

WEB SCRAPING

JORGE PEREZ RECUERO



I.E.S RAMÓN DEL VALLE INCLÁN

2º DAM - CURSO 2019/2020



Índice - Contenido

1. INTRODUCCIÓN	4
1.1 Aspectos generales y motivación.	4
1.2 Objetivos del proyecto	4
1.3 Pasos a seguir	5
2. SOLUCIONES EXISTENTES	5
2.1 Caso de uso general	6
2.2 Portales Web	6
2.2.1 Coches.net	6
2.2.2 Wallapop	7
2.2.3 MilAnuncios	7
2.3 Conclusiones de este apartado	7
3. EJEMPLO PRACTICO: APP Web Scraping	8
3.1 Microservicios	8
3.2 Vista con Angular	9
3.3 Estructura de la APP	9
3.4 Modelo	10
3.5 Servicios.	10
3.5.1 Web Scraping	10
3.5.2 API Gateway	10
3.5.3 Service Discovery	10
3.5.4 Server Config	11
4. DESARROLLO DE LA APP	12
4.1 Entorno de Desarrollo	12
4.1.1 IntelliJ IDEA - JetBrains	12
4.1.2 Apache - Maven	13
4.1.3 Spring Framework - Pivotal	15
4.1.4 Postman - Postdot Technologies	16
4.2 Proyecto Maven	17
4.3 Spring Boot	19
4.4 Spring Boot - API Rest	20
4.5 Spring Boot - API Documentation	23
4.6 Modelo	24
4.7 Service Discovery	27
4.8 Spring Cloud Gateway	28
4.9 Spring Cloud Config Server	29
4.10 Angular	31
4.10.1 Descarga de AngularCLI e iniciar proyecto	32

	<p style="text-align: center;">TFG WEB SCRAPING 2º DAM. Desarrollo Aplicaciones Multiplataforma. CURSO 19-20</p>	
---	--	---

5. DESPLIEGUE DE LA APP	33
6. CONCLUSIÓN	33
7. REPOSITORIO GITHUB	34

1. INTRODUCCIÓN

En esta primera sección se introduce tanto la motivación que ha llevado a hacer este proyecto como los objetivos llevados a cabo.

1.1 Aspectos generales y motivación.

Este proyecto surge por la cantidad de páginas de compra-venta de productos de segunda mano de cualquier tipo que podemos encontrar en internet, junto con la necesidad de las personas de encontrar la mejor oferta a la hora de comprar o vender los productos.

Existen actualmente multitud de portales web donde encontraremos un producto similar de segunda mano en cada una de ellas con un precio distinto. Si un usuario quiere saber si un producto tiene un precio competitivo o por debajo del precio de mercado, tendrían que estar constantemente navegando en internet (portales web y aplicaciones móviles) buscando la mejor oferta. Ahí es donde entra en juego este proyecto, consiste en la creación de una aplicación que recopile de forma automática información de las distintas web de compra-venta, las analice, y las recoja en una lista de las oferta disponible para el producto del que está interesado.

1.2 Objetivos del proyecto

El objetivo principal de este trabajo fin de grado es crear una aplicación de microservicios que utilizando un servicio de Web Scraping, facilite a los usuarios el encontrar una oferta para el producto que está buscando sin la necesidad de estar constantemente navegando y comprobando las ofertas disponibles Internet.

La idea principal es unificar la información que se encuentra en Internet para que el usuario solo tenga que ver la información del producto que quiere y dónde se encuentra para poder adquirirlo.

Los objetivos teóricos que voy a desarrollar son:

- **Arquitectura de Microservicios:**
 - Conceptos clave
 - Service Discovery
 - Gateway
 - Server Config
 - Ventajas
 - Inconvenientes
- **Api REST**
- **Spring Boot y Spring Cloud**
- **Web Scraping - JSoup y Selenium**
- **Angular**

Todo ello será estudiado e investigado a través del desarrollo de una aplicación basada en Microservicios, que se tomará como ejemplo práctico y a través de la cual se intentaran entender los conceptos clave de esta arquitectura, y su posición actual y a futuro en el mundo Software.

1.3 Pasos a seguir

El desarrollo del proyecto se estructura de la siguiente forma:

- Recopilación de información sobre los frameworks..
- Análisis de las distintas aplicaciones disponibles para extracción de información en internet (web scraping).
- Estudio del lenguaje de programación HTML y JavaScript para implementar la aplicación, como lenguaje base de la mayoría de estas aplicaciones de scraping.
- Investigación y creación de los servicios de Web Scraping.
- Diseño de la estructura de la aplicación.
- Desarrollo de la aplicación.
- Pruebas de validación y usabilidad.
- Creación de la memoria del proyecto.

2. SOLUCIONES EXISTENTES

En este apartado veremos algunos ejemplos de aplicaciones y portales web en las que podremos comprar y vender productos de segunda mano.

Como he mencionado anteriormente podemos encontrar un sin fin de aplicaciones web y móviles de compra/venta de productos de segunda mano, como por ejemplo coches.net, milanuncios.com, wallapop.com, etc. Pero estas web son independientes entre sí, y no existe ninguna en la que se pueda recopilar la información de varias de ellas y proporcionárselas al usuario.

Por este motivo surgió la idea de este proyecto para facilitar a las personas y que únicamente tengan que consultar una APP en vez de estar mirando constantemente las múltiples web y apps disponibles.

