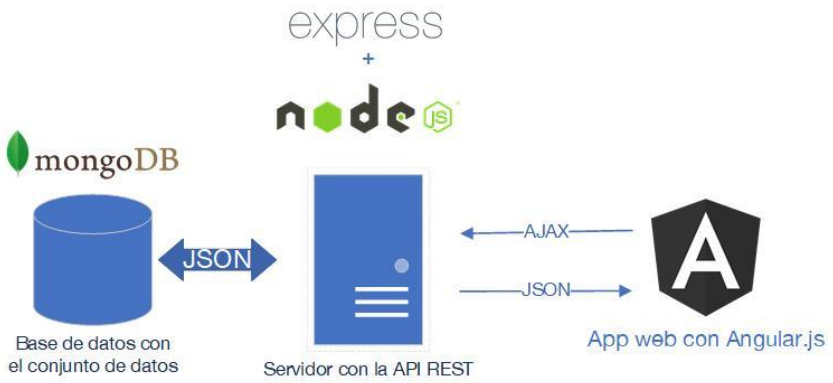

Figura 2.13: Arquitectura de la pila MEAN.

La comunicación entre los distintos módulos se realiza en formato JSON (JavaScript Object Notation), el cual permite representar objetos de una forma legible tanto para humano como para máquina, y fácilmente procesable por cualquier intérprete de JavaScript (Alarcón, 2015).

Existen variantes de la pila MEAN las cuales usan otros frameworks para el frontend en lugar de Angular, como la pila MERN (usa React) o la pila MEVN (usa Vue.js).

2.3.2.1.1.1. MongoDB

Es la base de datos que usa la pila MEAN. MongoDB es una base de datos NoSQL, es decir, en lugar de almacenar los datos en tablas con filas y columnas asociadas a valores, las columnas dejarán de existir y cada fila de la tabla sería un "documento" con una lista de parejas de clave-valor (Holmes, 2019). Las "columnas" serían las distintas claves con el mismo nombre que existen en las filas. Además, es conocida por ser rápida y escalable (Holmes, 2019).

MongoDB almacena los datos en formato BSON, o JSON binario, siendo una representación en binario las estructuras de datos de un archivo JSON, el cual es un formato en el que JavaScript maneja la información, haciendo que se mantenga el enfoque holístico del uso de JavaScript en toda la pila MEAN (Holmes, 2019).

Cada fila de una tabla, además de sus pares clave-valor, tiene una pareja clave-valor el cual es _id, un identificador único hexadecimal que asigna MongoDB a cada nueva fila (Holmes, 2019).

Un ejemplo de entrada en una base de datos MongoDB podría ser la siguiente:

```
{
      "_id" : ObjectId("5f470080aa2f9558568357e5"),
      "tipo" : "Turismo",
      "marca" : "Ford",
      "modelo" : "Fusion",
      "matricula" : "1234 ABC",
      "anio_alta" : 2018,
      "uso" : "Coche patrulla",
      "institucion" : "Policía Local",
      "regimen" : "En propiedad",
      "estado" : false,
      "anio_baja" : 2020
}
```

Para almacenar los datos necesarios para el desarrollo de la API, es necesario que estos tengan una estructura. **Mongoose** es una biblioteca para el modelado de datos de MongoDB para Node.js. El modelado de datos define los datos que pueden estar y los que deben de estar en un documento de MongoDB, así como sus tipos y si son únicos, definidos en el esquema. Mongoose también añade una capa de funciones útiles en la construcción de aplicaciones web, como una gestión de conexiones con la base de datos más sencilla, validación a nivel de esquema (Holmes, 2019)…

2.3.2.1.1.2. Node.js

Node.js es una plataforma web que permite crear un servidor y construir aplicaciones encima de él. No es en sí mismo el servidor ni un lenguaje. Contiene una biblioteca de servidor HTTP integrada que permite ejecutar el servidor sin depender de otros programas como NGINX y Apache, otorgando más control sobre cómo funciona el servidor, costando un aumento de la complejidad en la configuración del servidor (Holmes, 2019).

Su popularidad está aumentando gracias a que se puede programar en JavaScript, un lenguaje que ya era usado ampliamente para el frontend, eliminando la necesidad de saber PHP o Ruby para programar el backend si uno desea ser desarrollador full-stack. Sin embargo, hay una curva de aprendizaje en Node.js, incluso siendo un experimentado

programador frontend de JavaScript, debido a que hay retos y obstáculos distintos que afrontar en la programación del servidor que en la del frontend, primando el correcto flujo del código para que no se cuelgue la página y optimizar recursos (Holmes, 2019).

Otra ventaja de Node.js es que, cuando se programa adecuadamente, es muy rápido y eficiente, permitiendo servir a más usuarios con menor recursos comparados con otros servidores tradicionales. Node.js apenas impacta en los recursos del sistema al usar un simple hilo de proceso, mientras que los otros servidores suelen usar varios hilos.

Estos servidores son poco escalables, usando más potencia y RAM de la que suelen necesitar, para prevenir que la página sea lenta cuando haya muchos usuarios concurrentes. Esto se debe a que deben asignar un hilo por cada usuario concurrente. Un servidor que se ejecuta sobre un solo hilo, todos los usuarios tienen que usar el mismo hilo de proceso, el cual solo se encarga de responder o realizar peticiones, delegando a otros procesos tareas más complejas. Gracias a este enfoque, un servidor de Node.js es más escalable y eficiente en recursos que los de múltiples hilos. Además, este enfoque es posible por las características asíncronas de JavaScript (Holmes, 2019).

Node.js cuenta también con un gestor de paquetes llamado *npm*, el cual permite la descarga e instalación de módulos que extienden las funcionalidades de Node.js en una instancia. A fecha de la publicación de la segunda edición del libro sobre la pila MEAN de Holmes (2019), npm cuenta con más de 350.000 paquetes.

2.3.2.1.2. JSON Web Token: Seguridad

Node.js y Express no guardan información de cada sesión en el servidor, por lo que tendríamos que buscar un enfoque alternativo para la autenticación de los usuarios. En las APIs en las que se usa la pila MEAN, suele ser común el uso de JSON Web Token (JWT) como solución para la autenticación de ciertos usuarios, el cual es un objeto JSON cifrado en una cadena que puede decodificar tanto cliente como servidor. El proceso de login consistiría en enviar al servidor las credenciales de un usuario, las cuales validaría y, si coinciden con las almacenadas, se devuelve un token, el cual almacenaría el navegador para reutilizarlo. Como se ha mencionado anteriormente, una API REST no guarda estado, por lo que no sabe quién está llamando a un método protegido. Para saber la API si un usuario está autorizado a realizar cierta operación o no, el token se envía usando un *middleware*, el cual decodifica el token y realiza esa comprobación (Torres Gil, 2019). Además, estos tokens pueden ser perecederos, de forma que, tras el tiempo establecido, estos tokens no servirían para poder autenticar al usuario.

Un token JWT se compone de tres partes (Holmes, 2019):

- **Cabecera:** Un objeto JSON codificado el cual contiene el tipo y algoritmo de hashing usado.
- **Payload:** Un objeto JSON codificado que contiene los datos, siendo este el cuerpo del token.
- **Firma:** Un hash encriptado de la cabecera y el payload, usando un secreto que sólo sabe el servidor original.