2.1 Caso de uso general

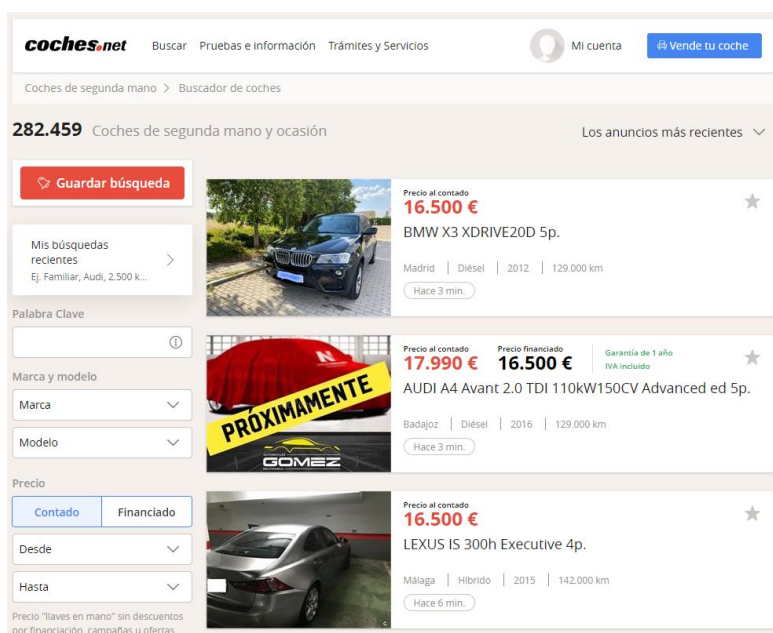
Antonio está buscando comprar un coche de segunda mano pero no tiene muy claro que coche quiere, lo que sí tiene claro es que quiere encontrar una buena oferta. Por lo tanto Juan entra en varios sitios web y se instala numerosas aplicaciones de coches de segunda mano para ir viendo que coches les llama la atención e ir comparando el precio de todos los sitios que va visitando. Tiene que hacer esto para quedarse con el mejor coche al menor precio posible, con lo que va acumulando varias apps instaladas en su teléfono y varios sitios web almacenados.

2.2 Portales Web

Vamos a ver ejemplos de portales web de compra-venta de productos de segunda mano cómo se estructuran, y cómo funcionan.

2.2.1 Coches.net

Este primer portal web es únicamente de compra-venta de coches. En su página inicial disponemos de varias secciones donde acceder a productos o información de la web. El apartado principal que describiremos nos permite ver un listado de coches de segunda mano o también permite hacer un filtrado por modelo, marca y varias características que un usuario este buscando en específico.



Como vemos, en el apartado coches de segunda mano, nos aparece un listado de coches juntos con su información más relevante que a un usuario le pueda interesar.

2.2.2 Wallapop



En este portal web la página de inicio de wallapop.com como vista principal nos aparecen distintos productos que se están vendiendo junto con su información correspondiente. Una vez buscamos un producto nos aparece varias imágenes del producto y distinta información que le será útil al usuario.

2.2.3 MilAnuncios



En esta web en su página principal directamente nos aparecen las distintas categorías de productos disponibles y un buscador por si queremos buscar un producto en concreto. Al seleccionar una de las categorías o poner un producto en el buscador, nos llevará a un listado de los productos disponibles y su información..

2.3 Conclusiones de este apartado

Tras revisar los portales web y apps existentes, la principal conclusión es que para un usuario que comprar un artículo es complicado y le llevará tiempo revisar todas las opciones disponibles actualmente. Tendrá que hacer una labor importante de comparación de características para conocer si el precio establecido es un buen precio. Por ello, con esta comparación se justifica el motivo de este TFG.

3. EJEMPLO PRÁCTICO: APP Web Scrapping

3.1 Microservicios

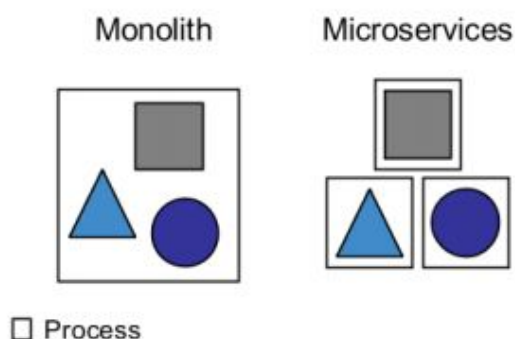
Hasta ahora, los sistemas se han estructurado mediante grandes aplicaciones en forma de monolitos, haciendo que el mantenimiento y la evolución sea demasiado compleja conforme estos proyectos se agrandan. En frente a esta tecnología, tenemos a los microservicios, que consisten en una arquitectura software que implementa servicios mediante la colaboración de otros servicios más pequeños y autónomos.

Cada microservicio se centra en una parte individual del modelo negocio. Así, se consigue la abstracción del resto del sistema de los detalles concretos de la implementación.

Además, esto permite que el despliegue de estos servicios pueda realizarse de forma individual y separada, por lo que el usuario final no tiene que ser conocedor de qué servicios son los que le están ofreciendo los recursos que está pidiendo. Esto, favorece, entre otras cosas, la escalabilidad.

Así pues, podemos enumerar ciertas ventajas de utilizar una arquitectura basada en microservicios para nuestra aplicación:

- Facilita la escalabilidad y la hace más eficiente
- La capacidad de prueba de cada microservicio es mayor.
- El mantenimiento se hace más sencillo y eficiente.
- Permite que en un mismo proyecto coexistan distintas tecnologías:
- Múltiples lenguajes, bases de datos, protocolos, etc.
- Hace mucho más sencillo el desarrollo paralelo de las distintas funcionalidades de la aplicación y, además, permite que este desarrollo sea independiente.
- Permite que los despliegues sean independientes, lo que aumenta la tolerabilidad a fallos graves.



3.2 Vista con Angular

Angular es un framework de código abierto desarrollado por Google. Con el objeto para crear aplicaciones web dinámicas y modernas. Fue presentado por primera vez en 2009, el framework se caracteriza por eliminar códigos innecesarios y garantizar aplicaciones más ligeras y rápidas.

Angular ayuda a crear aplicaciones interactivas y dinámicas de una sola página. Estas aplicaciones incluyen plantillas, enlace con datos, modularización, gestión de API RESTful, inyección de dependencias y manejo de AJAX.

Un frontend con Angular no depende de bibliotecas de terceros. Al utilizar este framework en sus proyectos, puede obtener múltiples beneficios, a continuación vamos a ver estas ventajas.

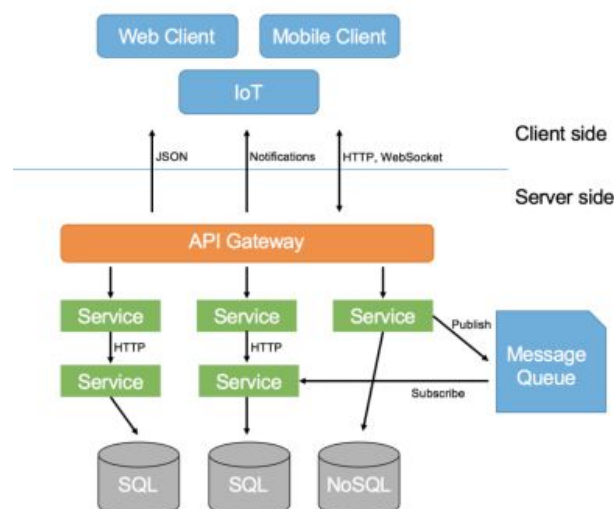
Las principales ventajas que podemos enumerar son:

- Promovido por Google: Las aplicaciones de Google también utilizan este framework. Su estabilidad parece garantizada.
- Las pruebas son extremadamente simples. Los módulos soportan partes de la aplicación, que son fáciles de manipular.
- Patrón MVC simplificado.
- Angular garantiza un desarrollo fácil, ya que elimina la necesidad de código innecesario.

3.3 Estructura de la APP

La arquitectura basada en microservicios puede tener tantas capas, servicios o componentes, como se requieran. Generalmente, las capas básicas de esta arquitectura son: API Gateway, Service Discovery, Servicios y sus correspondientes bases de datos.

Un ejemplo de arquitectura de aplicaciones basadas en microservicios pueden ser:



3.4 Modelo

El modelo de nuestra APP será bastante sencillo ya que solo trabajaremos un solo tipo de objeto:

Anuncios: Son aquellos objetos los cuales que vamos a obtener a raíz de los datos “scrapeados” de las diferentes páginas webs. Los cuales, los “empaquetaremos” en una lista.

3.5 Servicios.

3.5.1 Web Scraping

- **Descripción:** Este servicio soporta las peticiones, en las cuales recibirá los datos del anuncio.
- **Tecnología:** JSoup, Selenium, Api Rest, Spring-Boot, Spring-MVC, Lombok, Java 8.
- **Funcionalidad:** Realizar el scrapeo en las páginas web y construir una lista con todos los anuncios encontrados.
- **Modelo:** Trabaja con objetos de tipo Anuncio

3.5.2 API Gateway

- **Descripción:** Servicio encargado de centralizar las llamadas a los demás servicios a través de una URI que hace de entrada de peticiones.
- **Tecnología:** Api Rest, Spring-Boot, Spring-Cloud Gateway sobre Java 8.
- **Funcionalidad:** Se encarga de centralizar las llamadas a la aplicación en una URI principal que redirige las llamadas a los servicios configurados internamente.

3.5.3 Service Discovery

- **Descripción:** Servicio encargado de registrar las direcciones a los microservicios que componen la aplicación y redireccionar las peticiones hacia estos.
- **Tecnología:** Spring-Boot, Spring-Cloud y Netflix Eureka sobre Java 8
- **Funcionalidad:** Encargado de redirigir las llamadas realizadas a cada servicio a través de una URI genérica a la dirección del servidor en el que se encuentra dicho servicio.

3.5.4 Server Config

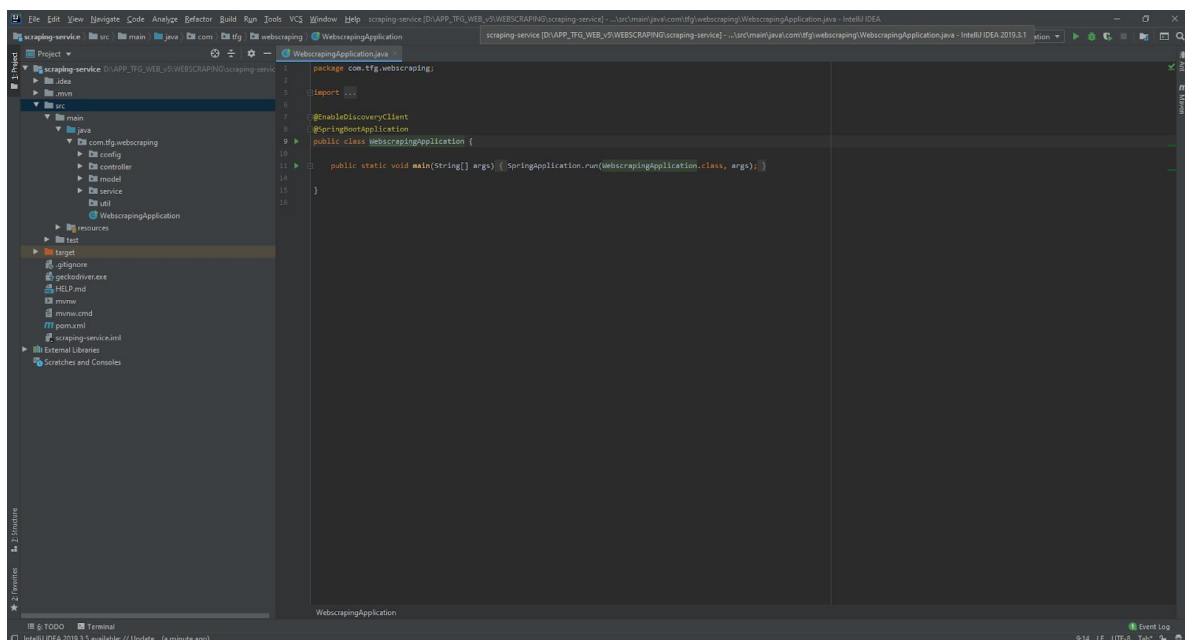
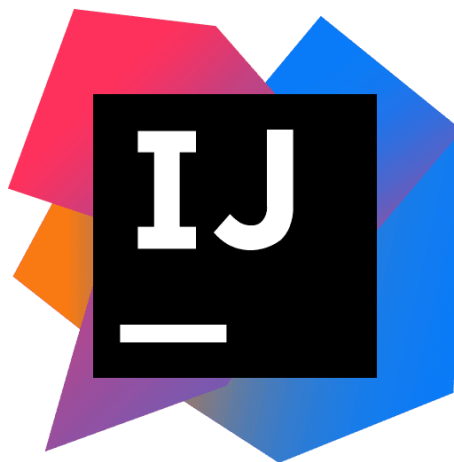
- **Descripción:** Este servicio está a la espera de recibir peticiones por parte del resto de servicios, para que se les proporcione la configuración necesaria para su ejecución en el entorno definido. Cuando este servicio recibe estas peticiones, recoge los archivos de configuración para cada servicio (de extensión “.yml” o “.properties”) de un repositorio definido para luego proporcionárselo al servicio que haya realizado la petición.
- **Tecnología:** Spring-Boot y Spring-Cloud: Spring Cloud Config Server sobre Java 8
- **Funcionalidad:** Busca sobre un repositorio los archivos de configuración de cada servicio que los solicite.