Un token JWT es una cadena que se compone de estas tres partes, separadas por un punto entre ellas. Un ejemplo real de un token sería este:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJzYW11ZWwiLCJpYXQiOjE
1OTg3MzQ4OTgsImV4cCI6MTU5ODczNTc5OH0.7jwVck4uWnd_m_aDtgOo-
sRGaqyHIB5D-beuA6Sy0Nw
```

En amarillo está subrayada la cabecera, en verde, el payload, y, en azul, la firma.

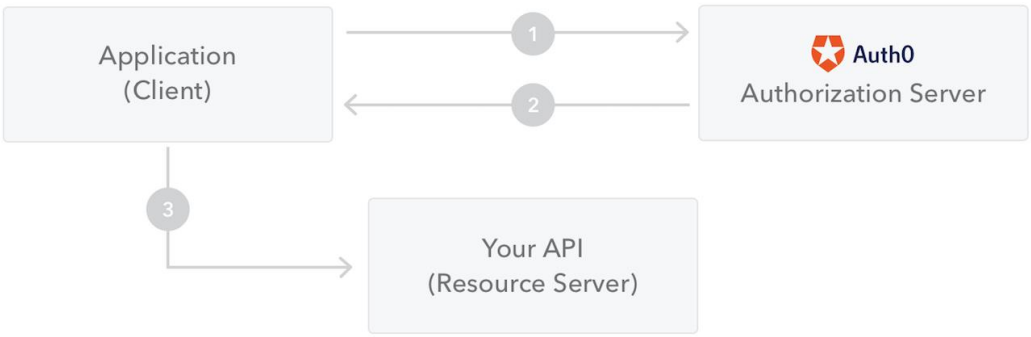

Figura 2.14: Obtención de un token con JWT (Fuente: jwt.io/introduction/).

En la anterior figura, se aclara el proceso de cómo un usuario obtiene un token para poder consumir cierto recurso de la API. Primero, la aplicación manda una petición de autorización al servidor correspondiente con las credenciales del usuario, y en el servidor, se comprueban si las credenciales son correctas. Si, efectivamente, las credenciales corresponden con las de un usuario, el servidor de autorización devuelve un token para acceder a la aplicación. El cliente usará este token para poder acceder a un recurso protegido, como algún endpoint de la API (jwt.io, s.f.).

2.3.2.2. Django

Django es un framework web escrito en Python que es de fácil aprendizaje y uso, así como muy ligero y directo (McGaw, 2009). Está basado en la arquitectura Modelo-Vista-Plantilla (MVT) y se define como un framework "con pilas incluidas", robusto y simple que pretende ayudar a los desarrolladores a escribir un código limpio, eficiente y potente. Es uno de los frameworks más famosos y usados, siendo usado por Instagram, YouTube, Google y la NASA. Al igual que Python, se basa en el patrón "Don't repeat yourself" (DRY), el cual consiste en el uso de plantillas para evitar la redundancia en el código (Nader, What is Django? Advantages and Disadvantages, 2020).

Entre sus características está el que viene con muchos paquetes por defecto (para autenticación, interfaz de administrador, gestión de sesiones…), tener una gran comunidad activa detrás, escalable, una interfaz de administrador muy intuitiva (Nader, What is Django? Advantages and Disadvantages, 2020), rápido por su uso de Python, de código abierto y una gestión de URLs sencilla (McGaw, 2009).

Además, da soporte a un gran número de bases de datos, pudiendo usar MySQL, PostgreSQL, MongoDB, SQLite…

Sin embargo, este framework tiene problemas con la facilidad para especificar URLs mediante expresiones regulares, no rinde muy bien en proyectos pequeños al contener muchos paquetes, los fallos de plantillas pasan muy inadvertidos, y da una sensación monolítica, al tener que seguir una forma popular de solucionar un problema (Nader, What is Django? Advantages and Disadvantages, 2020).

2.3.2.3. Ruby on Rails

Ruby on Rails (o, simplemente, Rails) es un framework web escrito en el lenguaje de programación Ruby, basado en el paradigma Modelo-Vista-Controlador (MVC), enfocándose este framework en hacer fácil la creación rápida de un modelo de datos, construir la lógica de los controladores justo encima y poner una capa de interfaz gráfica encima de todo (St. Laurent & Dumbill, 2010). Es un framework muy versátil que puede usarse tanto como un servidor completo con una gran capacidad de representar páginas usando plantillas como de ser consumido como una API RESTful (Nader, Getting Started With Rails, 2020).

Fue creado en 2004 por David Heinemeier Hansson, y fue lanzado usando la licencia MIT (Katz, Klabnik, Bigg, & Skinner, 2015), y desde entonces, ha sido una de las herramientas más populares para el desarrollo web, siendo usada por Airbnb, SoundCloud, Hulu, GitHub y Shopify, así como por muchas startups y autónomos (Hartl, 2020).

Como ventajas, cuenta con ser un framework que permite un desarrollo full-stack, algo muy interesante para aquellos que comienzan en el desarrollo web, destacando su enfoque integrado para la creación de aplicaciones web, así como su potencia y elegancia (Katz, Klabnik, Bigg, & Skinner, 2015). Otra ventaja sería que es fácil de aprender en un principio, gracias al dinamismo y simplicidad que caracteriza al lenguaje Ruby, el cual se asemeja a Python. Además, la documentación oficial ayuda a realizar aplicaciones completas. El problema es que, aunque es fácil de aprender lo básico, tiene una curva de aprendizaje muy empinada, siendo más complicado aprender cosas más avanzadas (Nader, Getting Started With Rails, 2020).

Ruby on Rails se enfoca también en ser un framework que haga muy productivo el trabajo de los desarrolladores web con respecto a otros frameworks, ya que con sus paradigmas “Don’t Repeat Yourself” como “Convention over Configuration” (con la que el desarrollador solo tiene que especificar aspectos no convencionales de la aplicación, realizando la configuración automáticamente) y que las aplicaciones web desarrolladas con Ruby on Rails se parecen mucho (Katz, Klabnik, Bigg, & Skinner, 2015), hacen que el proceso de desarrollo sea muy rápido, pudiendo desplegarlas en poco tiempo. Sin embargo, esto hace que sea un framework que da pocas alternativas al desarrollador a la hora de resolver un problema, de manera que solo hay una forma recomendada de resolverlo (Getting Started with Rails, s.f.), además de que el despliegue de una app web Rails no es fácil, ya que no suele tener tanto soporte que haga que esté listo para usar, teniendo que usar alguna plataforma como Docker para desplegar fácilmente la aplicación en un contenedor (Nader, Getting Started With Rails, 2020).

Rails incluye por defecto muchas aplicaciones como soluciones de seguridad (las cuales ayudan a los novatos a no cometer errores catastróficos), de generación de código, una rica capa ORM (Object Relational Mapping) y mecanismos de autenticación, además de contar con *gemas*, que son paquetes que añaden funciones a una app web Ruby on Rails que ayudan a no alargar el desarrollo de una app web, siendo creadas y mantenidas por una gran comunidad (Katz, Klabnik, Bigg, & Skinner, 2015). Un inconveniente de esto último es que las gemas y el framework hacen cambios muy importantes cada poco tiempo, obligando en cierta manera al desarrollador a estar al tanto de novedades (Rak, s.f.).

Otro inconveniente es el rendimiento que da este framework, sobre todo si se cometen errores, los cuales provocan mayores tiempos de espera. Node.js es más rápido en este sentido que Rails, siendo necesaria hacer una buena optimización de la aplicación (Rak, s.f.).

2.3.2.4. Phalcon

Phalcon es un framework open source full-stack para PHP. Aunque está escrito en C, esto no significa que los desarrolladores que se planteen usarlo necesiten conocer este lenguaje, ya

que sería como una extensión de PHP escrita en C, gracias a lo cual consigue un rendimiento considerablemente alto (Phalcon, s.f.).

Como características de este framework, tenemos la baja sobrecarga por el bajo consumo de recursos al usar programas en C, los cuales son compilados previamente (no interpretados). Gracias a sus optimizaciones y arquitectura de bajo nivel, este framework tiene una de las cargas más bajas en cuanto a frameworks basados en MVC (Modelo-Vista-Controlador). También se puede mencionar un gestor de eventos, que puede configurar oyentes para ejecutar un código al ocurrir cierto evento (Phalcon, s.f.).

Phalcon permite escribir APIs REST de forma sencilla, contando con un potente conjunto de ayudantes HTTP, así como capacidades avanzadas de enrutamiento. Gracias a su ORM, puede manipular registros en bases de datos como clases y objetos, soportando MySQL, PostgreSQL y SQLite. Las consultas a las bases de datos las realiza usando PHQL, el cual analiza una consulta SQL y la traduce al RDBMS usado. Además, se asegura de que las transacciones se realicen con éxito y mantener, de esa forma, la integridad de los datos. Asimismo, cuenta con un componente caché que mantiene en memoria los datos más frecuentemente usados o ya procesados, para mantener ese alto rendimiento que caracteriza a este framework. Permite también el fragmentado de bases de datos, pudiendo interactuar con varias bases de datos al mismo tiempo. (Phalcon, s.f.).

Cuenta con dos motores de plantillas, uno con el que se renderizan plantillas escritas en HTML con código PHP incrustado y otro usando el lenguaje de plantillas Volt, el cual es muy rápido y escrito en C. Contiene también componente multilenguaje, un constructor de formularios HTML y un notificador Flash para mandar mensaje al usuario (Phalcon, s.f.).

Como características de seguridad, tenemos la posibilidad de crear Listas de Control de Acceso (ACL) y servicio de encriptado mediante el componente Phalcon/Crypt, el cual ofrece capas simples de la biblioteca de cifrado PHP openssl (Phalcon, s.f.).

No todo lo que reluce es oro, aunque este framework suene muy prometedor, cuenta con muy pocos recursos y literatura, así como con una comunidad muy pequeña en comparación con otros frameworks. Además, es necesario saber C para depurar código (B., 2018), y puede ser un proceso más complicado que en otros framworks.

2.3.2.5. Laravel

Laravel es un framework MVC para PHP para desarrollo web, el cual es de código abierto, intuitivo, fácil de gestionar y escalable. El núcleo de la actitud de Laravel está en la creencia de que el proceso de desarrollo tiene que ser tan fácil y disfrutable como un uso de alta calidad de la aplicación. Para este fin, este framework está lleno de características que facilitan el desarrollo y que permite a los expertos enfocarse en los fundamentos en lugar de distraerse escribiendo código desde cero (DDI Development, s.f.).

Laravel se caracteriza por su modularidad, conteniendo más de 20 librerías por defecto, testabilidad, enrutamiento muy flexible, gestión de la configuración según el equipo en el que esté montada la aplicación, un constructor de consultas fluido y su ORM Eloquent (compatibles con PostgreSQL, SQLite, MySQL y SQL Server), un constructor de esquemas para definir el esquema de la base de datos en PHP, migraciones de bases de datos (almacenan información sobre cambios en esquemas), sembrado (para llenar tablas), el lenguaje de plantillas ligero Blade, facilidad para el envío de e-mails, funciones de autenticación incluidas, base de datos muy rápida Redis para almacenar información en caché, integración con distintos servicios de colas y buses de comandos y eventos (Bean, 2015). A estas características se le puede añadir una buena gestión de excepciones, ayudas

a la corrección de errores y vulnerabilidades antes del despliegue, separación entre las lógicas de negocio y presentación, variedad de recursos y paquetes, ciclos de desarrollo más cortos e ir al día con las características novedosas de cada versión de PHP (DDI Development, s.f.).

Sin embargo, este framework tiene una curva de aprendizaje empinada en etapas más avanzadas, puede generar dificultades con algunas actualizaciones, dar la sensación de ser más fácil de lo que realmente es (DDI Development, s.f.) y tiempos de respuesta más largos en comparación con otros frameworks (B., 2018).

2.3.3. Documentación de la API

Resultaría interesante ofrecer a los usuarios una documentación interactiva de la API, de forma que pueda ver cómo reacciona esta ante ciertas peticiones sin moverse de la misma página. Existen varias especificaciones conocidas para documentar una API, como las que vamos a exponer ahora:

- Swagger (ahora OpenAPI)
- RAML
- API Blueprint
- Slate

2.3.3.1. Swagger (ahora OpenAPI)

La especificación OpenAPI (anteriormente conocida como Swagger, aunque se le sigue conociendo por ese nombre y con ese nombre nos referiremos a ella) es una de las especificaciones más populares para la documentación de APIs. Desarrollada por Wordnik y mantenida hoy en día por SmartBear Software, Swagger se concibió no solo como una especificación para documentar APIs sino como un framework completo para la creación de APIs, de forma que genera tanto el código de la API, como el del cliente y la documentación de la API (Sandoval, 2015). La especificación hoy en día la lleva OpenAPI Initiative, y es abierta y mantenida por la comunidad (OpenAPI Initiative, 2020).

Esta especificación define una descripción estándar e independiente del lenguaje de programación para APIs HTTP, permitiendo a los usuarios comprender las capacidades de un servicio de forma sencilla y sin realizar una investigación más profunda. Cuando se define adecuadamente la documentación con Swagger, un consumidor de la API puede comprenderla e interactuar con ella de manera simple y rápida (OpenAPI Initiative, 2020).

Los casos de uso para una definición de una API en lenguaje de computadora incluyen, pero no se limitan a: una documentación interactiva, generación de código para la documentación, servidor y cliente, y automatización de pruebas unitarias. Los documentos de Swagger describen los servicios que una API puede ofrecer y los puede representar tanto en YAML (YAML Ain't Markup Language) como en JSON. Estos documentos pueden ser producidos y presentados estáticamente o ser generados dinámicamente desde una aplicación (OpenAPI Initiative, 2020).

Esta especificación no requiere reescribir una API en caso de querer documentar una API existente ni requiere atar el uso de cierto software a un servicio, aunque sí requiere que las capacidades de un servicio se puedan describir en la estructura de esta especificación, ya que no pretende cubrir todo estilo posible de APIs HTTP, aunque sí cubre todo lo que un API REST puede hacer. La especificación Swagger tampoco obliga a tomar cierto proceso en el desarrollo, como diseñar o codificar primero, facilitando ambas técnicas estableciendo interacciones claras con la API (OpenAPI Initiative, 2020).

Además, se han desarrollado diferentes derivaciones de esta especificación específicas para ciertos lenguajes como Java, Actionscript 3, Scala, HTML5… También está diseñada para ser

una especificación ascendente, al contrario que otras especificaciones para APIs, de forma que especifica el comportamiento que afecta a la API apostando por la creación de sistemas más complejos (Sandoval, 2015).

Como ventajas, tenemos que es una especificación muy popular, con una gran comunidad de usuarios y desarrolladores, y un gran soporte para múltiples lenguajes. Sin embargo, no cuenta con constructos avanzados para los metadatos (Sandoval, 2015).

2.3.3.2. RAML

RAML (RESTful API Modeling Language) es un lenguaje para definir APIs basadas en HTTP que encarna muchos o todos los principios de REST, siendo una aplicación de la especificación YAML 1.2. RAML provee mecanismos para definir APIs RESTful, generar código fuente para servidor/cliente y documentar comprensivamente para los usuarios una API (RAML, 2019). El que RAML personifique muchos principios de REST no quiere decir que no soporte el documentar APIs que usen otros estilos arquitectónicos como SOAP o RPC (Sandoval, 2015).

RAML está diseñado para mejorar la especificación de una API, proveyendo un formato que el proveedor y los consumidores puedan usar como un contrato mutuo. RAML puede, por ejemplo, facilitar el proveer documentación de usuario y puntas del código fuente para el cliente e implementaciones del servidor. Esto optimiza y mejora la definición y desarrollo de aplicaciones que usen APIs RESTful (RAML, 2019).

Esta especificación introduce un concepto innovador de tipos de recurso y atributos para caracterizar y reutilizar patrones de recursos y métodos asociados, los cuales minimizan la redundancia en el diseño de una API RESTful y fomentan la consistencia dentro y a lo largo de la API (RAML, 2019).

Esta especificación es descendente, de forma que descompone el sistema y explica el comportamiento de sus distintos componentes (Sandoval, 2015).

RAML puede usarse de varias maneras para extender su funcionalidad; dependiendo de cómo se estructure y defina, puede usarse para documentar o para hacer un plan a largo plazo de la API, aunque esto no está probado (Sandoval, 2015).

Como ventajas, se pueden destacar el soporte de constructos avanzados, una adopción decente, un formato legible para los humanos y un fuerte apoyo de la industria (por ejemplo, la API de Spotify está documentada con esta especificación), aunque como contra tiene carecer de herramientas a nivel de código (Sandoval, 2015).

2.3.3.3. API Blueprint

API Blueprint es una especificación que utiliza Markdown para sus archivos y descendente (Sandoval, 2015), desarrollada por la empresa californiana Apiary. Es simple y accesible para todos aquellos implicados en el ciclo de vida de las APIs, con una sintaxis concisa y expresiva, con el que se pueden diseñar y prototipar APIs a crear o documentar y probarlas una vez desplegadas (Apiary, s.f.).

Es una especificación transparente y abierta, enfocada en la colaboración, proveyendo herramientas para alcanzar las metas propuestas con la API y animar a las distintas partes implicadas en el desarrollo de la API al diálogo y colaboración. Asimismo, también está diseñado para fomentar un mejor diseño de APIs a través de abstracciones, y para diseñar la API antes de desarrollarla (Apiary, s.f.).

Aunque sea una especificación sencilla de entender y escribir, carece de constructos avanzados y tiene una instalación compleja, por no mencionar que tiene una adopción muy escasa con respecto a Swagger y RAML (Sandoval, 2015).

2.3.3.4. Slate

Slate es un generador de documentación para APIs que ayuda a crear estas de manera que sean visualmente atractivas, inteligente y con respuesta. Sus documentos se escriben en Markdown, soportando muestras de código en múltiples lenguajes y con resalto de sintaxis en más de 100 lenguajes (slatedocs, s.f.).

La página generada por Slate contiene todo lo relacionado con la documentación de una API en la misma página, sin ser necesario moverse por varias páginas para alcanzar el contenido deseado. Las muestras de código se encuentran en la parte derecha de la página y, en la izquierda, se encuentra una barra de contenido que se desplaza de forma automática y suave que muestra en qué parte de la documentación nos encontramos. Slate cuenta con soporte para lenguajes que se escriben de derecha a izquierda como el árabe. Por defecto, la documentación de una API en Slate se guarda públicamente en GitHub, permitiendo a otros usuarios realizar cambios, aunque se pueden alojar en cualquier otro sitio (slatedocs, s.f.).

Aunque genere una página muy atractiva con la documentación, tiene muchos enfoques que no han sido probados y una base de usuarios pequeña.

3. Desarrollo de la Propuesta de Trabajo Fin de Grado

En este capítulo, se desarrollará el proyecto técnico como tal, y se seleccionarán las herramientas con el que se implementará.

3.1. Elección de las Herramientas a usar

Tras haber visto las distintas alternativas en cuanto a portales de datos y frameworks para el desarrollo de la API, así como la especificación para su documentación, procederemos a la elección de las distintas herramientas que usaremos en el proyecto técnico.

3.1.1. Elección del portal de datos abiertos

Para la elección del portal de datos, se tendrán en cuenta los siguientes criterios:

- Gratuidad del software.
- Posibilidad de enlazar a datos remotos.
- Previsualización de datos tabulares y geoespaciales.
- Acceso a conjuntos con cierta granularidad.
- Mantenimiento y configuración de una simple instancia local realizable por una persona.
- Posibilidad de tener el portal operativo en un día de trabajo.

Teniendo en cuenta los anteriores criterios, la conclusión a la que se llega es que el portal de datos que se usará de cara al proyecto será **CKAN**.

A diferencia de soluciones SaaS como OpenDataSoft o Socrata, CKAN es un software totalmente gratuito y sin demos limitadas (aunque una demo limitada de uno de estos podría servir para el número y tamaño de los conjuntos de datos que se usarán).

La otra alternativa, que es construir un portal de datos abiertos desde cero, sería inviable en el sentido de que se invertiría demasiado tiempo en desarrollar una solución muy parecida a otras que hay en el mercado, por lo que quedaría descartada desde un principio.

Resulta esencial por la naturaleza de este TFG que el portal de datos que se construya soporte conjuntos de datos que están almacenados en un servidor remoto, por lo que soluciones en las que no se puede enlazar conjuntos de datos a archivos remotos tendrían que descartarse, como Junar y ArcGIS Open Data, estando totalmente seguros de que este requisito lo cumplen CKAN y OpenDataSoft. Socrata, si bien permite el acceso a recursos remotos, no previsualiza su contenido.

Una ventaja que tiene CKAN sobre las otras alternativas es que, al ser de código abierto y con una arquitectura modular, es que es muy fácil de extender sus funcionalidades. De hecho, la función de almacenar archivos la ofrece una extensión, al igual que las funciones de previsualización de datos geoespaciales o PDFs. Además, tiene detrás una comunidad de desarrolladores que se encarga de actualizar el software periódicamente.

Otro punto que ha resultado útil para tomar una decisión es que CKAN da más juego a un técnico para determinar las funcionalidades del portal y gestionarlo, ya que todo eso se encarga de hacerlo un equipo especializado de una empresa que ofrezca un servicio SaaS. La apertura del portal de datos en un servidor público es algo que se escapa de los objetivos de este TFG, ya que trata sobre cómo crear un portal de datos abiertos y cómo acceder desde este a un recurso remoto mediante una API REST, y eso se puede realizar desde un equipo local. Además, como se verá más tarde, la instalación del portal en local no es algo que requiera ni mucho menos un equipo de técnicos especializados detrás de él.

3.1.2. Elección del framework para el desarrollo de la API

A la hora de elegir un framework para la API, tendremos en cuenta lo siguiente:

- Buen rendimiento para un proyecto pequeño.
- Bajo consumo de recursos.
- Framework que sea muy utilizado, de forma que tenga una gran comunidad alrededor de él y bastante literatura al respecto.
- Sencillo para un novato.
- Modularidad.
- Que no sea muy inflexible a la hora de solucionar problemas.

Al plantear una API muy sencilla con pocos endpoints, y siendo un proyecto pequeño, Django no sería la mejor opción, ya que, como se ha visto antes, este framework no tiene un buen rendimiento con proyectos pequeños, esto debido a su paradigma de ser un framework "con pilas incluidas", es decir, que contiene muchos paquetes que hacen que Django pueda realizar por defecto una gran cantidad de funciones. Sin embargo, estos paquetes cargarían mucho la ejecución inútilmente, puesto que no van a ser usados.

Puede que Phalcon tenga un rendimiento que lo haga ser considerado un framework prometedor sin embargo, no goza de la literatura y la comunidad que sí tienen los otros frameworks vistos, así como la corrección de errores se hace más complicada que en otros, pudiendo no resultar muy agradable para alguien que se esté iniciando en el desarrollo web.

Tras eliminar Django y Phalcon, nos quedaríamos con Express, Ruby on Rails y Laravel. Teniendo en cuenta los criterios, resultaría preferible usar **Express**. Express+Node.js rinde mejor que Rails y Laravel, ya que no tiene módulos que no se usan y consumen recursos inútilmente, lo que bajaría el rendimiento. Además, gracias a npm, se pueden añadir paquetes fácilmente a una instalación con Node.js. Y, con respecto a la flexibilidad para la solución de problemas, Express hace gala de ser una framework muy flexible en cuanto a eso, al contrario que Ruby on Rails, llevando muy al extremo su paradigma *convention over configuration*.

El uso de Express nos lleva entonces a usar la llamada pila MEAN, usando sus componentes MongoDB como base de datos, Express como framework para el backend y Node.js como aplicación web en el servidor. Angular no se usaría, ya que nuestra aplicación web no necesita un frontend, porque desde el portal de datos se realizarán las consultas estableciendo cierto filtro o no sobre el conjunto de datos, siendo devueltas estas consultas en formato JSON, sin necesidad de presentar estas en una aplicación web.

3.1.3. Elección de la especificación para documentar la API.

Para la documentación de la API, tendremos en cuenta los siguientes factores:

- Gran soporte por parte de la comunidad.
- Que permita la documentación de una API tras su implementación.
- Que realice interacciones con la API.
- Popular, con una gran adopción y cantidad de recursos en la Web.

API Blueprint y Slate quedarían descartados al ser especificaciones que no tienen una adopción generalizada por parte de la comunidad. De esa forma, nos quedaría Swagger y RAML.

En este caso, usaremos **Swagger**, debido a que es más popular y a que RAML carece de una documentación fuerte y tutoriales aparte de la especificación. Además, se puede integrar muy fácilmente a Node.js como veremos posteriormente.

3.2. Construcción del Portal de Datos Abiertos

En esta sección del desarrollo del trabajo, cubriremos la construcción del portal de datos abiertos, desde su instalación hasta su uso y sus distintos apartados.

3.2.1. Instalación y configuración del portal de datos

Aunque, a fecha en la que se están escribiendo estas palabras, ya se ha liberado la versión 2.9 de CKAN, estaremos trabajando con la versión 2.8, ya que es la que se instaló en la máquina virtual en la que se realiza el proyecto técnico.

Para realizar el proyecto técnico, se usará una máquina virtual VMware con el sistema operativo Ubuntu 16.04 LTS en su versión de 64 bits, con 8 GB de RAM y 4 núcleos del procesador asignados. La versión de Ubuntu utilizada es antigua (data de hace 4 años), pero se utiliza esa debido a que esa es la que requiere CKAN para poder instalar desde paquete.

Siguiendo la guía de instalación de CKAN, se puede ver que se puede instalar de tres maneras: desde un paquete, desde el código fuente y desde Docker Compose. Se ha preferido una instalación desde paquete debido a que desde el código fuente en una versión más actual de Ubuntu sería más complicado, y sobre la instalación usando Docker no se tenían suficientes conocimientos sobre este gestor de contenedores, aunque también podía resultar una opción interesante.

Para la instalación desde paquete de CKAN, se requiere tener cierto software ya instalado y configurado, siendo necesario realizar la instalación con el comando apt-get install de los siguientes paquetes:

- Servidor HTTP Apache *(apache2)*
- Módulo adaptador WSGI (Python) para Apache *(libapache2-mod-wsgi)*
- Biblioteca en C del cliente PostgreSQL *(libpq5)*
- Servidor Redis (motor de base de datos para claves con interfaz de red) *(redis-server)*
- Control de versiones Git *(git-core)*
- Servidor web/proxy ligero Nginx *(nginx)*

Todo ello con el siguiente comando;

```
sudo apt-get install -y nginx apache2 libapache2-mod-wsgi libpq5 redis-server git-core.
```

Lo siguiente es descargar el paquete adecuado de CKAN e instalarlo con:

```
wget http://packaging.ckan.org/python-ckan_2.9-xenial_amd64.deb
sudo dpkg -i python-ckan_2.9-xenial_amd64.deb.
```

Tras instalar el paquete de CKAN, es necesario instalar el gestor de bases de datos PostgreSQL y configurarlo para CKAN. Se instala el paquete

```
sudo apt-get install -y postgresql
```

, se crea usuario con introduciendo posteriormente la contraseña

```
sudo -u postgres createuser -S -D -R -P ckan_default
```

y se crea la base de datos para CKAN.

```
sudo -u postgres createdb -O ckan_default -E utf-8
```

Una vez configurado PostgreSQL, modificamos la propiedad sqlalchemy.url en el archivo de configuración de CKAN (/etc/ckan/default/production.ini), introduciendo la contraseña, base de datos y usuario de la base de datos.

Otro software necesario para CKAN es motor de búsqueda Apache Solr, instalándolo con el comando:

```
sudo apt-get install -y solr-jetty
```

y configurando su archivo de configuración (/etc/default/jetty8), cambiando las variables `NO_START=0`, `JETTY_HOST=localhost` y `JETTY_PORT=8983`. Se reinicia Solr.

```
sudo service jetty8 restart.
```

Cambiamos el archivo schema.xml por el de CKAN.

```
sudo mv /etc/solr/conf/schema.xml /etc/solr/conf/schema.xml.bak
sudo ln -s /usr/lib/ckan/default/src/ckan/ckan/config/solr/schema.xml
/etc/solr/conf/schema.xml
```

, y reiniciamos Solr otra vez. Para terminar con la configuración de Solr, cambiamos la variable solr_url con la URL que usa nuestro servidor de Solr en el archivo de configuración de CKAN.

Faltaría modificar el archivo de configuración de CKAN con la URL del portal de datos (ckan.site_url), la cual es en este caso http://localhost. Se arranca la base de datos.