4. DESARROLLO DE LA APP

4.1 Entorno de Desarrollo

4.1.1 IntelliJ IDEA - JetBrains

Para el desarrollo de esta aplicación se utilizará el reconocido IDE IntelliJ. Esto se debe a que posiblemente se trata del IDE que, actualmente, incorpora más y mejores herramientas para el desarrollo de aplicaciones web. En este caso, además, se trata de un IDE con total compatibilidad con el desarrollo de microservicios.



Web: <https://www.jetbrains.com/idea/>

4.1.2 Apache - Maven

Maven nace como una herramienta dedicada a facilitar la compilación y generación de ejecutables para aplicaciones. Así como para facilitar la gestión de dependencias y librerías.

Actualmente, para todo desarrollador, es necesario tener en el desarrollo de sus proyectos un gestor de dependencias y librerías. Esto se debe a que una de las grandes premisas de la programación se basa en la no repetición de código, la generalización, la parametrización y la modularidad.

En este caso se utilizará Maven como gestor de dependencias para nuestro proyecto, ya que es uno de los más potentes del mercado y con una gran versatilidad con respecto al desarrollo de microservicios.

La administración de las dependencias y librerías utilizadas en el proyecto se realiza a través de unos archivos con formato XML y al que nos referiremos como el archivo POM de la aplicación (POM.xml).

Un ejemplo de archivo POM utilizado en el desarrollo de esta aplicación, puede ser este:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.3.0.RELEASE</version>
  </parent>

  <groupId>com.tfg</groupId>
  <artifactId>webscraping</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>webscraping</name>
  <description>Spring project with Web Scraping</description>

  <properties>
    <java.version>1.8</java.version>
    <spring-cloud.version>Greenwich.SR1</spring-cloud.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>2.2.6.RELEASE</version>
    </dependency>

    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <scope>runtime</scope>
    </dependency>
  </dependencies>
</project>
```

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>

<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.0.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.jsoup/jsoup -->
<dependency>
  <groupId>org.jsoup</groupId>
  <artifactId>jsoup</artifactId>
  <version>1.11.3</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>3.141.59</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-test -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <version>2.2.6.RELEASE</version>
  <scope>test</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-config -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
  <version>2.2.3.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-eureka-client -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
  <version>2.2.3.RELEASE</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-netflix-ribbon -->
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-ribbon</artifactId>
  <version>2.2.3.RELEASE</version>
</dependency>

</dependencies>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

</project>
```

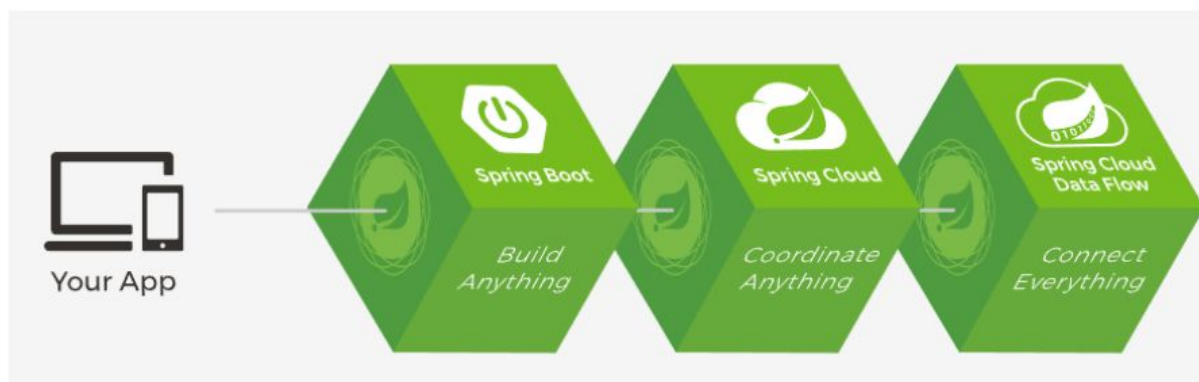
Web: <https://maven.apache.org/>

4.1.3 Spring Framework - Pivotal



Spring nace como sustituto, o complemento, al modelo EJB de Java referente a la plataforma Java EE. Es decir, fue diseñado con la intención de agilizar el desarrollo de aplicaciones empresariales.

Tras más de una década de evolución y desarrollo hoy en día Spring ofrece una serie de módulos que pueden trabajar, o no, de forma conjunta y que conforman el llamado “Spring Framework”.



Entre los módulos más relevantes para este proyecto, encontramos:

- **Spring Boot:** Se trata del módulo que permite y arrancar y configurar de forma rápida y sencilla aplicaciones basadas en Spring.
- **Spring Cloud:** Da a los desarrolladores herramientas para crear de forma rápida y sencilla aplicaciones distribuidas basadas en entornos en la nube y facilita la configuración de forma centralizada y “cloud-native”.
- **Spring Data:** Su propósito es unificar y facilitar el acceso a distintos tipos de tecnologías de persistencia, tanto a bases de datos relacionales como a las del tipo NoSQL.

Web: <https://spring.io/>

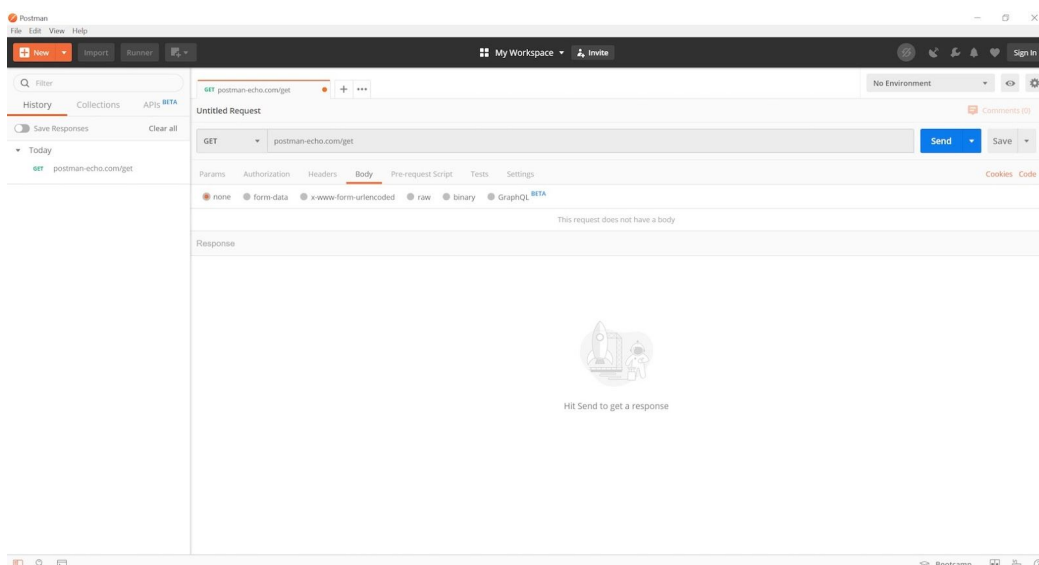
4.1.4 Postman - Postdot Technologies



Postman nace como una extensión para el navegador web Google Chrome con la propuesta de hacer más fácil a los desarrolladores realizar pruebas con aplicaciones web realizando peticiones específicas y analizar sus respuestas.

Postman se compone de diferentes herramientas que permiten realizar tareas diferentes con las API REST. Desde la creación de peticiones a APIs, tests de validación del comportamiento de éstas, la posibilidad de crear diferentes entornos de trabajo y parametrizarlos y, además, ofrece la posibilidad de hacer todas estas tareas junto con tu equipo de desarrollo, ofreciendo la posibilidad compartir tus datos con otros compañeros y de exportar los mismos.

Actualmente Postman se presenta como una aplicación de escritorio que permite realizar todas estas tareas de una forma más potente y con la integración de ciertos servicios en la nube.



Web: <https://www.postman.com/>

4.2 Proyecto Maven

Este proyecto se construirá utilizando el administrador de librerías y dependencias Maven. Por lo tanto, primero, han de configurarse adecuadamente los metadatos de la aplicación como son el Group ID o el Artifact ID de cada componente o módulo, entre otros.

En nuestro caso, el patrón que se utilizará como group-id en todos los proyectos es el siguiente: “com.tfg”. Donde “com” es lo predeterminado, pero para ser más correcto debería ser las siglas o unas señas de la empresa, por ejemplo, “dam” que corresponde a “Desarrollo de Aplicaciones Multiplataforma” y “FTG” corresponde al nombre del desarrollador o equipo de trabajo. Finalmente, el artifact-id hace referencia al identificador que Maven utilizará para resolver las dependencias.

Project Metadata

Group	com.tfg
<hr/>	
Artifact	register-server
<hr/>	

Para conocer qué dependencias necesita nuestro proyecto, es posible realizar búsquedas sobre qué tipos de librerías se ofrecen que nos puedan resultar de utilidad. Luego, solo será necesario copiar sus descriptores de Maven y colocarlos en los archivos POM de cada módulo.

Por ejemplo, las dependencias básicas para crear una aplicación Spring Boot las podemos encontrar en la web principal de Spring o en el repositorio de Maven Central. Para crear una aplicación básica con Spring Boot, tan solo necesitaremos la siguiente dependencia en el POM de nuestra aplicación:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
  <version>2.2.6.RELEASE</version>
</dependency>
```

Además, será necesario establecer nuestra aplicación como hija de una aplicación Spring Boot. Para ello, solo habrá que añadir las siguientes líneas:

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.3.0.RELEASE</version>
</parent>
```

Por último, será necesario establecer el plugin de Spring Boot a través del cual se compilará nuestro proyecto y el cual permitirá a Spring analizar el proyecto en busca de las dependencias y los javabeans necesarios para la ejecución de su contexto:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Todas las librerías disponibles las podemos encontrar en la web oficial de Maven: <https://mvnrepository.com/>. Además, existen otras dos webs muy útiles para nuestro caso:

Search Maven: Una web creada por la comunidad que facilita la búsqueda de librerías actualizadas y funcionales en el repositorio de Maven - <https://search.maven.org/>



Spring Initializr: Se trata de una herramienta oficial de los desarrolladores del framework Spring, a través de la cual se puede inicializar un proyecto sencillo con la mayoría de las librerías esenciales para el uso del framework - <https://start.spring.io/>



4.3 Spring Boot



Spring Boot nos ofrece crear aplicaciones Spring sin tener que realizar el tedioso proceso de configuración. No es necesario hacer ningún fichero de configuración.

Por ejemplo, para la creación de una aplicación web, Spring Boot, automáticamente nos proporcionará un servidor Tomcat embebido el cual estará configurado y conectado junto con un servlet propio de Spring. Además, por supuesto, toda la configuración por defecto será personalizable.