```
sudo ckan db init
```

Al no necesitar en un principio la subida de archivos (ya que accederemos a recursos remotos), no es necesario configurar las extensiones DataStore y FileStore.

Para finalizar con el proceso de instalación, reiniciamos Apache y Nginx

```
sudo service apache2 restart
sudo service nginx restart
```

, y accedemos al portal de datos con la URL configurada anteriormente.

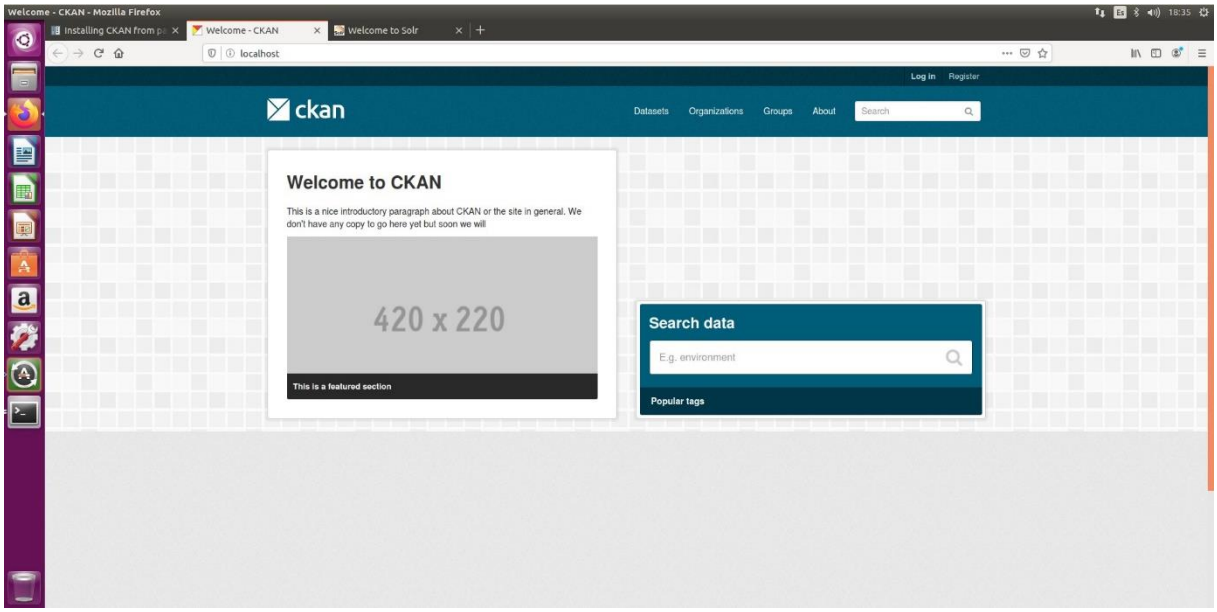

Figura 3.1: Portal CKAN tras el término de la instalación.

Se pueden hacer los siguientes cambios en el archivo de configuración para adecuar ciertas características del portal. En este caso, se han hecho los siguientes cambios en el archivo de configuración:

- **Creación de grupos por parte del usuario:** se habilitará para permitir a un usuario crear grupos *(ckan.auth.user_create_groups = true).*
- **Creación de organizaciones por parte del usuario:** se habilitará para permitir a un usuario crear organizaciones *(ckan.auth.user_create_organizations = true).*
- **Título del sitio:** lo cambiaremos a "CKAN - TFG Samuel Rodríguez Simón" *(ckan.site_title = CKAN - TFG Samuel Rodríguez Simón).*
- **Descripción del sitio:** *(ckan.site_description = Portal de datos abiertos para la realización del TFG.).*
- **Configuración regional por defecto:** cambiamos a es para tener por defecto el portal en español *(ckan.locale_default = es).*
- **Orden de configuraciones regionales:** pondremos en primer lugar el español, seguido por el inglés y el francés *(ckan.locale_order = es en fr pt_BR ja it cs_CZ ca el sv sr sr@latin no sk fi ru de pl nl bg ko_KR hu sa sl lv).*

3.2.2. Uso del portal de datos

En esta sección, explicaremos el uso que se hace de un portal CKAN siendo un usuario, apoyándonos en la guía de usuario de CKAN (s.f.).

Primero hablaremos sobre los conjuntos de datos y los recursos, que son las unidades en las que se organizan los datos en CKAN. Los conjuntos de datos son paquetes de datos, por ejemplo, pueden ser estadísticas de crimen por región o las temperaturas por estación. Cuando los usuarios buscan conjuntos de datos, los resultados los llevarán a conjuntos de datos individuales.

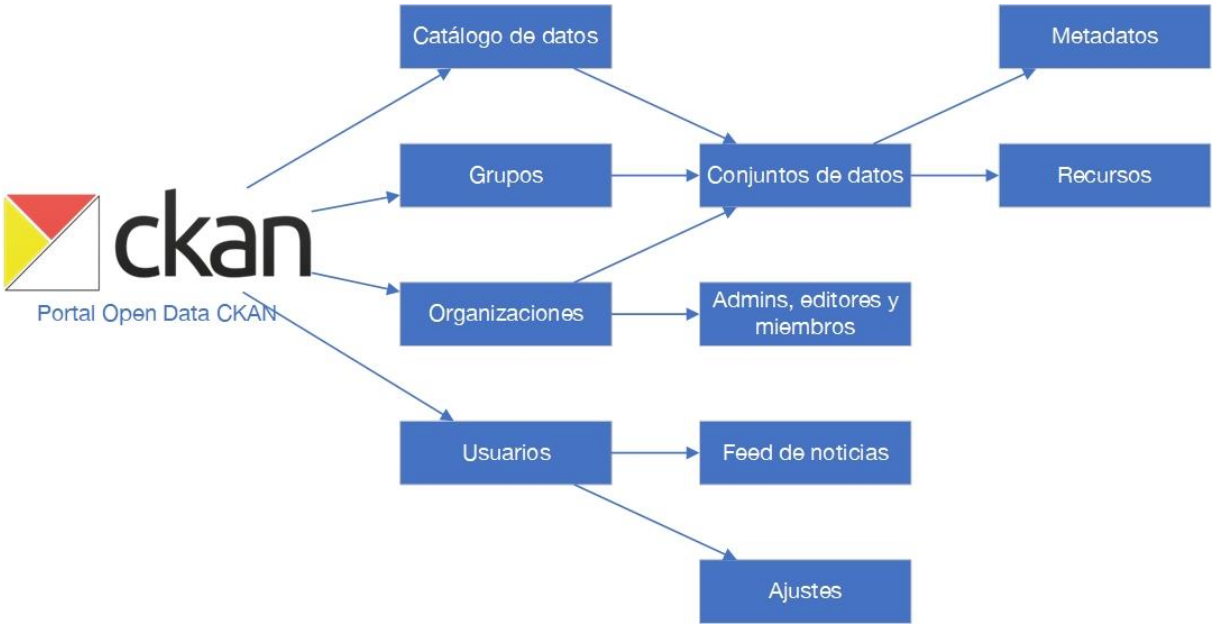

Figura 3.2: CKAN y cómo se organizan sus distintos componentes.

Los conjuntos de datos contienen dos cosas:

- **Metadatos:** por ejemplo, título y editor, fecha, los formatos en los que está disponible el conjunto de datos, bajo qué licencia está…

- **Recursos**, los cuales contienen los datos. CKAN no se preocupa de en qué formato están los datos, pudiendo estar en cualquiera. CKAN puede almacenar los recursos internamente o, simplemente, como un enlace, estando el recurso en cualquier otro sitio de la web. Un conjunto de datos puede contener cualquier número de recursos, por ejemplo, pueden contener los datos por año, pero también los mismos datos en diferentes formatos.

Un portal de CKAN también se compone de usuarios y organizaciones. Los usuarios se registran en el portal y, con esa cuenta, pueden iniciar sesión. No es necesaria una cuenta para poder buscar datos, pero sí para poder operar con los conjuntos de datos. Normalmente, cada conjunto de datos pertenece a una "organización", las cuales se gestionan como ellas vean oportuno. Estas organizaciones pueden ser, por ejemplo, las consejerías de un gobierno autonómico. Estas organizaciones tienen administradores que pueden añadir a otros usuarios a la organización y otorgarles roles con ciertos privilegios como la creación de conjuntos de datos. En una configuración por defecto, los conjuntos de datos se crean como privados, pudiendo ser publicados pulsando un botón que puede requerir un nivel de privilegios más alto. Normalmente, es necesario que se creen los conjuntos de datos dentro de una organización, aunque, como en este caso, es posible configurar el portal para permitir la creación de conjuntos que no pertenecen a ninguna organización.

3.2.2.1. Conjuntos de datos y recursos

En CKAN, la página de un conjunto de datos incluye nombre, descripción y otros datos de interés, así como enlaces y breves descripciones sobre sus recursos. Además, en esta página hay otras dos pestañas: secuencia de actividades (para ver los cambios recientes sobre el conjunto de datos) y grupos (para ver los grupos asociados a este conjunto de datos).

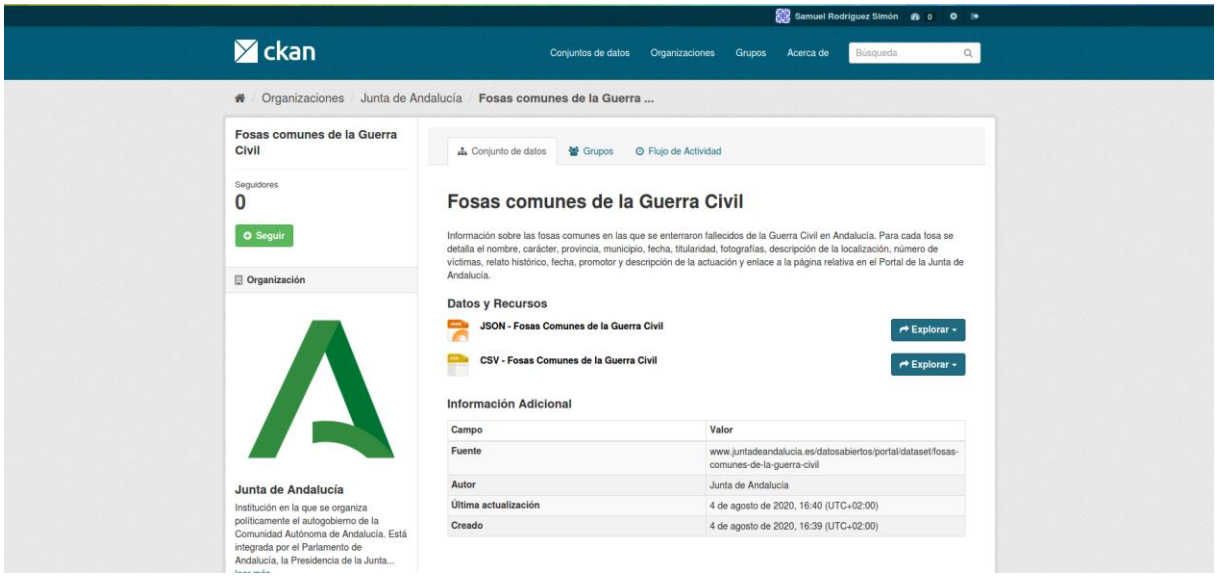

Figura 3.3: Página de un conjunto de datos en CKAN.

Cada recurso también tiene su propia página, con información sobre él, la posibilidad de descargarlo y una previsualización según el formato del recurso (una tabla en caso de una hoja de cálculo, un mapa para datos geoespaciales, imágenes, PDFs o HTMLs).

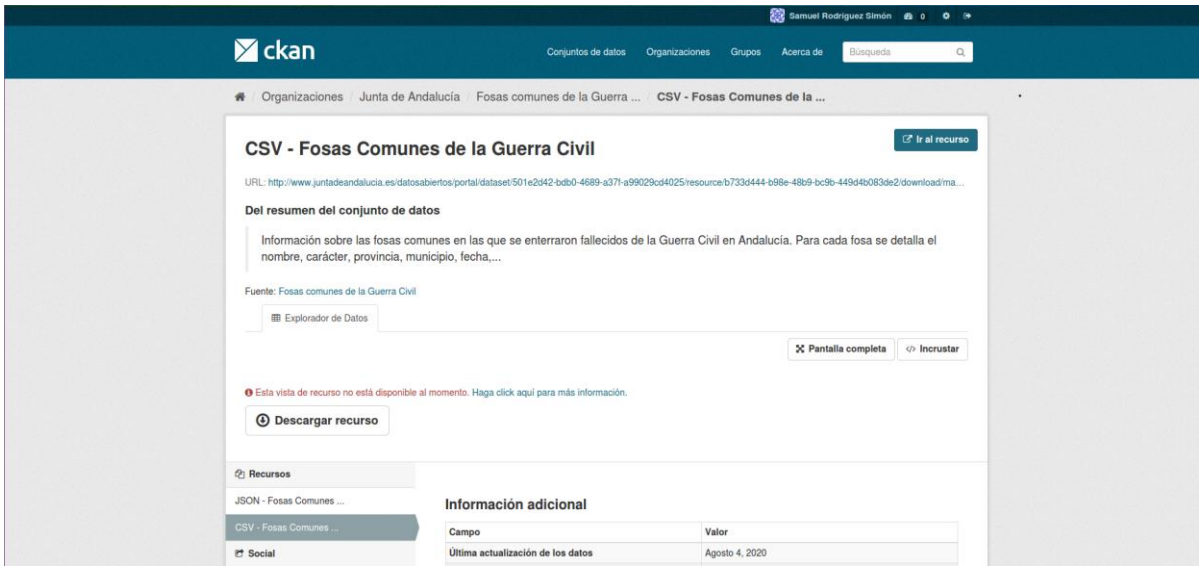

Figura 3.4: Página de un recurso en un conjunto de datos.

Para añadir un conjunto de datos, podemos hacerlo desde un botón situado tanto en la página de “Conjuntos de datos” como en una organización a la que pertenezca el usuario. CKAN pide los siguientes datos para crear el conjunto de datos: título, descripción, etiquetas, licencia y organización, aunque también se pueden añadir opcionalmente metadatos sobre el autor. Lo siguiente que se pide son los datos para crear el primer recurso del conjunto de datos. Se dan tres opciones para añadir los datos: enlazar a un archivo, enlazar a una API o subir el archivo. También se pide título, descripción y formato. Después se da la opción de guardar el recurso y crear el siguiente o terminar de crear el conjunto de datos, pudiendo añadir o modificar los recursos más en adelante.

Para editar el conjunto de datos, se haría pulsando el botón editar en la página del conjunto de datos, llevando a una página que permite cambiar los metadatos del conjunto, pudiendo modificar los metadatos y la privacidad. Para borrarlo, se haría pulsando el correspondiente botón en la página del conjunto de datos.

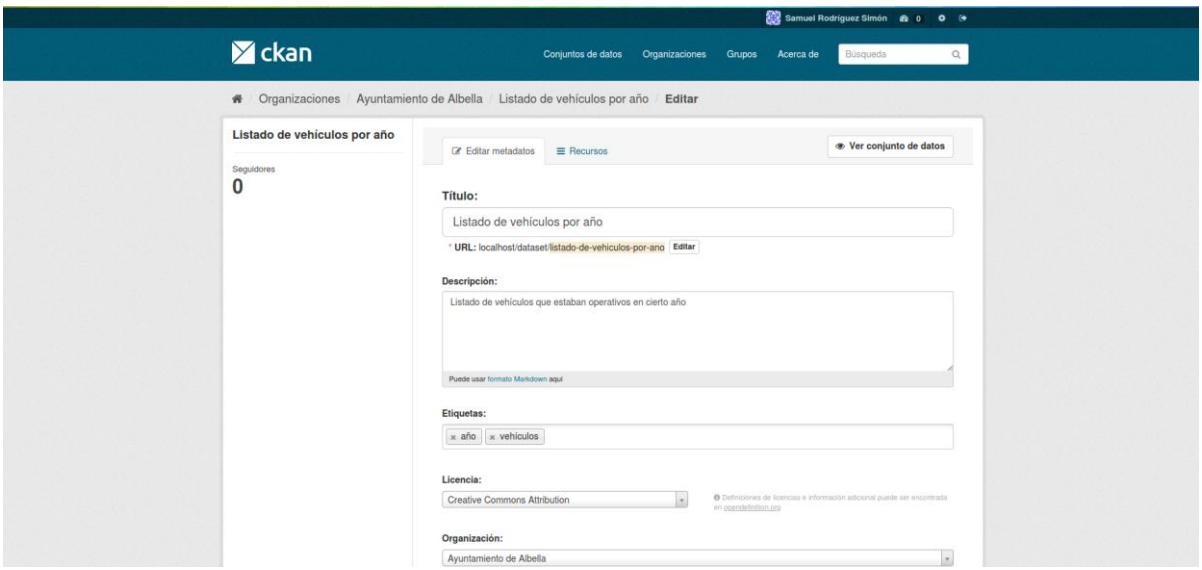

Figura 3.5: Página de edición de un conjunto de datos. Hay una pestaña con los recursos para poder editar su información sobre ellos.

También se le pueden añadir y eliminar recursos, así como editar los existentes. Para ello, tenemos que ir primero a editar el conjunto de datos, y en la barra izquierda, aparecen los recursos del conjunto de datos, desde donde se puede acceder a la página para modificar el recurso (también se puede acceder desde el propio recurso). En esa página, se pueden modificar los metadatos y el enlace o el archivo con los datos.

Para buscar conjuntos de datos en CKAN, o bien pueden introducirse términos en la barra de búsqueda o buscar en el catálogo de datos. Si hay un gran número de conjuntos de datos, puede ayudar el uso de filtros, como por organización o formato.

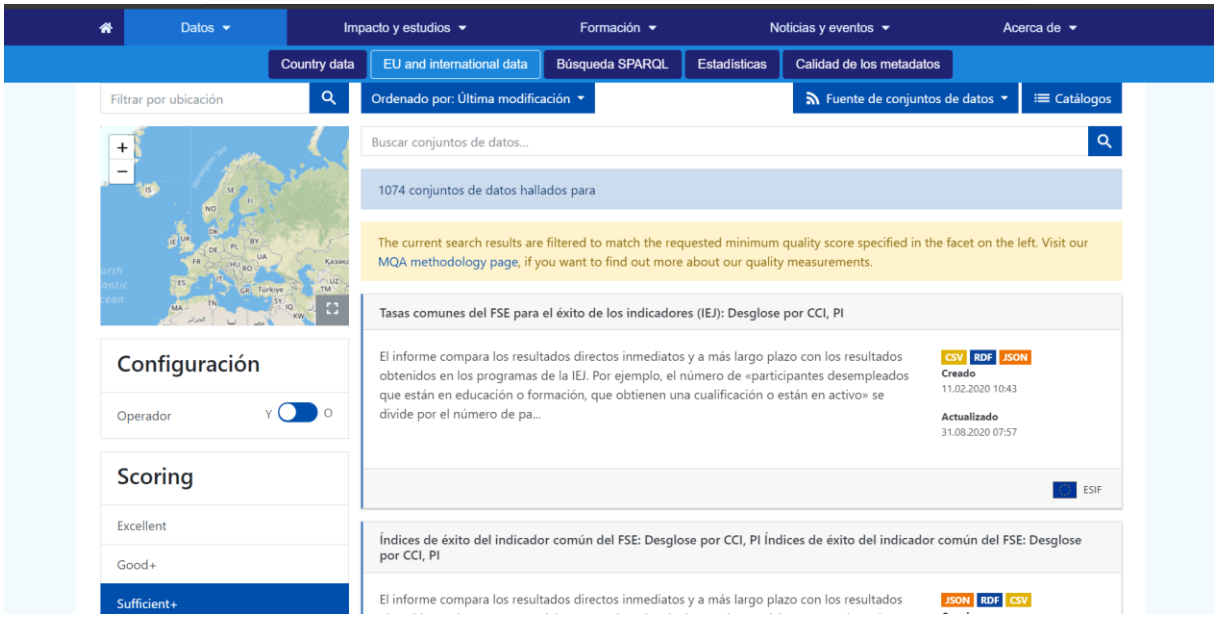

Figura 3.6: Catálogo de datos del Portal Europeo de Datos (Fuente: https://www.europeandataportal.eu/data/eu-international-datasets).

3.2.2.2. Organizaciones

Los conjuntos de datos, normalmente, pertenecen a organizaciones, las cuales tienen también usuarios con distintos niveles de privilegios. Cada organización tiene su página en la cual se puede buscar entre sus conjuntos de datos y muestra información.

Para crear una organización, se iría al apartado de “Organizaciones” y se pulsaría el botón de “Crear organización”. Se abre una página en la cual se introducen los datos de la organización, como nombre (obligatorio), descripción y un URL para una imagen. Tras crear la organización, se muestra la página de esta y aún no tendría conjuntos de datos. Además, el creador sería el administrador de la organización.

Un administrador tiene opciones dentro de la organización para gestionarla pulsando el botón “Admin”. Esta lleva a una página para el administrador con dos pestañas:

- **Información:** en esta se podría editar la información de la organización.
- **Miembros:** permite añadir, borrar y cambiar los roles para los usuarios dentro de la organización. Por defecto, CKAN permite tres roles:
  - **Miembro:** puede ver los conjuntos de datos privados de la organización.
  - **Editor:** puede publicar y editar conjuntos de datos.
  - **Administrador:** puede añadir y eliminar miembros, aparte de cambiarles el rol.

Para explorar los conjuntos de datos pertenecientes a una organización, tendríamos que introducir los términos deseados en la barra de búsqueda mientras estemos en la página de

esa organización. Se devuelven, como resultado, los conjuntos de datos acordes con esos términos restringidos a esa organización.

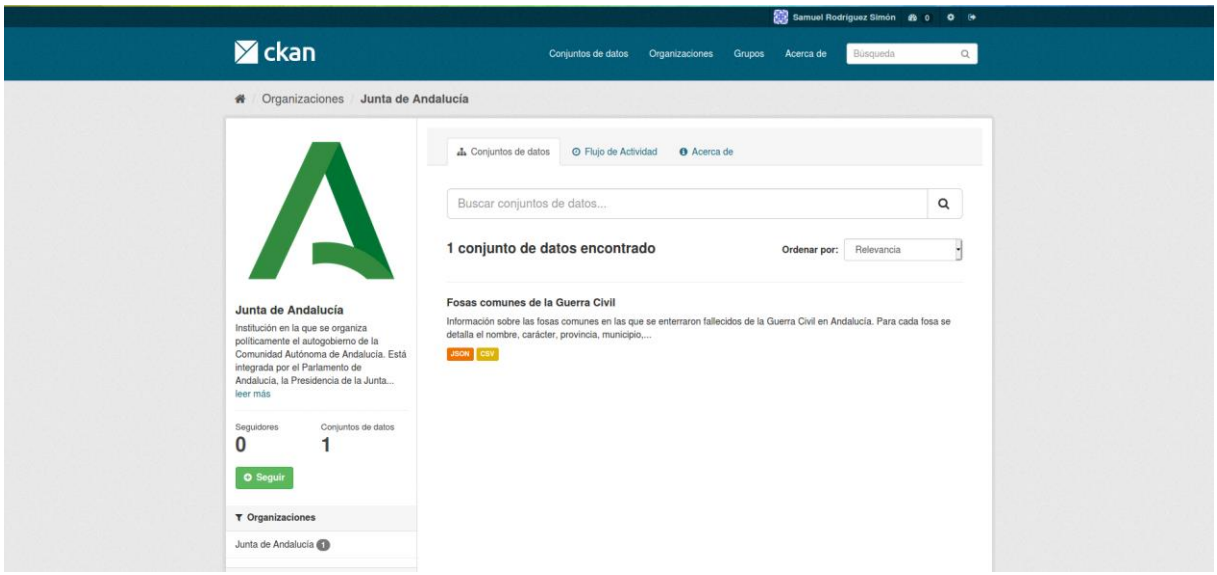

Figura 3.7: Página de una organización en CKAN.

Si un usuario registrado está interesado en los conjuntos de datos publicados por una organización, tiene la opción de seguirla, para ser notificado de la publicación de nuevos conjuntos de datos o cambios en estos.

3.2.2.3. Usuarios

Los usuarios registrados tienen ciertas opciones de personalización en un portal de datos CKAN. Además, estos tienen asociada una clave API para poder interactuar con el portal de datos vía API.

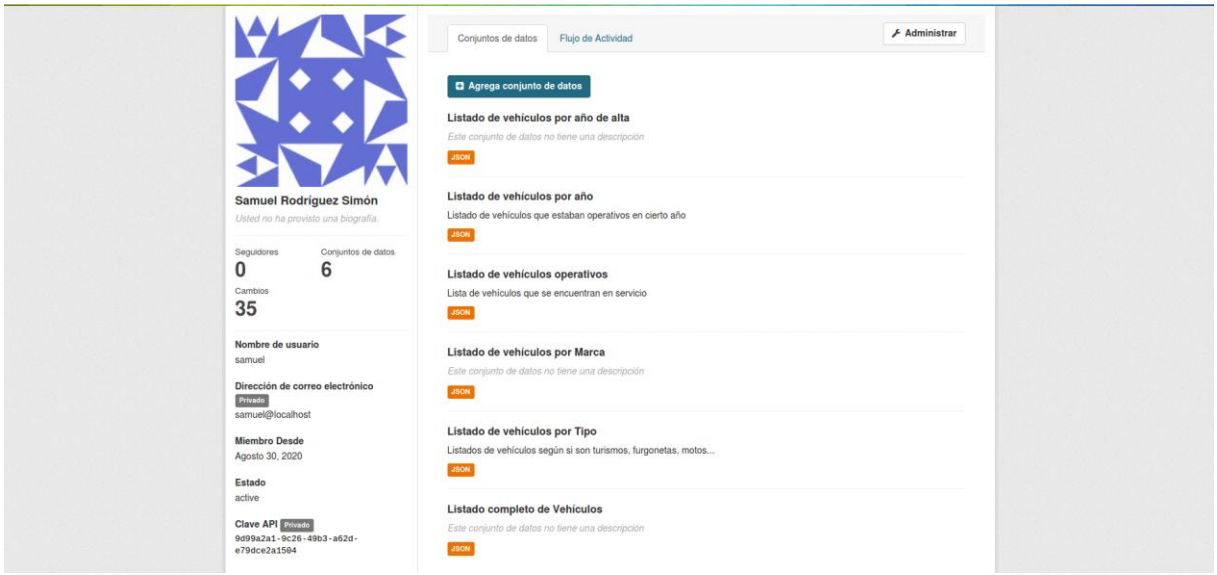

Figura 3.8: Perfil de un usuario del portal de datos CKAN.

Cada usuario tiene su tablero al cual se accede haciendo clic al icono de al lado del nombre de usuario. Ese botón también cuenta con un número, que es el número de notificaciones que tiene el usuario sin mirar en su feed de noticias. Este cuenta con dos pestañas:

- **Feed de noticias:** Muestra las publicaciones nuevas y los cambios en los conjuntos de datos seguidos y en los de los usuarios individuales y las organizaciones seguidas por el usuario. Si un usuario deja de estar interesado en seguir a alguno de los anteriores, tiene la posibilidad de dejarlos de seguir.
- **Ajustes:** permiten cambiar la información que tiene CKAN sobre el usuario, incluida aquella que pueden ver otros usuarios. Se puede acceder directamente pulsando el botón con un engranaje en la barra superior. Esta página permite cambiar nombre de usuario, nombre, dirección de correo electrónico, descripción del usuario (un texto opcional) y contraseña.

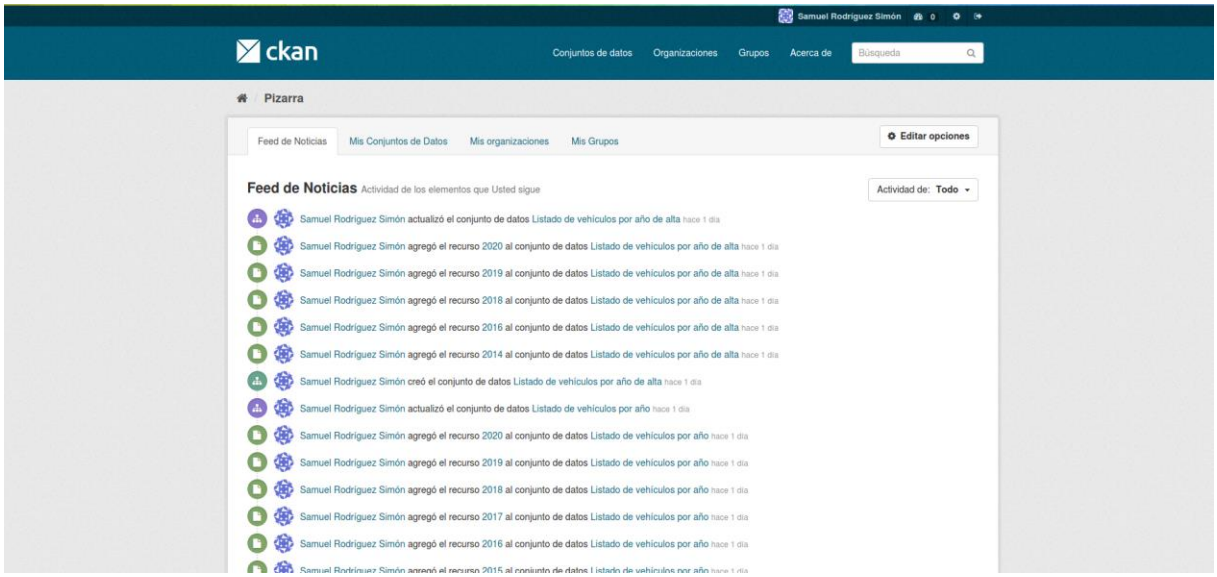

Figura 3.9: Feed de Noticias de un usuario.

3.3. Desarrollo de la API REST

En esta sección, trataremos todo lo relativo al desarrollo de la API, desde la definición del modelo de datos hasta la documentación, pasando por la implementación de los endpoints y la seguridad de algunos.

3.3.1. Modelo de datos para el conjunto

Antes de empezar con el desarrollo de la API, es necesario concretar el conjunto de datos que se desea tener pública en el portal de datos abiertos, para poder desarrollar la API entorno al modelo de datos que vayamos a establecer y que se pueda publicar remotamente.

Se ha decidido optar por un conjunto de datos sobre los vehículos de los que dispone una administración local (por ejemplo, un ayuntamiento), el cual contendrá los siguientes campos:

- **Tipo de vehículo *(tipo)*:** indica si el vehículo es un turismo, furgoneta, camión, autobús urbano... *Tipo cadena de texto*. *Requerido.*
- **Marca *(marca)*:** *Tipo cadena de texto*. *Requerida.*
- **Modelo *(modelo)*:** *Tipo cadena de texto*.
- **Matrícula *(matricula)*:** *Tipo cadena de texto*. *Requerida y única.*
- **Año de alta *(anio_alta)*:** Representa el año en el que el vehículo se ha dado de alta como vehículo de la administración. *Tipo número*. *Requerido.*
- **Uso *(uso)*:** Indica el tipo de uso que se le da al coche, por ejemplo, transporte público o servicios sanitarios. *Tipo cadena de texto*.
- **Institución *(institucion)*:** Organismo público al que pertenece el vehículo, por ejemplo, Servicio Andaluz de Salud o Policía Local. *Tipo cadena de texto*.

- **Régimen *(regimen)*:** Indica si el vehículo está en propiedad o alquilado. *Tipo cadena de texto*.
- **Estado *(estado)*:** Indica si el vehículo se encuentra operativo o no. *Tipo booleano*. *Requerido.*
- **Año de baja *(anio_baja)*:** indica el año en el que se ha dado de baja el vehículo, o ha dejado de ser parte de la administración pública. *Tipo número*.

Este conjunto de datos nos daría la posibilidad de consultarlo en su totalidad y parcialmente, de forma que podemos ver los vehículos que, por ejemplo, hayan servido durante cierto año, estén actualmente activos, de cierta marca…

Se ha considerado en usar fechas para los años, pero se ha considerado que, por año, es suficiente granularidad.

3.3.2. Configuración del entorno y preparación del proyecto

Para realizar el proyecto, usaremos la misma máquina virtual que CKAN, la cual es una máquina virtual VMware con 8GB de RAM y 4 hilos de procesos asignados, con el sistema operativo Ubuntu 16.04 LTS. Además, seguiremos, de aquí a terminar este capítulo, el tutorial al respecto de Manuel Torres Gil (2019), el cual explica con bastante sencillez la creación de una API REST con la pila MEAN con bastante sencillez, aunque obviamente aplicado a nuestro caso.

Lo primero es obtener los paquetes de MongoDB, Node.js, Express y Nodemon (el cual recarga Node.js ante algún cambio en el código) e instalarlos, así como iniciar la base de datos de MongoDB. Los comandos usados para la instalación y configuración de paquetes estarán expuestos en el correspondiente anexo (Torres Gil, 2019).

El siguiente paso consiste en crear un directorio para el proyecto (el cual se sitúa en la carpeta personal de la máquina virtual) en el que creamos un directorio de Express e instalamos Mongoose para la interacción con MongoDB. La estructura de directorios quedaría de la siguiente manera (Torres Gil, 2019):

```
├── api_server (directorio creado manualmente para alojar los archivos de la API)
│   ├── controllers (definen la lógica con la que tiene que reaccionar la API para dar
                     respuesta a cada una de las posibles peticiones)
│   ├── models
│   └── routes
├── app.js (en este archivo se define, entre otros, archivos de rutas, motor de plantillas y
            la ubicación de la carpeta de vistas)
├── bin
│   └── www
├── node_modules (contiene los módulos de Node.js instalados)
│   ├── …
├── package.json (contiene información descriptiva de la aplicación, punto de inicio y
                  dependencias)
├── package-lock.json
```

```
├── public (donde colocaremos imágenes, scripts y hojas de estilo para que no bloqueen
                al servidor mientras se sirven a los clientes)
│   ├── images
│   ├── javascripts
│   └── stylesheets
├── routes (contiene los archivos de rutas que indican los controladores que dan
                respuesta a cada petición)
│   ├── index.js
│   └── users.js
└── views (contiene las vistas de presentación de los datos de la aplicación)
    ├── error.jade
    ├── index.jade
    └── layout.jade
```

Antes de poder trabajar con la API, es necesario inicializar la base de datos e introducirle alguna entrada de ejemplo. Esta entrada de ejemplo será la primera que aparezca en el correspondiente anexo. Para usar MongoDB, introducimos en el terminal el comando `mongo`. Hemos llamado a la base de datos `vehiculosmunicipales` y a la colección, `vehiculos`.

Para cerrar esta parte del desarrollo de la API, creamos el archivo `api_server/models/db.js`, en el que detallaremos la conexión a la base de datos, eventos en la conexión, eventos de terminación/reinicio de la aplicación y los modelos que usaremos. Incluimos este archivo en `app.js` y probamos el funcionamiento del servidor con el comando `nodemon` sobre la carpeta del proyecto.