Podemos resumir el éxito de Spring Boot en las siguientes características:

Configuración: Spring Boot cuenta con un complejo sistema de contexto de aplicación, que escanea y analiza la aplicación dividiéndola por componentes en tiempo de compilación y ejecución. Spring Boot es capaz de autoconfigurar todos los aspectos de una aplicación y permitir ejecutarla sin necesidad de definir absolutamente nada.

Gestión de dependencias: Tan sólo es necesario indicar el tipo de proyecto que se desarrollará Spring Boot será el encargado de resolver las dependencias necesarias entre módulos y componentes para que la aplicación funcione.

Despliegue: Spring Boot es capaz de ejecutarse tanto como aplicación Stand-alone como aplicación web mediante su servidor web integrado.

Modularidad: Spring Boot se basa en una estructura modular independiente que permite que la comunidad diseñe y proponga nuevos módulos para integrarlos en su desarrollo.

Mediante Spring Boot tenemos la posibilidad de crear aplicaciones completas ejecutables en pequeños entornos para los cuales ya no es necesario tener un servidor de aplicaciones completo junto con una gran cantidad de recursos.

Spring boot, además, tiene complementos muy interesantes como Spring Cloud con los cuales se nos permite crear aplicaciones “Cloud Native”, o lo que es lo mismo, aplicaciones diseñadas desde el inicio para ser ejecutadas en la nube.

Estas ventajas, son clave para la creación de microservicios, ya que hacen que no sea necesaria la instalación de servidores, así como la necesidad de realizar complejas configuraciones.

4.4 Spring Boot - API Rest

Llamamos API al conjunto de funciones y procedimientos que realizan distintas funciones con el fin de ser utilizadas por otro software. Este concepto se encuentra incluido en el concepto software de librería.

Las siglas API vienen del inglés Application Programming Interface, que en español sería Interfaz de Programación de Aplicaciones. Esto se debe a que una API nos permite implementar las funciones y procedimientos que engloba en nuestro proyecto sin la necesidad de programarlas de nuevo.

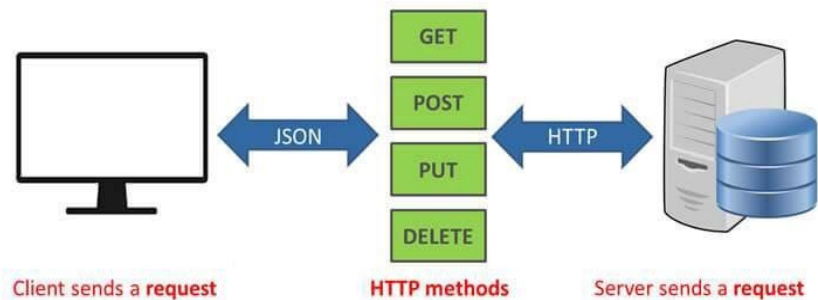
En términos de programación, es una capa de abstracción. Por otro lado, API REST se define como una librería que se basa totalmente en el estándar de comunicación HTTP. Dicho de otra forma, una API REST es un servicio que nos aporta funciones heredadas a través de un servicio web que no es nuestro, dentro de una aplicación propia, de manera segura.



REST, REpresentational State Transfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Por ello, al ser un estándar de red, requiere de ciertas restricciones que permitan la correcta comunicación entre usuarios. Estas restricciones son:

- Conexión cliente-servidor libre; El cliente no necesita saber los detalles de la implementación del server, y este tampoco debe preocuparse por cómo se usan los datos que envía.
- Cada petición enviada al servidor es independiente.
- Compatibilidad con un sistema de almacenamiento en caché.
- Cada recurso del servicio REST debe tener una única dirección, manteniendo una interfaz genérica.
- Permite utilizar diferentes capas para la implementación del servidor.

En nuestro caso, los servicios desarrollados siguen este patrón, ya que a día de hoy, **REST** y **JSON**, son la combinación más utilizada para el desarrollo de APIs, y la que un mayor apoyo de parte de la comunidad poseen.



Con esto, además, Spring Framework posee toda una serie de librerías, módulos y funcionalidades a disposición de los desarrolladores para crear nuevas APIs REST. Para nosotros, algunas de estas características son:

- La anotación **@RestController**, que indica que la clase Java que la contenga será la encargada de manejar las **peticiones HTTP** que nuestra aplicación recibirá, y que, además, nuestra aplicación ofrecerá respuestas acordes a las restricciones REST. En nuestro caso, como se ha indicado anteriormente, se ofrecerán respuestas **JSON**.
- Spring nos ofrece una serie de anotaciones para indicar al controlador REST qué métodos Java manejarán qué peticiones HTTP. Estas peticiones HTTP, pueden ser desde **GET, POST, PUT PATCH, OPTIONS, DELETE**, etc. Por lo tanto, Spring nos ofrece las siguientes anotaciones para cada operación: **@GetMapping**, **@PostMapping**, **@PutMapping**, **@DeleteMapping**, etc.
- Es recomendable, que los métodos de la clase "Controller" devuelvan objetos del tipo **ResponseEntity<T>**, donde "T" hace referencia al tipo de objeto que será transformado a JSON a través del traductor interno de Spring, llamado Jackson. Estos objetos, además de incluir el objeto Java a traducir como respuesta, nos permiten personalizar las cabeceras HTTP para dar unas respuestas más complejas y completas.

Como ya hemos dicho, REST nos permite implementar los métodos HTTP y es fácil observar, que existen similitudes entre estos y las operaciones de una interfaz CRUD. Esto se debe a que realmente, una petición GET estará asociada a un query SELECT, una petición POST estará asociada a un query INSERT, una petición PUT a un UPDATE y así sucesivamente. Esto no tiene por qué ser siempre así, ya que por ejemplo, podría darse el caso de que una API consulte información de los recursos o procese ciertos datos para dar una respuesta sencilla. Esto dependerá de las necesidades del proyecto, pero en general funciona de la manera antes mencionada.