```
Mongoose connected to mongodb://localhost/vehiculosmunicipales
```

Si obtenemos esta respuesta, es que nuestro servidor está ejecutándose correctamente.

3.3.3. Desarrollo de los endpoints de la API

En este apartado, detallaremos el desarrollo de los endpoints, que son las URLs que van a dar respuesta a las peticiones que lleguen al servidor.

Para la implementación de la API, se ha usado el editor de código *Visual Studio Code* de *Microsoft*, y el programa *Postman* para probar los endpoints que usen métodos POST, PUT y DELETE, puesto que, con un navegador web, si es posible acceder a estos endpoints, sería necesario implementar una aplicación web para poder enviarlos con datos.

Antes de plantear el primer endpoint, es necesario implementar los modelos que va a usar nuestra API. Los esquemas de Mongoose se encargan de definir la estructura de los documentos de la colección en MongoDB asociada. En estos se detallan los campos, sus tipos y restricciones. El esquema para los vehículos está incluido en el archivo `api_server/models/vehículo.js`, el cual se añade al final del archivo `app.js`.

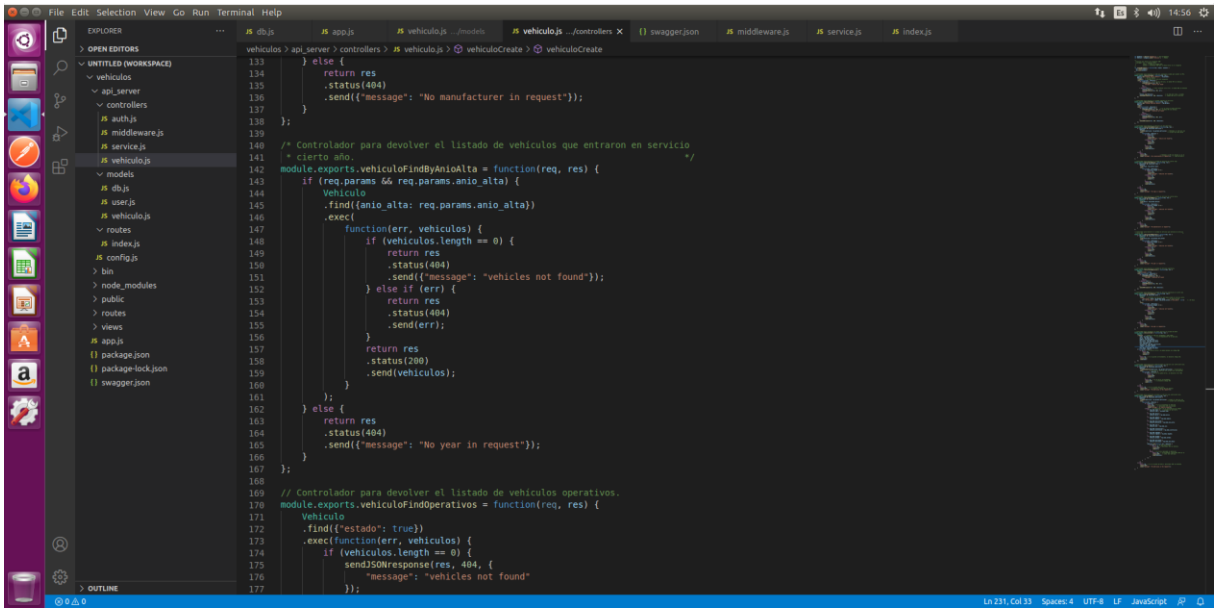

Figura 3.10: Visual Studio Code, el programa con el que se escribirá la API.

Los endpoints que tendrá nuestra API serán los siguientes:

Ruta	**Método**	**Descripción**	**Respuestas**
/	GET	Obtiene un vehículo cualquiera de la base de datos. Es un endpoint simple usado para probar la API antes de implementar el resto de endpoints de consulta. No se le pasa ningún parámetro y devuelve un vehículo.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículo.
/matricula/{matricula}	GET	Obtiene un vehículo según la matrícula que tenga, la cual se pasa por parámetro.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículo.
/todos	GET	Obtiene la lista completa de vehículos.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.
/operativos	GET	Obtiene la lista de vehículos operativos, es decir, que su campo estado sea verdadero.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.
/tipo/{tipo}	GET	Obtiene la lista de vehículos de cierto tipo, es decir, turismo, furgoneta, moto...	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.

/marca/{marca}	GET	Obtiene la lista de vehículos de cierta marca.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.
/anio_alta/{anio_alta}	GET	Obtiene la lista de vehículos que se hayan dado de alta cierto año.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.
/anio/{anio}:	GET	Obtiene la lista de vehículos que hayan estado operativos durante cierto año.	**200 (OK):** Operación correcta. **404 (Not Found):** Error al obtener vehículos.
/vehiculo	POST	Añade un vehículo nuevo a la base de datos. Se pasan como parámetro todos los campos, siendo obligatorio pasar tipo, marca, matrícula, año de alta y estado. Si la matrícula no se ha introducido o es inválida, se muestra un error. Es necesario iniciar sesión para usar este endpoint.	**201 (Created):** Operación correcta. **400 (Bad Request):** Error al crear. **401 (Unauthorized):** Token inválido. **403 (Forbidden):** Acceso prohibido (no se ha introducido token en la cabecera).
/vehiculo/{matricula}	PUT	Actualiza algún(os) campo(s) de un vehículo de la base de datos. Se introduce la matrícula para acceder al endpoint y se puede pasar por parámetro el resto de los campos. Si la matrícula no se ha introducido o es inválida, se muestra un error. Es necesario iniciar sesión para usar este endpoint.	**200 (OK):** Operación correcta. **400 (Bad Request):** Error al crear. **401 (Unauthorized):** Token inválido. **403 (Forbidden):** Acceso prohibido (no se ha introducido token en la cabecera).
	DELETE	Elimina un vehículo de la base de datos. Se pasa como parámetro la matrícula. Si la matrícula no se ha introducido o es inválida, se muestra un error. Es necesario iniciar sesión para usar este endpoint.	**204 (No Content):** Operación correcta. **400 (Bad Request):** Error al crear. **401 (Unauthorized):** Token inválido. **403 (Forbidden):** Acceso prohibido (no se ha introducido token en la cabecera).

Tabla 3.1: Endpoints de la API.

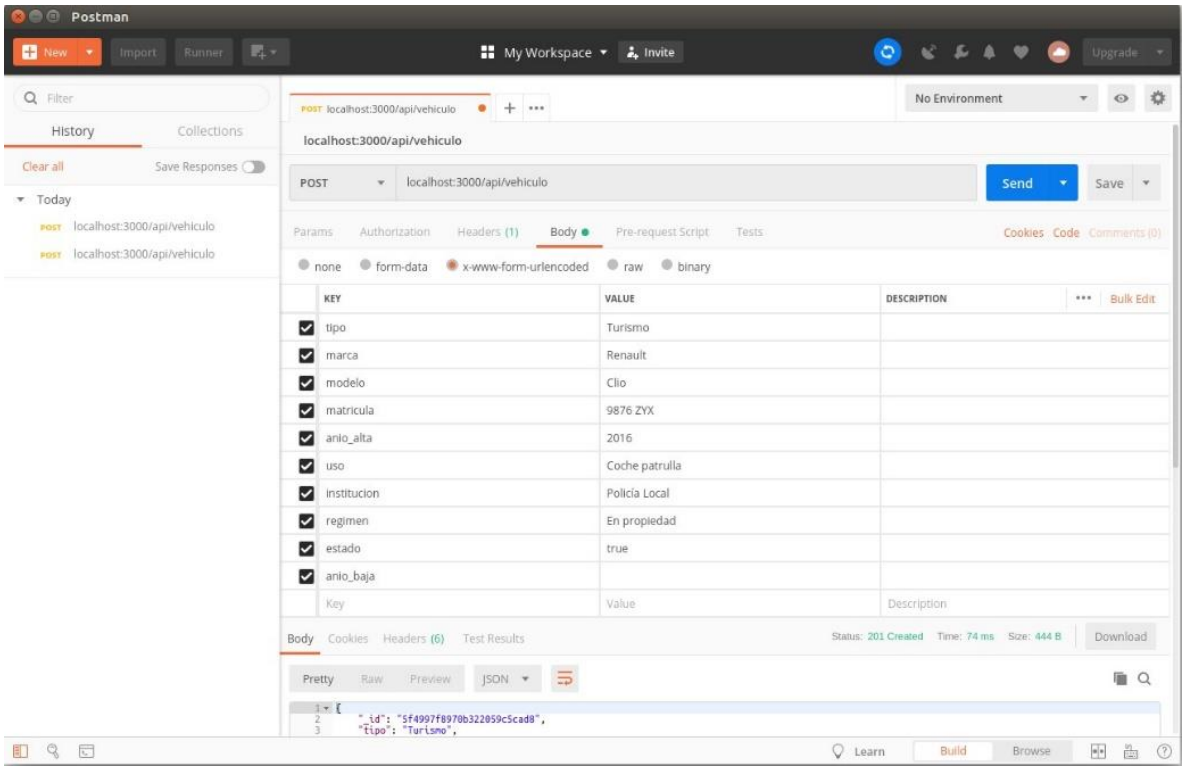

Figura 3.11: Petición PUT con Postman, programa con el que se probaron también los endpoints PUT y DELETE.

Estos controladores aparecen implementados en el archivo `/api_server/controllers/vehiculo.js`. Las rutas asociadas a cada endpoint aparecen en el archivo `/api_server/routes/index.js`.

3.3.4. Protección de ciertos endpoints: JWT

Si bien dejaremos los endpoints de consulta sin protección para que cualquiera pueda consultar detalles de cualquier vehículo almacenado, se tendrían que proteger los endpoints de creación, actualización y eliminación para prevenir que cualquier persona no autorizada pueda alterar el contenido del conjunto de datos.

Para añadir JWT a nuestra API, se instalan los paquetes *jsonwebtoken* y *moment*, para generar JWT y establecer la caducidad de los tokens respectivamente, a través de npm.

No es necesario modificar los controladores para integrarles la protección mediante JWT, ya que comprobar que un usuario puede realizar cierta acción es función del middleware.

Tras instalar los paquetes, se tiene que crear un archivo con la clave secreta para firmar los tokens, el cual es `/api_server/config.js`. Lo siguiente es crear el archivo que generará los tokens en sí, siendo este `/api_server/controllers/service.js`. Este archivo contiene una función *(ensureAuthenticated)* que construye el *payload* (el cual contiene los datos o privilegios) con datos como el usuario y fechas de expedición y caducidad. El siguiente paso consiste en crear el archivo con las funciones de registro e inicio de sesión, las cuales están implementadas en `/api_server/controllers/auth.js`. A continuación, se crea el middleware, el cual comprueba si un token es válido y, por tanto, si un usuario tiene privilegios para acceder a la función que desea ejecutar; este se encuentra en `/api_server/controllers/middleware.js`. Por último, añadimos al archivo de rutas (`/api_server/routes/index.js`) las rutas a las funciones de autenticación (Torres Gil, 2019). También es necesario crear un modelo en Mongoose para el usuario, el cual se

encuentra almacenado en `/api_server/models/user.js`, y, posteriormente, añadirlo a la parte de modelos del archivo `/api_server/models/db.js`. Una ruta con el middleware aplicado quedaría de la siguiente manera:

```
router.post('/vehiculo', middleware.ensureAuthenticated,
ctrlVeh.vehiculoCreate);
```

Los endpoints de autenticación quedarían descritos de la siguiente manera:

Ruta	**Método**	**Descripción**	**Respuestas**
/auth/signup	POST	Registra a un nuevo usuario en la base de datos	**200 (OK):** Operación correcta. **400 (Bad Request):** Error al crear el usuario.
/auth/login	POST	Obtiene un vehículo según la matrícula que tenga, la cual se pasa por parámetro.	**200 (OK):** Operación correcta. **401 (Unauthorized):** Nombre de usuario y/o contraseña incorrecta. **404 (Not Found):** Vehículo no encontrado.

Tabla 3.2: Endpoints para autenticación.

Las llamadas que hagamos a los endpoints asegurados con JWT podrían dar estas respuestas (Torres Gil, 2019):

- Si enviamos la petición sin introducir el token en la cabecera, se nos devuelve un error 403 (Forbidden) con el siguiente mensaje:
```
{
    "message": "Petición sin cabecera de autorización"
}
```
- Si enviamos la petición con un token caducado, se devuelve un error 401 (Unauthorized) con el mensaje:
```
{
    "message": "Token caducado"
}
```

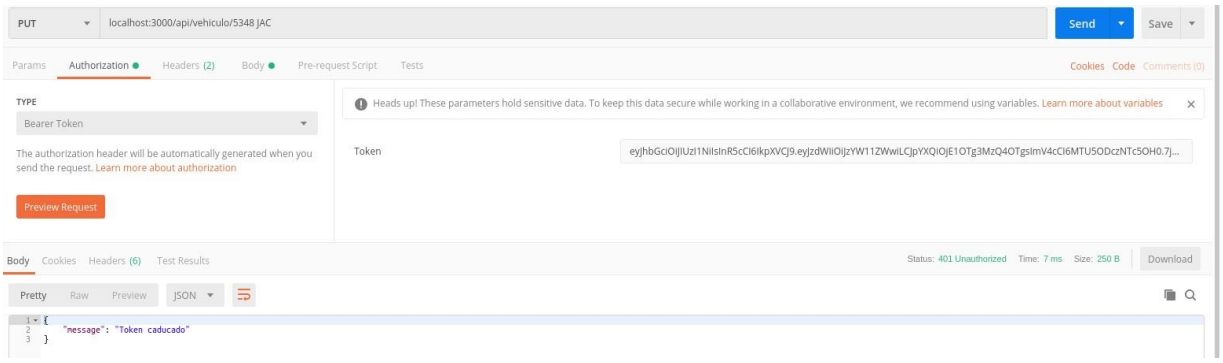

Figura 3.12: Respuesta al envío de una petición con un token caducado.

- Si nos registramos (localhost:3000/api/auth/signup) o iniciamos sesión (localhost:3000/api/auth/login) con las credenciales adecuadas, la respuesta será un token con el código de estado 200 OK.

- Si accedemos a un endpoint protegido con el token sin caducar en la cabecera (en Postman, se introduce como *Bearer Token* en la pestaña de *Autorización*), se devuelve la respuesta con código 200 OK.

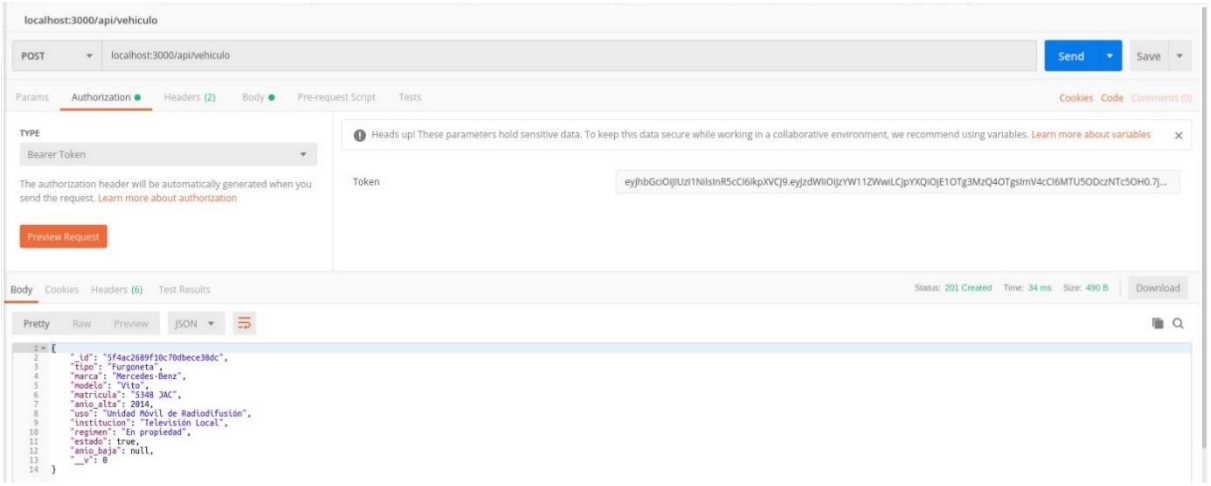

Figura 3.13: Petición POST exitosa usando un token.

Para obtener un token, como se ha mencionado anteriormente, es necesario iniciar sesión o registrarse, haciendo la petición POST correspondiente. Si se registra un nuevo usuario o un usuario existente introduce sus credenciales correctamente y sin errores, se devuelve un token en la respuesta. Este token se introduce en la cabecera como *Bearer Token* en las peticiones que requieran autenticación, tal y como se muestra en la figura de arriba.

Al no tener una aplicación web para introducir los datos fácilmente desde el navegador, usaremos Postman para introducir las credenciales en el cuerpo de las peticiones de registro e inicio de sesión, e introducir un token en la cabecera para las peticiones que necesiten autorización.

3.3.5. Documentación de la API: Swagger

Para documentar los endpoints de la API, se ha usado Swagger, el cual es un conjunto de herramientas que permite crear o documentar APIs REST. En nuestro caso, lo usaremos solo para documentar, ya que tenemos la API ya creada. Para este TFG, instalaremos la extensión para VS Code *OpenAPI (Swagger) Editor* de *42Crunch*, la cual hará menos tediosa la creación del archivo que almacena la documentación, dándonos atajos para acceder a cada ruta, petición, modelo, respuesta.... Este archivo tendrá el nombre de `swagger.json`, y se ubica en la carpeta raíz del proyecto. Su código se incluirá en el anexo correspondiente.

Este archivo muestra, jerárquicamente, información sobre la API (nombre, desarrollador, versión…), etiquetas (para organizar los endpoints), las diferentes rutas de la API con sus respectivas operaciones y los modelos que usa la API.

Para incorporar este archivo a la API, primero es necesario instalar el paquete de npm *swagger-ui-express*, y añadiríamos una nueva ruta para acceder a la documentación en el archivo `app.js`. La documentación de la API está disponible en la URL. http://localhost:3000/api-docs.

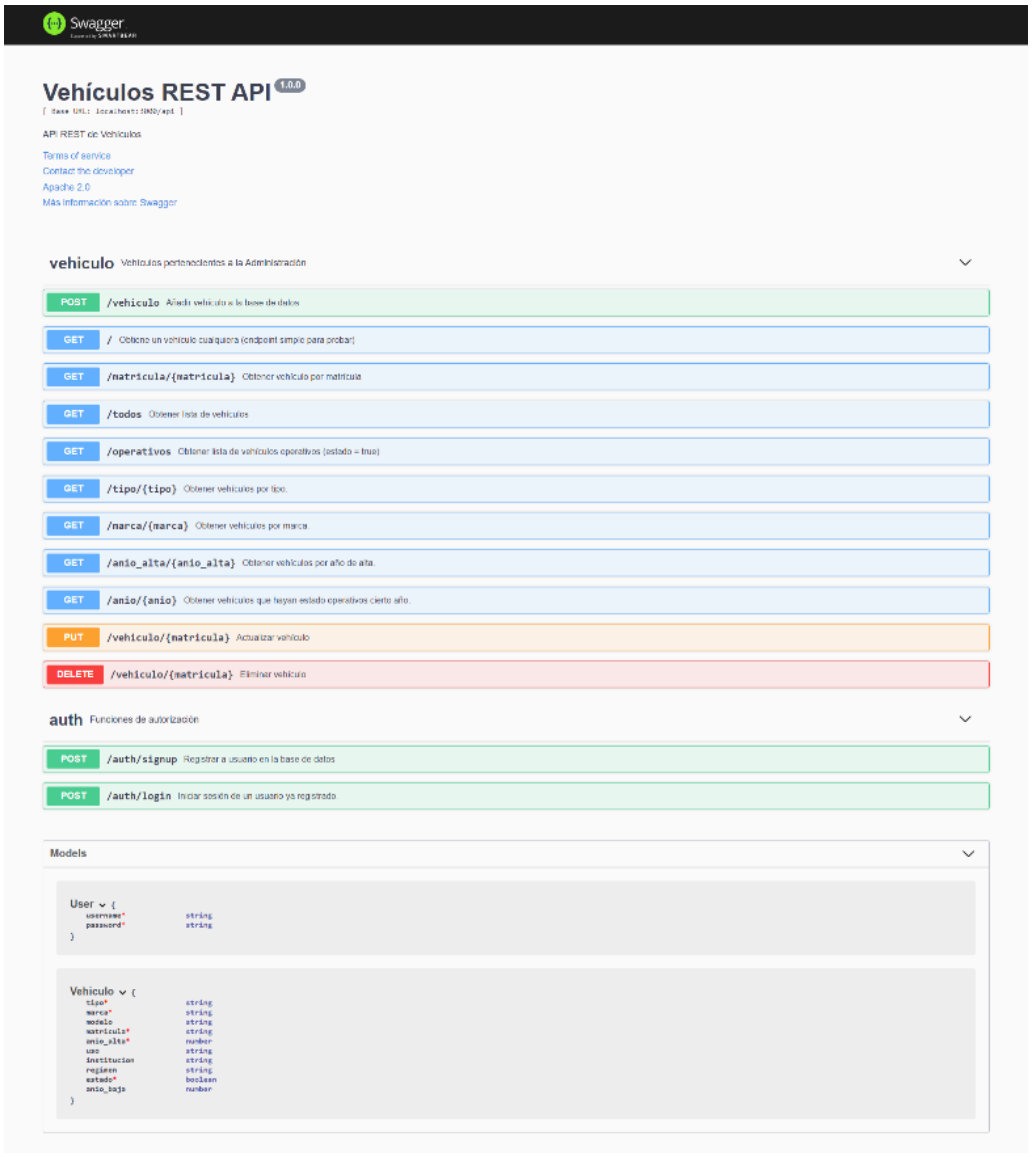

Figura 3.14: Vista de la página generada por Swagger para la documentación de la API.

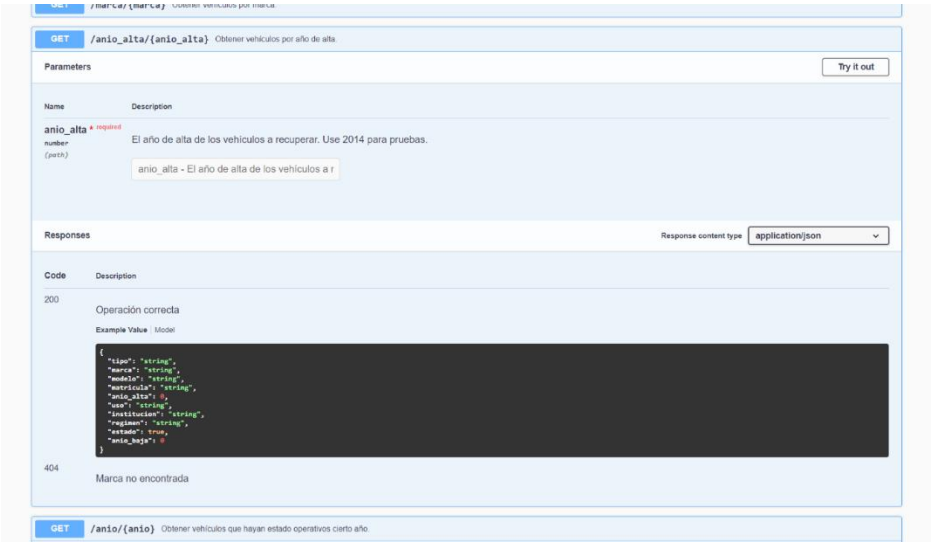

Figura 3.15: Endpoint /anio_alta/{anio_alta} documentado en la página generada por Swagger.

3.4. Acceso al conjunto de datos desde el portal de datos abiertos

En esta sección, mostraremos cómo se ha creado el conjunto de datos dentro del portal de datos abiertos para acceder a los distintos endpoints de la API, desde los cuales obtendremos un archivo JSON con los vehículos correspondientes al recurso del conjunto de datos desde el cual estamos accediendo.

En el portal de datos abiertos establecido en el capítulo 3, primero se ha creado una organización llamada *Ayuntamiento de Albella* (una institución totalmente ficticia usada como ejemplo para este TFG), en la cual publicaremos los conjuntos de datos que describiremos próximamente. Esta organización está administrada por un usuario cuyo nickname es *samuel* (es decir, un usuario relacionado con el autor de este Trabajo Fin de Grado). También se ha creado un grupo bajo el nombre de *Vehículos*, el cual recopila los conjuntos de datos creados para este fin, y administrado por el usuario *admin*.

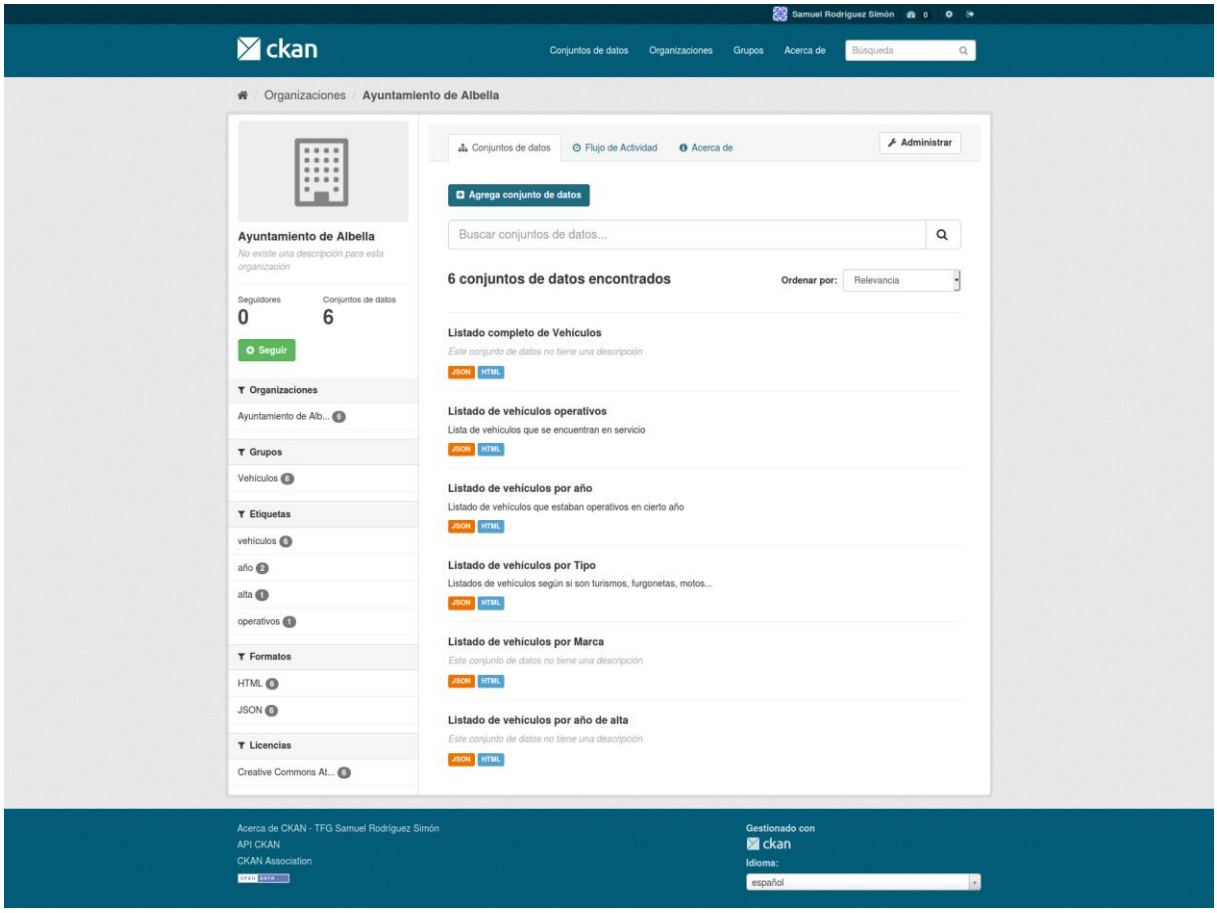

Figura 3.16: Organización creada con los conjuntos de datos.

Los conjuntos de datos contienen recursos, que son los que nos dan acceso a la API a través de enlaces los cuales llevan a un endpoint de la API (con su correspondiente parámetro si es necesario). Además, en cada conjunto de datos, se añadirá un recurso que enlaza a la página con la documentación de la API. Un conjunto de datos, si depende de ciertos parámetros, tendrán

Los conjuntos de datos creados son los siguientes:

- **Listado completo de Vehículos:** tiene un solo recurso, el cual accede a http://localhost:3000/api/todos y muestra en formato JSON el listado completo de vehículos.

- **Listado de vehículos operativos:** tiene un solo recurso, el cual accede a http://localhost:3000/api/operativos y muestra en formato JSON el listado de vehículos operativos.
- **Listado de vehículos por Tipo:** se han creado recursos para los distintos tipos de vehículos. Estos recursos acceden a http://localhost:3000/api/tipo/{tipo}, en el que pasamos por parámetro {tipo}, y muestra la lista de vehículos de ese tipo.
- **Listado de vehículos por Marca:** existen recursos para varias marcas conocidas del mercado. Estos recursos acceden a http://localhost:3000/api/marca/{marca}, en el que pasamos por parámetro {marca}, y muestra la lista de vehículos de esa marca.
- **Listado de vehículos por Año de Alta:** tiene recursos correspondientes a los años 2014, 2016, 2018, 2019 y 2020. Cada uno de estos recursos accede a http://localhost:3000/api/anio_alta/{anio}, en el que {anio} sería el año de alta que se consulta. El resultado es una lista de vehículos dados de alta ese año.
- **Listado de vehículos por Año:** tiene recursos para cada año desde el 2015. Cada uno de estos recursos accede a http://localhost:3000/api/anio/{anio}, en el que {anio} sería el año que se consulta. El resultado es una lista de vehículos que han estado operativos ese año.

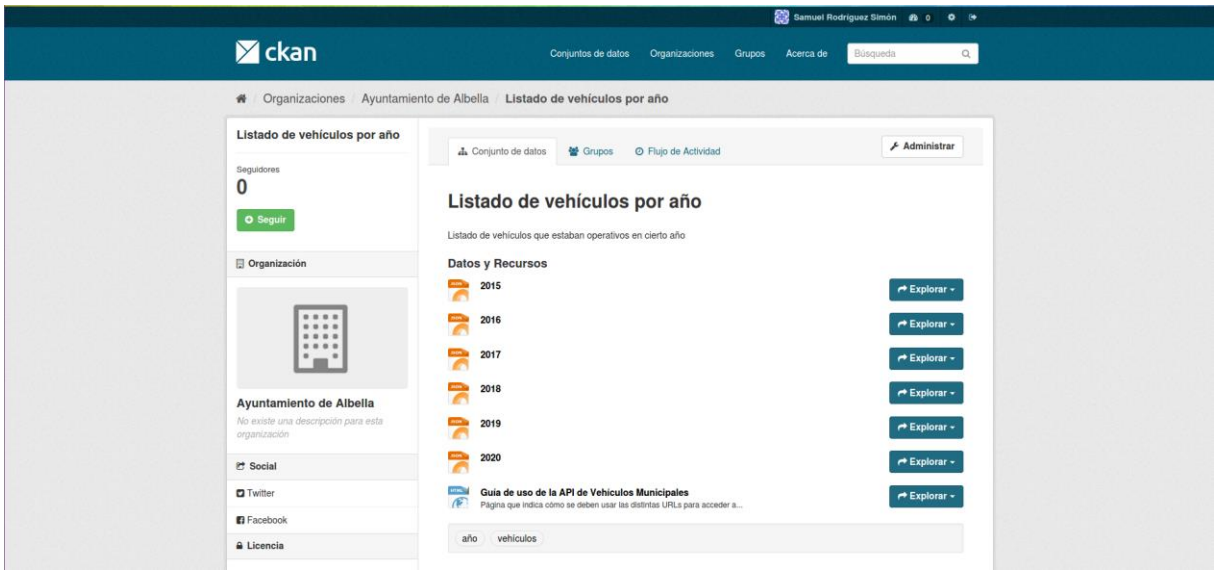

Figura 3.17: Conjunto de datos de vehículos por año.

A estos conjuntos de datos se les agregaría más recursos según pasen los años (en los conjuntos correspondientes) y según se añadan a la flota vehículos de otros tipos o marcas.

Lo normal es que se publiquen recursos que lleven a URLs que no den una lista vacía, debido a que los publicadores tienen la capacidad de saber de qué años son sus vehículos, sus marcas, sus tipos…

Aunque también, a modo de ilustración de lo que sucede cuando consultamos un recurso con un listado vacío, hemos creado en el conjunto de vehículos por marca un recurso que consulta los vehículos de la marca Porsche (atacando la URL http://localhost:3000/api/marca/Porsche), de la cual no hay vehículos. Al acceder al recurso, se nos devuelve una respuesta JSON consistente en:

```
message:   "vehicles not found"
```

Los recursos no se pueden previsualizar, ya que, al configurar cualquier recurso de estos conjuntos de datos, la única previsualización a la que da opción es a la de subir una imagen.

Por esto, se puede concluir que no podemos hacer una previsualización tabular de un archivo JSON, la cual es posible que hubiese una extensión que permitiese eso.

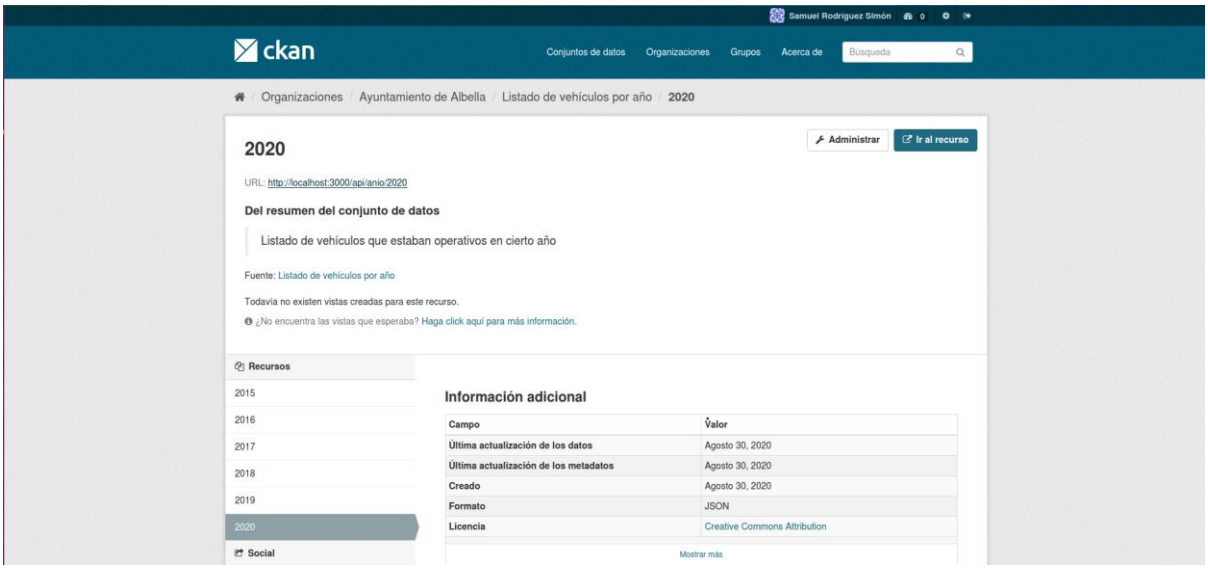

Figura 3.18: Recurso dentro de un conjunto de datos. No hay previsualizaciones disponibles para archivos JSON y se muestra un enlace al archivo.

4. Conclusiones

En este trabajo, se ha comenzado con un capítulo de un estilo más monográfico para poner en contexto sobre el fenómeno del Open Data. Como se ha visto, resultaría muy beneficioso para la sociedad que las organizaciones pusieran a su servicio datos para su reutilización, los cuales tienen un gran potencial para la creación de valor en diversas formas. Es cierto que el Open Data está muy enfocado en las organizaciones públicas como una muestra de transparencia y rendición de cuentas ante los ciudadanos, pero es una tendencia que podría ampliarse a las organizaciones privadas, pudiendo salir muy beneficiadas de compartir ciertos datos con la sociedad. Sin embargo, por cuestiones principalmente sobre privacidad y propiedad intelectual, no siempre es posible abrir ciertos datos. De todas formas, el Open Data es una industria con capacidad de crear muchos millones de euros y dar empleo a bastantes personas. Estos datos abiertos, a ser posible, deben de estar en formatos abiertos y estructurados, de forma que obtengan cierta clasificación según el esquema de estrellas definido por Tim Berners-Lee.

Tras esa introducción, se ha hablado sobre los portales a través de los cuales se suele poner al servicio de la ciudadanía estos datos abiertos, los cuales son páginas desde las que se pueden descargar los conjuntos de datos que publican estas organizaciones, aunque suelen disponer de más información como noticias, información sobre la organización o sobre la API para acceder a los datos. Se han comparado las características de distintas alternativas de software para crear un portal de datos, eligiendo CKAN por ser totalmente gratuita, modular, dar juego a los desarrolladores, fácil de configurar para una demostración y uno de los más populares. Se ha visto también cómo se ha realizado la instalación y configuración de la instancia usada de CKAN, la cual se ha instalado en una máquina virtual, así como un pequeño recorrido por un portal de datos CKAN por defecto. La configuración de un portal sencillo resulta muy sencilla, ya que solo consiste en instalar los paquetes necesarios y alterar el archivo de configuración de CKAN. Los portales de datos abiertos reales tienen que cambiar la estética de la página e, incluso, desarrollar alguna extensión en caso de necesitar una funcionalidad para la que no existe ninguna extensión.

Se ha desarrollado una API REST para acceder a una base de datos de vehículos de un ayuntamiento desde el portal de datos abiertos, haciendo uso de la pila MEAN, la cual también se ha introducido y se programa íntegramente en JavaScript. Esta API ha sido sencilla de desarrollar, debido a que el principal cometido es consultar según determinado parámetro un listado de vehículos, aunque también se han implementado operaciones de creación, eliminación y actualización. Estas tres últimas operaciones están protegidas por medio de JWT, un mecanismo de autenticación el cual devuelve un token a un usuario autorizado el cual se guarda en el navegador y sirve para autenticar a un usuario. La API se ha documentado con Swagger, el cual muestra la documentación de los endpoints de la API en un formato bastante atractivo. El desarrollo de la API, gracias al material proporcionado por mis tutores, me ha resultado muy sencillo teniendo en cuenta el tener muy poca experiencia previa en JavaScript y con una base de datos NoSQL como lo es MongoDB.

Para poder acceder a la API desde el portal de datos, hemos creado varios conjuntos de datos según qué datos se desean filtrar, teniendo varios recursos en algunos de esos conjuntos según los distintos valores que un parámetro puede tomar. Resulta muy interesante que se pueda consultar directamente ciertos datos desde el portal de datos sin tener que descargar el conjunto de datos y procesarlo para obtenerlos filtrados según ese parámetro. Se podrían crear más conjuntos de datos según otros parámetros, pero los expuestos en este TFG son suficientes para ilustrar el acceso al conjunto de datos con distintos filtros desde el portal de datos.

5. Desarrollos futuros

Este proyecto técnico podría ampliarse de la siguiente manera:

- Lo primero sería configurar una instancia de CKAN para ejecutarse sobre un servidor, ya que en este trabajo, tanto el portal de datos como la API REST estaban ejecutándose en un servidor en local. La instancia de CKAN tendría un tema configurado consultando la guía de personalización correspondiente de la documentación de CKAN. Además, también se instalarían las extensiones que se vean necesarias para, por ejemplo, previsualización de un archivo JSON en formato tabular. Si no existiese esa extensión, se desarrollaría y se podría subir al repositorio de extensiones de CKAN, beneficiando al resto de la comunidad que pueda estar interesada en esa extensión. Se podría plantear el uso de una versión de CKAN integrada a una CMS, como lo son DKAN (Drupal) o DataPress (WordPress), los cuales facilitarían desarrollar un portal de datos abiertos con varias páginas.
- Se podrían desarrollar más endpoints de la API para poder disponer de más filtros sobre los datos, como un filtro según la institución a la que pertenece o la obtención de un listado de vehículos en propiedad y otro de vehículos alquilados. También podrían crearse nuevos endpoints de actualización, como uno para dar de baja un vehiculo. Se podría modificar el modelo de datos sobre el usuario con más datos e implementar más endpoints relacionados con los usuarios, como para obtener los datos del usuario, cambiar estos datos y eliminar el usuario.
- Podría desarrollarse una aplicación para hacer más sencilla la gestión de los datos del conjunto. Esta aplicación se podría programar con cualquier framework para frontend, como Angular.js (que forma parte de la pila MEAN), React o Vue.js, y serviría para obtener y modificar vehículos, así como crearlos de una forma más accesible y sin tener que introducir los datos de las peticiones correspondientes en Postman, la cual es una aplicación que requiere de cierto conocimiento técnico y diseñada para probar las peticiones que se realizan a la API. La aplicación web estaría diseñada para ser usada por un funcionario que no reúna esos conocimientos técnicos.
- Al crearse nuevos endpoints, se podrían crear nuevos conjuntos de datos en CKAN que permitiesen realizar otro tipo de consultas, como por ejemplo obtener un listado de los vehículos alquilados por el ayuntamiento o de vehículos dados de baja tal año. Sin embargo, podrían llegar a publicarse una gran cantidad de conjuntos de datos.
- Si se construye el portal de datos con la posibilidad de almacenar conjuntos de datos en él, se puede programar un script en Python para obtener un conjunto de datos remoto usando la técnica de *web scraping* (la cual consiste en adquirir datos de un archivo remoto), para después almacenarlo en CKAN a través de la API que ofrece este portal de datos.

6. Bibliografía

Alarcón, J. (19 de Enero de 2015). *Qué es el stack MEAN y cómo escoger el mejor para ti*. Recuperado el 25 de Agosto de 2020, de CampusMVP: https://www.campusmvp.es/recursos/post/Que-es-el-stack-MEAN-y-como-escoger-el-mejor-para-ti.aspx

Almansa Morales, A. (2017). *Transparencia y datos abiertos en la Administración pública.* Instituto Nacional de Administración Pública (INAP).

Apiary. (s.f.). *API Blueprint*. Recuperado el 6 de Septiembre de 2020, de API Blueprint: https://apiblueprint.org/

B., G. (15 de Febrero de 2018). *Tutorial completo de Phalcon para principiantes*. Recuperado el 5 de Septiembre de 2020, de Hostinger Tutoriales: Tutorial completo de Phalcon para principiantes

Bean, M. (2015). *Laravel 5 Essentials.* Packt Publishing.

CKAN. (2017). *About*. Recuperado el 18 de Agosto de 2020, de CKAN: ckan.org/about

CKAN. (s.f.). *Installing CKAN*. Recuperado el 20 de Agosto de 2020, de CKAN 2.8.5 documentation: docs.ckan.org/en/2.8/maintaining/installing/index.html

CKAN. (s.f.). *User Guide*. Recuperado el 20 de Agosto de 2020, de CKAN 2.8.5 Documentation: docs.ckan.org/en/2.8/user-guide.html

DDI Development. (s.f.). *Advantages and disadvantages of Laravel Framework for web Development*. Recuperado el 5 de Septiembre de 2020, de DDI Development: https://ddi-dev.com/blog/programming/pros-and-cons-of-laravel-framework-for-web-app-development/

Elkstein, M. (Febrero de 2008). *What is REST?* Recuperado el 24 de Agosto de 2020, de Learn REST: A Tutorial: http://rest.elkstein.org/2008/02/what-is-rest.html

Esri. (s.f.). *ArcGIS Open Data | Discover, publish, and share open data*. Recuperado el 19 de Agosto de 2020, de Esri: www.esri.com/en-us/arcgis/products/arcgis-open-data

Express. (s.f.). *Express - Infraestructura de aplicaciones web Node.js*. Recuperado el 25 de Agosto de 2020, de Express: https://expressjs.com/es/

Geary, D., & Horstmann, C. (2010). *Core JavaServer™ Faces.* Pearson.

Getting Started with Rails. (s.f.). Recuperado el 4 de Septiembre de 2020, de Ruby on Rails Guides: https://guides.rubyonrails.org/getting_started.html#what-is-rails-questionmark

Gurin, J. (2014). *Open data now : the secret to hot startups, smart investing, savvy marketing, and fast innovation.* McGraw Hill Education.

Hammond, S. (. (21 de Agosto de 2012). *Basic Orientation to CKAN (for developers)*. Recuperado el 18 de Agosto de 2020, de GitHub: gist.github.com/seanh/3414107

Hartl, M. (2020). *Ruby on Rails Tutorial* (Sexta ed.). Addison-Wesley Professional.

Holmes, S. (2019). *Getting MEAN with Mongo, Express, Angular, and Node* (Segunda ed.). Manning Publications.

Huyer, E., & van Knippenberg, L. (2020). *The Economic Impact of Open Data: Opportunities for value creation in Europe.* Bruselas: Comisión Europea. Retrieved Agosto 12, 2020,

from https://www.europeandataportal.eu/sites/default/files/the-economic-impact-of-open-data.pdf

Iniciativa Aporta. (Mayo de 2019). *Modelo de Gobernanza.* Recuperado el 15 de Agosto de 2020, de Iniciativa Aporta: https://datos.gob.es/sites/default/files/datosgobes/190522_iniciativaaporta_modelogobernanza2019.pdf

Junar. (s.f.). *Junar - Plataforma de datos*. Recuperado el 19 de Agosto de 2020, de Junar: junar.com/?l=es

jwt.io. (s.f.). *Introduction to JSON Web Tokens*. Recuperado el 7 de Septiembre de 2020, de jwt.io: https://jwt.io/introduction/

Katz, Y., Klabnik, S., Bigg, R., & Skinner, R. (2015). *Rails 4 In Action.* Manning Publications.

Lerman, R. (18 de Abril de 2018). *Seattle tech company Socrata bought by Tyler Technologies*. Obtenido de The Seattle Times: www.seattletimes.com/business/technology/seattle-tech-company-socrata-bought-by-tyler-technologies/

McGaw, J. (2009). *Beginning Django E-Commerce.* Apress.

McGreggor, A. (10 de Febrero de 2014). *CKAN vs. Socrata (respuesta de Adam McGreggor).* Obtenido de Open Data Stack Exchange: opendata.stackexchange.com/questions/1517/ckan-vs-socrata/1524#1524

Monino, J.-L., & Sedkaoui, S. (2016). *Big Data, Open Data and Data Development.* ISTE Ltd/John Wiley and Sons Inc.

Nader, Y. (9 de Abril de 2020). *Getting Started With Rails*. Recuperado el 4 de Septiembre de 2020, de hackr.io: https://hackr.io/blog/getting-started-with-rails

Nader, Y. (9 de Abril de 2020). *What is Django? Advantages and Disadvantages*. Recuperado el 4 de Septiembre de 2020, de hackr.io: hackr.io/blog/what-is-django-advantages-and-disadvantages-of-using-django

Neumaier, S., & Umbrich, J. (2016). Measures for Assessing the Data Freshness in Open Data Portals. *2016 2nd International Conference on Open and Big Data (OBD)* (págs. 17-24). Viena: IEEE.

Open Data Charter. (2015). *Principles*. Recuperado el 14 de Agosto de 2020, de International Open Data Charter: opendatacharter.net/principles/

Open Government Working Group. (Diciembre de 2007). *The Annoted 8 Principles of Open Government Data*. Recuperado el 14 de Agosto de 2020, de opendatagov.org: opengovdata.org

Open Knowledge Foundation. (2012). *El manual de Open Data*. Recuperado el 15 de Junio de 2020, de Open Data Handbook: http://opendatahandbook.org/guide/es/

Open Knowledge Foundation. (Noviembre de 2014). *Definición de Conocimiento Abierto*. Recuperado el 26 de Junio de 2020, de Open Definition: opendefinition.org/od/2.0/es/

Open Knowledge Foundation. (2014). *What is open?* Recuperado el 26 de Junio de 2020, de Open Knowledge Foundation: okfn.org/opendata/

OpenAPI Initiative. (18 de Junio de 2020). *The OpenAPI Specification*. Recuperado el 5 de Septiembre de 2020, de GitHub: github.com/OAI/OpenAPI-Specification/

OpenDataSoft. (6 de Diciembre de 2012). *From open data to API-driven business.* Recuperado el 19 de Agosto de 2020, de SlideShare: www.slideshare.net/OpenDataSoft/open-datasoft-apidays12042012v0

OpenDataSoft. (9 de Julio de 2012). *OpenDataSoft platform designed for big data issues.* Obtenido de SlideShare: https://www.slideshare.net/OpenDataSoft/opendatasoft-platform-designed-for-big-data-issues

OpenDataSoft. (s.f.). *Crear un conjunto de datos*. Recuperado el 19 de Agosto de 2020, de Documentación de OpenDataSoft: help.opendatasoft.com/platform/es/publishing_data/01_creating_a_dataset/creating_a_dataset.html

OpenDataSoft. (s.f.). *Esto es OpenDataSoft*. Recuperado el 19 de Agosto de 2020, de OpenDataSoft: www.opendatasoft.com/es/esto-es-opendatasoft

OpenDataSoft. (s.f.). *Search API v2 documentation*. Recuperado el 19 de Agosto de 2020, de OpenDataSoft: help.opendatasoft.com/apis/ods-search-v2

Pardo, D. (14 de Febrero de 2019). *¿Qué es y para qué sirve una API? Aquí las respuestas necesarias*. Recuperado el 24 de Agosto de 2020, de Pandora FMS: https://pandorafms.com/blog/es/para-que-sirve-una-api/

Phalcon. (s.f.). *Framework PHP de alto rendimiento*. Recuperado el 5 de Septiembre de 2020, de Phalcon Framework: https://phalcon.io/es-es

Phalcon. (s.f.). *Introducción*. Recuperado el 5 de Septiembre de 2020, de Phalcon Documentation: https://docs.phalcon.io/4.0/es-es/introduction

Rak, V. (s.f.). *Pros and Cons of Ruby on Rails*. Recuperado el 4 de Septiembre de 2020, de Sloboda Studio: https://sloboda-studio.com/blog/pros-and-cons-of-ruby-on-rails/

RAML. (6 de Junio de 2019). *RAML Version 1.0: RESTful API Modeling Language*. Recuperado el 6 de Septiembre de 2020, de GitHub: https://github.com/raml-org/raml-spec/blob/master/versions/raml-10/raml-10.md/

Safe Software. (25 de Junio de 2015). *Open Data Portals: 9 Solutions and How they Compare.* Recuperado el 19 de Agosto de 2020, de SlideShare: https://www.slideshare.net/SafeSoftware/open-data-portals-2015-webinar

Sandoval, K. (8 de Septiembre de 2015). *Top Specification Formats for REST APIs*. Recuperado el 6 de Septiembre de 2020, de Nordic APIs: https://nordicapis.com/top-specification-formats-for-rest-apis/

slatedocs. (s.f.). *slate/README.md*. Recuperado el 6 de Septiembre de 2020, de GitHub: https://github.com/slatedocs/slate/blob/main/README.md

St. Laurent, S., & Dumbill, E. (2010). *Learning Rails: Live Editions.* O'Reilly Media.

Torres Gil, M. (8 de Mayo de 2019). *API REST para la pila MEAN (JWT y Swagger incluido)*. Obtenido de Github: ualmtorres.github.io/TutorialAPIRESTMEANEn3ms/

Tyler Technologies. (s.f.). *Socrata Open Data & Citizen Engagement*. Recuperado el 19 de Agosto de 2020, de Tyler Technologies: www.tylertech.com/products/socrata/open-data-citizen-engagement

Young, A., & Verhulst, S. (2016). *The Global Impact of Open Data.* O'Reilly Media, Inc.

Anexo I. Comandos usados para la instalación de los paquetes de la pila MEAN

Los comandos expuestos en este anexo han sido los usados para instalar y configurar el entorno en el que se encuentra la API, siguiendo los pasos del tutorial de Manuel Torres Gil (2019).

```
# Instalación de MongoDB
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
9DA31620334BD75D9DCB49F368818C72E52529D4
sudo echo "deb [ arch=amd64,arm64 ]
https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.0
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org
sudo service mongod start

# Instalación de Node.js
sudo curl -sL https://deb.nodesource.com/setup_10.x | sudo -E bash -
sudo apt-get install -y nodejs

# Instalación de Express
npm install -g express-generator

# Instalación de Nodemon
npm install -g nodemon

# Creación de la carpeta de proyecto
mkdir vehiculos
cd vehiculos
express
npm install

# Instalación de Mongoose
npm install mongoose –save

# Creación de la carpeta api_server
mkdir -p api_server/models
mkdir -p api_server/controllers
mkdir -p api_server/routes

# Creación de la base de datos en MongoDB
$ mongo
> create database vehiculosmunicipales;
> use vehiculosmunicipales;
> db.books.insert(
{
```

```
      "tipo": "Turismo",
      "marca": "Ford",
      "modelo": "Fusion",
      "matricula": "1234 ABC",
      "anio_alta": 2018,
      "uso": "Coche patrulla",
      "institucion": "Policía Local",
      "regimen": "En propiedad",
      "estado": false,
      "anio_baja": 2020
});

# Instalación de JSON Web Token y Moment
npm install jsonwebtoken --save
npm install moment –save

# Instalación de Swagger
npm install swagger-ui-express –save
```

Código 1: Comandos utilizados para instalar y configurar el proyecto para la API.

Anexo II. Conjunto de datos usado

El conjunto de datos completo se ha obtenido tras extraerlo de MongoDB a través del comando `db.vehiculos.find()` sobre la base de datos `vehiculosmunicipales`.

- { "_id" : ObjectId("5f470080aa2f9558568357e5"), "tipo" : "Turismo", "marca" : "Ford", "modelo" : "Fusion", "matricula" : "1234 ABC", "anio_alta" : 2018, "uso" : "Coche patrulla", "institucion" : "Policía Local", "regimen" : "En propiedad", "estado" : false, "anio_baja" : 2020 }
- { "_id" : ObjectId("5f49b886806a312ec85fe06f"), "tipo" : "Turismo", "marca" : "Renault", "modelo" : "Megane", "matricula" : "1111 AAA", "anio_alta" : 2005, "uso" : "Coche patrulla", "institucion" : "Policía Local", "regimen" : "En propiedad", "estado" : false, "anio_baja" : 2013, "__v" : 0 }
- { "_id" : ObjectId("5f49b8af806a312ec85fe070"), "tipo" : "Turismo", "marca" : "Renault", "modelo" : "Clio", "matricula" : "9876 ZYX", "anio_alta" : 2016, "uso" : "Coche patrulla", "institucion" : "Policía Local", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ac2689f10c70dbece38dc"), "tipo" : "Furgoneta", "marca" : "Mercedes-Benz", "modelo" : "Vito", "matricula" : "5348 JAC", "anio_alta" : 2014, "uso" : "Unidad Móvil de Radiodifusión", "institucion" : "Televisión Local", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ec2a56c51e1265e7db785"), "tipo" : "Moto", "marca" : "BMW", "modelo" : "R1200GS", "matricula" : "5462 EAD", "anio_alta" : 2015, "uso" : "Moto Patrulla", "institucion" : "Policía Local", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ec3106c51e1265e7db786"), "tipo" : "Autobús", "marca" : "Mercedes-Benz", "modelo" : "Citaro", "matricula" : "8342 WLA", "anio_alta" : 2017, "uso" : "Transporte Interurbano", "institucion" : "Empresa Local de Autobuses", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ec3cd6c51e1265e7db788"), "tipo" : "Autobús", "marca" : "Mercedes-Benz", "modelo" : "Citaro", "matricula" : "5348 RAG", "anio_alta" : 2002, "uso" : "Transporte Interurbano", "institucion" : "Empresa Local de Autobuses", "regimen" : "En propiedad", "estado" : false, "anio_baja" : 2017, "__v" : 0 }
- { "_id" : ObjectId("5f4ec4ef6c51e1265e7db78a"), "tipo" : "Autocar", "marca" : "Volvo", "modelo" : "B11R", "matricula" : "2345 HVA", "anio_alta" : 2012, "uso" : "Transporte a otras ciudades", "institucion" : "Empresa Local de Autobuses", "regimen" : "En alquiler", "estado" : false, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ec6446c51e1265e7db78b"), "tipo" : "Moto", "marca" : "Honda", "modelo" : "Africa-Twin", "matricula" : "5348 ADV", "anio_alta" : 2019, "uso" : "Moto Patrulla", "institucion" : "Policía Local", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }
- { "_id" : ObjectId("5f4ec69b6c51e1265e7db78c"), "tipo" : "Camión", "marca" : "Mercedes-Benz", "matricula" : "5345 MIO", "anio_alta" : 2010, "uso" : "Camión de bomberos", "institucion" : "Bomberos", "regimen" : "En propiedad", "estado" : true, "anio_baja" : null, "__v" : 0 }

Anexo III. Modelos de Mongoose

Modelo de Vehículo *(api_server/models/vehiculo.js)*

```
var mongoose = require('mongoose');

// Esquema del vehículo
var vehicleSchema = mongoose.Schema({
    tipo: {
        type: String,
        required: true
    },
    marca: {
        type: String,
        required: true
    },
    modelo: {
        type: String
    },
    matricula: {
        type: String,
        required: true,
        unique: true
    },
    anio_alta: {
        type: Number,
        required: true
    },
    uso: {
        type: String
    },
    institucion: {
        type: String
    },
    regimen: {
        type: String
    },
    estado: {
        type: Boolean,
        required: true
    },
    anio_baja: {
        type: Number
    }
});

// Creación del modelo a partir del esquema
mongoose.model('Vehiculo', vehicleSchema);
```

Código 2. Implementación del modelo de vehículo en Mongoose (vehiculo.js).

Modelo de Usuario *(api_server/models/user.js)*

```
var mongoose = require('mongoose');

// Esquema de usuario
var userSchema = mongoose.Schema({
    username: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    }
});

// Creación del modelo a partir del esquema
mongoose.model('User', userSchema);
```

Código 3: Implementación del modelo de usuario en Mongoose (user.js).

Anexo IV. Archivos de código de la API

Archivos relacionados con la autenticación

Clave secreta (/api_server/config.js)

```
// Este archivo contiene el secreto con el que se encriptan los
tokens.
module.exports = {
    TOKEN_SECRET: process.env.TOKEN_SECRET || "password"
};
```

Código 4: Implementación de la clave secreta (config.js).

Función de creación del token (/api_server/controllers/service.js)

```
var jwt = require('jsonwebtoken');  // Uso de JWT
var moment = require('moment');   // Uso de Moment
var config = require('../config');  // Carga del secreto

// Función encargada de crear el token. Toma al usuario como
parámetro.
exports.createToken = function(user) {
  var payload = {  // Construcción del payload.
    sub: user,  // Inclusión del usuario.
    iat: moment().unix(), // Inclusión de la fecha y hora actual.
    exp: moment().add(15, "minutes").unix(),  // Caducidad del
token.
  };
  return jwt.sign(payload, config.TOKEN_SECRET);  // Creación del
};                                    // token añadiendo el secreto.
```

Código 5: Implementación de la función de creación de tokens (service.js).

Funciones de autenticación (/api_server/controllers/auth.js)

```
var mongoose = require('mongoose');
var User = mongoose.model('User');
var service = require('./service'); // Carga del archivo con la
función createToken.

// Función para el registro de usuarios.
module.exports.signup = function(req, res) {
  User
  .create({username: req.body.username, password:
req.body.password},
    function(err, user) {
    if (err) {
      return res.status(400).send(err); // Devolver error si no
    }                                   // se ha creado el usuario.
    return res
    .status(200)
    .send({token: service.createToken(req.body.username)});
  });                          // Devolver token si se crea el usuario
con éxito.
};
```

```
// Función para iniciar sesión.
module.exports.login = function(req, res) {
  if (req.body.username && req.body.password) {
    User
    .count({username: req.body.username, password:
req.body.password}) // Contar
    .exec(function(err, user) { // el número de usuarios cón ese
nombre y contraseña.
      if (!user) {  // Si no existe usuario con esas credenciales,
se envía un error.
        return res.status(401).send({"message": "Invalid user and/or
password"});
      } else if (err) { // Devolver error si ha habido un error al
        return res.status(404).send(err); // realizar la consulta.
      }
      return res
      .status(200)  // Si existen las credenciales, se devuelve un
token de acceso.
      .send({token: service.createToken(req.body.username)});
    });
  } else {  // Devolver error si falta algún parámetro.
    return res.status(401).send({"message": "Invalid user and/or
password"});
  }
};
```

Código 6: Implementación de las funciones de autenticación (auth.js).

Middleware (/api_server/controllers/middleware.js)

```
var jwt = require('jsonwebtoken');  // Uso de JWT.
var moment = require('moment');
var config = require('../config');

// Función para comprobar si un usuario tiene acceso a cierto
endpoint.
exports.ensureAuthenticated = function(req, res, next) {
  if(!req.headers.authorization) {  // Comprobar la existencia de
autorización
    return res                      // en la cabecera.
    .status(403)
    .send({message: "Petición sin cabecera de autorización"});
  }

  var token = req.headers.authorization.split(" ")[1];  // Obtener
el token.
  var payload = jwt.verify(token, config.TOKEN_SECRET, function(err,
payload) {
    if (err) {
      switch (err.name) { // Comprobar errores.
        case 'JsonWebTokenError':
```

```
          return res.status(401).send({message: "Signatura
incorrecta"});
        case 'TokenExpiredError':
          return res.status(401).send({message: "Token caducado"});
        default:
          return res.status(401).send(err);
      }
    }
    req.user = payload.sub; // Cargar datos del payload para pasar a
la sig. etapa.
    next(); // Paso a la etapa siguiente.
  });
}
```

Código 7: Implementación del middleware (middleware.js).

Gestión de conexiones/desconexiones (/api_server/models/db.js)

```
var mongoose = require('mongoose'); // Uso de mongoose

var dbURI = 'mongodb://localhost/vehiculosmunicipales'; // URI de
nuestra BD
mongoose.connect(dbURI); // Conexion a la BD

// EVENTOS DE CONEXION
mongoose.connection.on('connected', function() {
    console.log('Mongoose connected to ' + dbURI);
});
mongoose.connection.on('error', function(err) {
    console.log('Mongoose connection error: ' + err);
});
mongoose.connection.on('disconnected', function() {
    console.log('Mongoose disconnected');
});

// EVENTOS DE TERMINACION/REINICIO DE LA APLICACION
// Es llamado cuando la aplicación se termina/reinicia
gracefulShutdown = function(msg, callback) {
    mongoose.connection.close(function() {
        console.log('Mongoose disconnected through ' + msg);
        callback();
    });
};
// Para reinicios de nodemon
process.once('SIGUSR2', function() {
    gracefulShutdown('nodemon restart', function() {
        process.kill(process.pid, 'SIGUSR2');
    });
});
// Para terminacion de la aplicacion
process.on('SIGINT', function() {
    gracefulShutdown('app termination', function() {
```

```
        process.exit(0);
    });
});

// ESQUEMAS Y MODELOS
require('./vehiculo');
require('./user');
```

Código 8: Implementación de la lógica de conexiones y desconexiones con la BD (db.js).

Archivo app.js (/app.js)

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var swaggerUi = require('swagger-ui-express');  // Uso de Swagger
var swaggerDocument = require('./swagger.json');

// Conectar a la BD y cargar los modelos
require('./api_server/models/db');

/* Archivo de rutas para direccionar las
 * peticiones a sus correspondientes funciones. */
var apiRouter = require('./api_server/routes/index');
//var indexRouter = require('./routes/index');
//var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));  // Establecer
carpeta de vistas.
app.set('view engine', 'jade'); // Jade como motor de plantillas

app.use('/api-docs', swaggerUi.serve,
swaggerUi.setup(swaggerDocument));  // Ruta
app.use(logger('dev'));
// de Swagger.
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/api', apiRouter);     // Uso de las rutas de la API cuando
//app.use('/', indexRouter);        // lleguen peticiones a /api.
//app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
```

```
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err :
{};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```

Código 9: Archivo app.js.

Archivo de rutas (/api_server/routes/index.js)

```
var express = require('express');
var router = express.Router();

// Archivos con el código de los controladores
var ctrlVeh = require('../controllers/vehiculo');   // Vehículos
var ctrlAuth = require('../controllers/auth');   // Registro y login
var middleware = require('../controllers/middleware');
// Comprobación de acceso

// Rutas a endpoints GET
router.get('/', ctrlVeh.vehiculoFindOne);
router.get('/todos', ctrlVeh.vehiculoFindAll);
router.get('/matricula/:matricula',
ctrlVeh.vehiculoFindByMatricula);
router.get('/tipo/:tipo', ctrlVeh.vehiculoFindByTipo);
router.get('/marca/:marca', ctrlVeh.vehiculoFindByMarca);
router.get('/alta/:anio_alta', ctrlVeh.vehiculoFindByAnioAlta);
router.get('/operativos', ctrlVeh.vehiculoFindOperativos);
router.get('/anio/:anio', ctrlVeh.vehiculoFindByAnio);

// Rutas a endpoints POST
router.post('/vehiculo', middleware.ensureAuthenticated,
ctrlVeh.vehiculoCreate);

// Rutas a endpoints DELETE
router.delete('/vehiculo/:matricula',
middleware.ensureAuthenticated, ctrlVeh.vehiculoDelete);

// Rutas a endpoints PUT
router.put('/vehiculo/:matricula', middleware.ensureAuthenticated,
ctrlVeh.vehiculoUpdate);
```

```
// Rutas de registro y login (POST)
router.post('/auth/signup', ctrlAuth.signup);
router.post('/auth/login', ctrlAuth.login);

module.exports = router;
```

Código 10: Implementación de las rutas de la API (index.js).

Controladores (/api_server/controllers/vehiculo.js)

```
var mongoose = require('mongoose'); // Uso de Mongoose
var Vehiculo = mongoose.model('Vehiculo');  // Modelo

/*
 * Función para enviar una respuesta JSON
 * Entradas: res -> respuesta HTTP
 *           status -> Código de estado HTTP
 *           content -> Contenido JSON que se desea enviar en la
respuesta
*/
var sendJSONresponse = function(res, status, content) {
  res.status(status);
  res.json(content);
};

// Controlador para encontrar un vehículo cualquiera (usado para
probar la API).
module.exports.vehiculoFindOne = function(req, res) {
    console.log('Finding vehicle details', req.params);
    Vehiculo    // Uso del modelo
    .findOne()  // Búsqueda de un vehículo cualquiera
    .exec(function(err, vehiculo) {
        if (!vehiculo) {    // Si no hay vehículo, se manda 200 con
mensaje.
            sendJSONresponse(res, 200, {
                "message": "vehicle not found"
            });
            return;
        } else if (err) {    // Para cualquier otro error, se manda
404 con mensaje.
            console.log(err);
            sendJSONresponse(res, 404, err);
            return;
        }
        console.log(vehiculo);                  // Si todo está
bien, se manda
        sendJSONresponse(res, 200, vehiculo);   // código 200 con el
vehículo.
    });
};

// Controlador para devolver el listado completo de vehículos.
```

```
module.exports.vehiculoFindAll = function(req, res) {
    console.log('Finding vehicles details', req.params);
    Vehiculo
    .find()
    .exec(function(err, vehiculos) {
        if (vehiculos.length == 0) {    // Si la lista no tiene
vehículos,
            sendJSONresponse(res, 200, {// se manda 200 con mensaje.
                "message": "vehicles not found"
            });
            return;
        } else if (err) {
            console.log(err);
            sendJSONresponse(res, 404, err);
            return;
        }
        sendJSONresponse(res, 200, vehiculos);
    });
};

// Controlador para devolver un vehículo con tal matrícula
module.exports.vehiculoFindByMatricula = function(req, res) {
    if (req.params && req.params.matricula) {
        Vehiculo
        .findOne({matricula: req.params.matricula}) // Obtenemos el
vehículo con
        .exec(                                      // la matrícula
del param.
            function(err, vehiculo) {
                if (!vehiculo) {
                    return res
                    .status(200)
                    .send({"message": "vehicle not found"});
                } else if (err) {
                    return res
                    .status(404)
                    .send(err);
                }
                return res
                .status(200)
                .send(vehiculo);
            }
        );
    } else {
        return res
        .status(404)                    // Mandamos un 404 con
mensaje si no se
        .send({"message": "No licence-plate in request"});  //  pasa
parámetro.
    }
};
```

```
// Controlador para devolver el listado de vehículos de cierto tipo.
module.exports.vehiculoFindByTipo = function(req, res) {
    if (req.params && req.params.tipo) {
        Vehiculo
        .find({tipo: req.params.tipo})
        .exec(
            function(err, vehiculos) {
                if (vehiculos.length == 0) {
                    return res
                    .status(200)
                    .send({"message": "vehicles not found"});
                } else if (err) {
                    return res
                    .status(404)
                    .send(err);
                }
                return res
                .status(200)
                .send(vehiculos);
            }
        );
    } else {
        return res
        .status(404)
        .send({"message": "No type in request"});
    }
};

// Controlador para devolver el listado de vehículos de cierta
marca.
module.exports.vehiculoFindByMarca = function(req, res) {
    if (req.params && req.params.marca) {
        Vehiculo
        .find({marca: req.params.marca})
        .exec(
            function(err, vehiculos) {
                if (vehiculos.length == 0) {
                    return res
                    .status(200)
                    .send({"message": "vehicles not found"});
                } else if (err) {
                    return res
                    .status(404)
                    .send(err);
                }
                return res
                .status(200)
                .send(vehiculos);
            }
        );
```

```
    } else {
        return res
        .status(404)
        .send({"message": "No manufacturer in request"});
    }
};

/* Controlador para devolver el listado de vehículos que entraron en
servicio
 * cierto año.
*/
module.exports.vehiculoFindByAnioAlta = function(req, res) {
    if (req.params && req.params.anio_alta) {
        Vehiculo
        .find({anio_alta: req.params.anio_alta})
        .exec(
            function(err, vehiculos) {
                if (vehiculos.length == 0) {
                    return res
                    .status(200)
                    .send({"message": "vehicles not found"});
                } else if (err) {
                    return res
                    .status(404)
                    .send(err);
                }
                return res
                .status(200)
                .send(vehiculos);
            }
        );
    } else {
        return res
        .status(404)
        .send({"message": "No year in request"});
    }
};

// Controlador para devolver el listado de vehículos operativos.
module.exports.vehiculoFindOperativos = function(req, res) {
    Vehiculo
    .find({"estado": true})
    .exec(function(err, vehiculos) {
        if (vehiculos.length == 0) {
            sendJSONresponse(res, 200, {
                "message": "vehicles not found"
            });
            return;
        } else if (err) {
            console.log(err);
            sendJSONresponse(res, 404, err);
```

```
            return;
        }
        sendJSONresponse(res, 200, vehiculos);
    });
};

// Controlador para devolver el listado de vehículos operativos en
cierto año.
module.exports.vehiculoFindByAnio = function(req, res) {
    if (req.params && req.params.anio) {
        Vehiculo
        .find({   // Filtramos los vehículos que hayan estado en
servicio entre
            "anio_alta": {$lte: req.params.anio},   // los años de
alta
            $or: [{"anio_baja": {$gte: req.params.anio}},
{"anio_baja":  null}]    // y de baja.
        })
        .exec(
            function(err, vehiculos) {
                if (vehiculos.length == 0) {
                    return res
                    .status(200)
                    .send({"message": "vehicles not found"});
                } else if (err) {
                    return res
                    .status(404)
                    .send(err);
                }
                return res
                .status(200)
                .send(vehiculos);
            }
        );
    } else {
        return res
        .status(404)
        .send({"message": "No year in request"});
    }
};
// Controlador para introducir un nuevo vehículo en la base de datos
module.exports.vehiculoCreate = function(req, res) {
    Vehiculo
    .create({   // Creamos un vehículo asignando a cada campo
        tipo: req.body.tipo,    // su correspondiente de la
petición.
        marca: req.body.marca,
        modelo: req.body.modelo,
        matricula: req.body.matricula,
        anio_alta: req.body.anio_alta,
        uso: req.body.uso,
```

```
        institucion: req.body.institucion,
        regimen: req.body.regimen,
        estado: req.body.estado,
        anio_baja: req.body.anio_baja
    },function(err, book) {
        if (err) {  // Si hay un error, se manda mensaje con código
400.
            return res
            .status(400)
            .send(err);
        }
        return res  // Si se guarda correctamente, se devuelve
código 201.
        .status(201)
        .send(book);
    });
};

// Controlador para borrar de la base de datos un vehículo con
cierta matrícula.
module.exports.vehiculoDelete = function(req, res) {
    if (req.params && req.params.matricula) {
        Vehiculo
        .findOneAndDelete({matricula: req.params.matricula}) //
Encontramos y
        .exec(          // borramos el vehículo con la matrícula
introducida.
            function(err, vehiculo) {
                if (err) {  // Si no se puede borrar, se devuelve
error 400.
                    return res
                    .status(400)
                    .send(err);
                }
                return res  // Si se borra correctamente,
                .status(204)    // se devuelve código 204.
                .send(null);
            }
        );
    } else {
        return res  // Si no se pasa matrícula,
        .status(404)    // se devuelve código 404 con mensaje.
        .send({"message": "No matricula in the request"});
    }
};

// Controlador para actualizar los datos un vehículo con cierta
matrícula.
module.exports.vehiculoUpdate = function(req, res) {
    if (req.params && req.params.matricula) {
        Vehiculo
```

```
        .findOne({matricula: req.params.matricula}) // Usamos el
vehículo con
        .exec(                                      // la matrícula
introducida.
            function(err, vehiculo) {
                if (!vehiculo) {
                    return res  // Si no se encuentra el vehículo,
                    .status(404)    // devolvemos 404 con mensaje.
                    .send({"message": "no vehicle found"});
                } else {    // Actualizamos con el contenido de los
campos
                    if (req.body.tipo) {    // pasados por
parámetro.
                        vehiculo.tipo = req.body.tipo;
                    }
                    if (req.body.marca) {
                        vehiculo.marca = req.body.marca;
                    }
                    if (req.body.modelo) {
                        vehiculo.modelo = req.body.modelo;
                    }
                    if (req.body.anio_alta) {
                        vehiculo.anio_alta = req.body.anio_alta;
                    }
                    if (req.body.uso) {
                        vehiculo.uso = req.body.uso;
                    }
                    if (req.body.institucion) {
                        vehiculo.institucion = req.body.institucion;
                    }
                    if (req.body.regimen) {
                        vehiculo.regimen = req.body.regimen;
                    }
                    if (req.body.estado) {
                        vehiculo.estado = req.body.estado;
                    }
                    if (req.body.anio_baja) {
                        vehiculo.anio_baja = req.body.anio_baja;
                    }
                    vehiculo.save(function (err, vehiculo) {
                        if (err) {  // Si hay algún error,
                            return res  // devolvemos 404 con
mensaje.
                            .status(404)
                            .send(err);
                        }
                        else {  // Si se ha realizado la operación
                            return res  // correctamente, devolvemos
200 con la
                            .status(200)    // info. del vehículo.
                            .send(vehiculo);
```

```
                        }
                    });
                }
            }
        );
    } else {
        return res  // Si no se pasa parámetro, devolvemos 404 con
mensaje.
        .status(404)
        .send({"message": "No matricula in the request"});
    }
};
```

Código 11: Implementación de los controladores de un vehículo (vehiculo.js).

UNIVERSIDAD DE ALMERÍA

Resumen/Abstract

Los datos abiertos que publican todo tipo de organizaciones pueden ser beneficiosos para toda la sociedad, ya que pueden aportar un gran valor de diferentes maneras. Estos datos abiertos se ponen a disposición de la sociedad a través de sitios web denominados “portales de datos abiertos”, desde los cuales se pueden descargar en forma de conjuntos de datos en una amplia variedad de formatos. En este trabajo, hablaremos en profundidad sobre todo lo que rodea a los datos abiertos, y se construirá un portal de datos usando CKAN, un software de código abierto para tal fin.

A través de este portal, se pueden servir conjuntos de datos ubicados en otro servidor. Es por eso que también se construirá una API REST para interactuar con una base de datos la cual contiene el conjunto de datos que se desea publicar en el portal de datos abiertos. Esta API se construirá utilizando la pila MEAN. Las comunicaciones entre los diferentes componentes de la pila MEAN se realizan usando JSON como formato, obteniendo las respuestas a las peticiones HTTP en este formato. Será en este formato en el que publiquemos el conjunto de datos y subconjuntos en el portal de datos abiertos.

Open Data published by organisations of every type can be benefitial for all the society, since they can add value in different ways. Open Data is placed at the disposal of the society via websites called “Open Data portals”, from which can be downloaded in form of datasets in a wide variety of formats. In this work, we will talk in depth about everything that surrounds Open Data, and a Open Data portal will be built using CKAN, an open source software for that purpose.

Through that portal, datasets located in another server can be served. That is why a REST API for interacting with a database which contains the dataset that is wanted to be published in the Open Data portal will be built too. This API will be built using the MEAN stack. Communications between different components of the MEAN stack are carried out using JSON as its format, getting reponses to HTTP requests in that format. It will be in this format in which we publish the dataset and subsets in the Open Data portal.