Para nuestro desarrollo, podemos ver cómo queda definido una interfaz API y su implementación “Controller”:

```
@RequestMapping(EndPointUri.Api.Anuncios.ANUNCIO)
public interface AnuncioApi {

    @PostMapping
    ResponseEntity<List<AnuncioDTOSalida>> findAllCar(AnuncioDTOentrada carDTO);

}
```

En la Interfaz AnuncioAPI, definimos los métodos que manejará nuestro servicio junto con las anotaciones pertinentes, con la intención de que se puedan abordar varias implementaciones si fuera necesario.

```
@CrossOrigin(origins = "http://localhost:4200", maxAge = 3600)
@RestController
public class AnuncioController implements AnuncioApi{

    @Autowired
    private AnuncioService anuncioService;

    @Override
    @PostMapping("/findallcar")
    public ResponseEntity<List<AnuncioDTOSalida>> findAllCar(@RequestBody AnuncioDTOentrada datosDTOentrada) {

        List<AnuncioDTOSalida> listaAnuncios = anuncioService.getAllAnuncios(datosDTOentrada);

        if (listaAnuncios.size() == 0 || listaAnuncios == null) {
            System.out.println("entra aqui");
            return ResponseEntity.notFound().build();
        } else {
            return ResponseEntity.status(HttpStatus.FOUND).body(listaAnuncios);
        }
    }
}
```

4.5 Spring Boot - API Documentation

Para la documentación de estas APIs, es recomendable utilizar librerías ya existentes que permiten acceder a un índice con los métodos permitidos por la API, sus respuestas, requerimientos, accesibilidad, etc.

A día de hoy, una de las mayores librerías o herramientas que da este soporte es Swagger. Con Swagger es posible documentar una API de forma sencilla sin la necesidad de mucha configuración o escritura.

Para utilizar Swagger, tan solo es necesario incluir la dependencia principal a esta librería Maven en el POM principal del proyecto:

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
```

Además, será necesario activar esta utilidad en la clase ejecutable principal de nuestra aplicación, en este caso, tendríamos algo como:

```
@EnableSwagger2
```

Con esta dependencia nuestro proyecto contará con unos “endpoints” a través de los cuales cualquier persona que acceda a la dirección referente a Swagger, la cual es `*/v2/api-docs`, obtendrá toda la información en formato JSON sobre los métodos de la API en cuestión.

Por otro lado, tenemos una interfaz gráfica ofrecida por Swagger para documentar nuestra aplicación de forma más sencilla y dinámica, que nos ofrece toda esta información en una pantalla HTML a través de la dirección `*/swagger-ui.html`:

(foto interfaz grafica swagger)

Para el uso de esta herramienta, será necesario incluir la dependencia correspondiente en el POM:

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

4.6 Modelo

Se sigue el patrón usual para los modelos de las aplicaciones web. Se distinguen dos tipos de objetos diferenciados, que son los Value Objects y los Data Transfer Objects:

VO (Value Object): Son los datos que se pueden o se encuentran persistidos en la BD. Es decir, son el tipo de datos que se intercambian con la base de datos y que quizás requieran tener ciertas propiedades específicas para su manejo en la base de datos. Además, en el contexto de Spring, existen ciertas anotaciones que pertenecen al módulo de Spring Boot, llamado Spring Data, y que permiten configurar de forma sencilla este tipo de objetos.

```
@Entity
@Getter
@Setter
@ToString
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Anuncio {
    @Id
    @GeneratedValue
    private int id;
    private String marcaCar;
    private String modeloCar;
    private String titleCar;
    private String priceCar;
    private String anoCar;
    private String cvsCar;
    private String kmsCar;
    private String enlaceCar;
}
```

DTO (Data Transfer Object): Son la representación de los datos que se manejan en la aplicación para ser transferidos o recibidos como recursos en las peticiones realizadas hacia la aplicación. Solo son utilizados en el intercambio de datos en las peticiones web.

```
@Getter
@Setter
@Builder
@AllArgsConstructor
public class AnuncioDTOentrada {
    @NotBlank
    @JsonProperty("marca_car")
    private String marcaCar;
    @NotBlank
    @JsonProperty("modelo_car")
    private String modeloCar;
}
```

```
@Getter
@Setter
@Builder
@AllArgsConstructor
public class AnuncioDTOsalida {

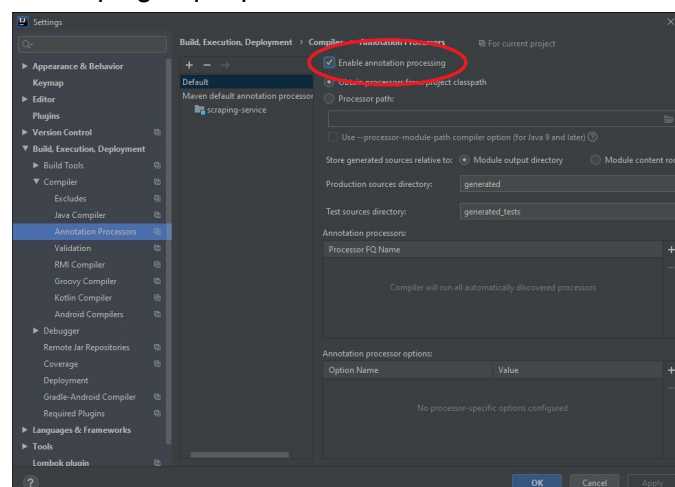
    @NotBlank
    @JsonProperty("title")
    private String titleCar;
    @NotBlank
    @JsonProperty("price")
    private String priceCar;
    @NotBlank
    @JsonProperty("ano")
    private String anoCar;
    @NotBlank
    @JsonProperty("cvs")
    private String cvsCar;
    @NotBlank
    @JsonProperty("kms")
    private String kmsCar;
    @NotBlank
    @JsonProperty("enlace")
    private String enlaceCar;
}
```

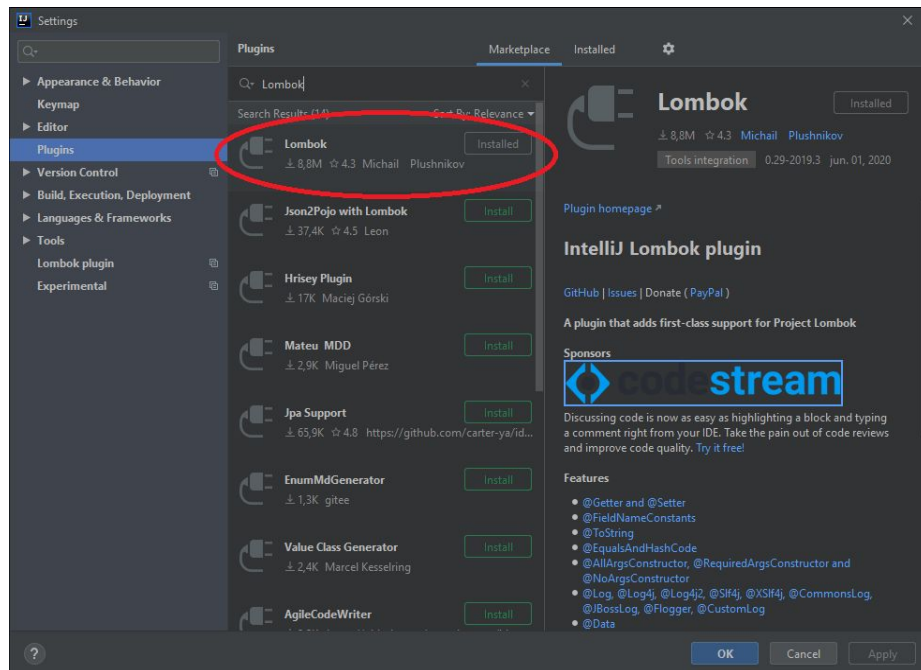
Para la creación de las clases de objetos POJO que definen nuestros objetos, se ha utilizado la herramienta LOMBOK. Lombok consiste en una librería para Java que nos ofrece, a través de anotaciones, reducir el código de las clases POJO de nuestra aplicación.

Para poder usar esta librería, ha de definirse como dependencia en el archivo POM.xml de la aplicación:

```
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

Además, generalmente, será necesario activar el procesador de anotaciones del IDE, así como, a veces, instalar un plugin que procese estas anotaciones:





Las anotaciones le indican a Lombok qué métodos debe generar escaneando los atributos de la clase donde haya sido utilizado en tiempo de compilación. Las anotaciones más importantes son:

@Getter: Genera los métodos “get” para los atributos definidos.

@Setter: Genera los métodos “set”

@AllArgsConstructor: Genera un constructor con todos los atributos como argumentos

@NoArgsConstructor: Genera un constructor sin argumentos **@Builder** Permite utilizar el patrón “builder” para crear nuevos objetos, como por ejemplo:

Generalmente, los datos se intercambian con formato “JSON” o “XML” entre otros. Luego, estos datos son manejados por el controlador de la API y son traducidos al contexto del lenguaje orientado a objetos.

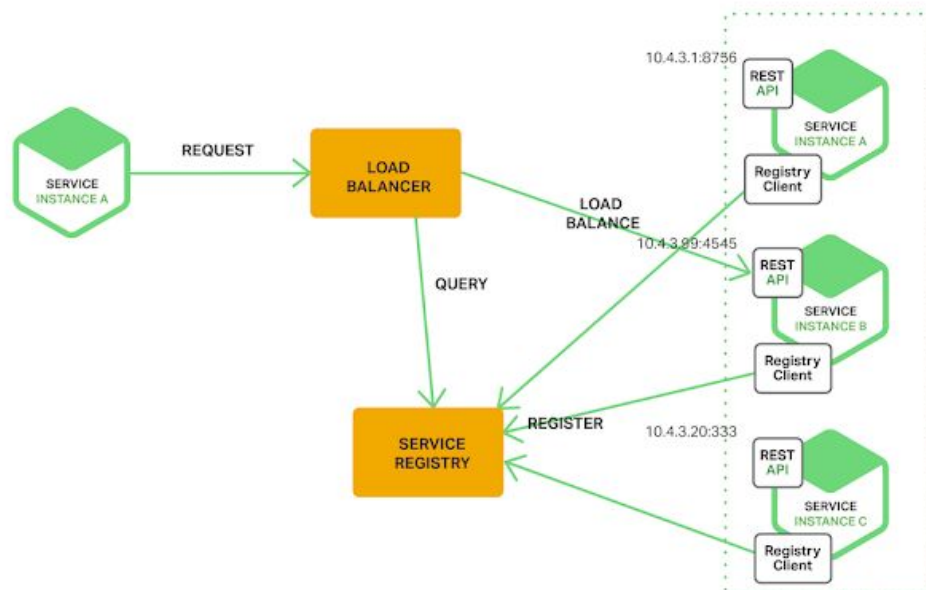
Más tarde, interiormente, los servicios encargados manejar las peticiones llegadas al controlador del API, transforman y manejan esos objetos según las necesidades de la aplicación.

Por ello, la traducción de los datos de **DTO** a **VO** se ha de realizar en la capa de Servicio de nuestra aplicación. Esta recibirá los DTOs del controlador y guardará los objetos VO a través de la capa de datos o repositorio.

4.7 Service Discovery

El Service Discovery es un proyecto en sí mismo basado en Spring Boot y Spring Cloud. Lo que hace es resolver las peticiones a los servicios de una aplicación compuesta por microservicios. Esto quiere decir, que es el encargado de llamar a las direcciones (endpoints o uris) de cada servicio.

Dicho de otra forma, será el que tenga la “libreta de direcciones” de los demás servicios que componen la aplicación. Es el encargado de buscar a qué dirección corresponde el servicio que se está tratando de consumir y ofrecer dicho servicio a la petición tras ser recibida.



Para ello, ha de crearse un proyecto Spring que tenga las dependencias de Spring Cloud dentro. Además, en el main de la aplicación Spring Boot, ha de aparecer la siguiente anotación: **@EnableEurekaServer**

Su configuración principalmente se realiza en formato “.yml”, la cual es más limpia y más legible. Entre estas propiedades que se definen, se pueden configurar elementos del tipo:

server:

port: \${PORT:8761}

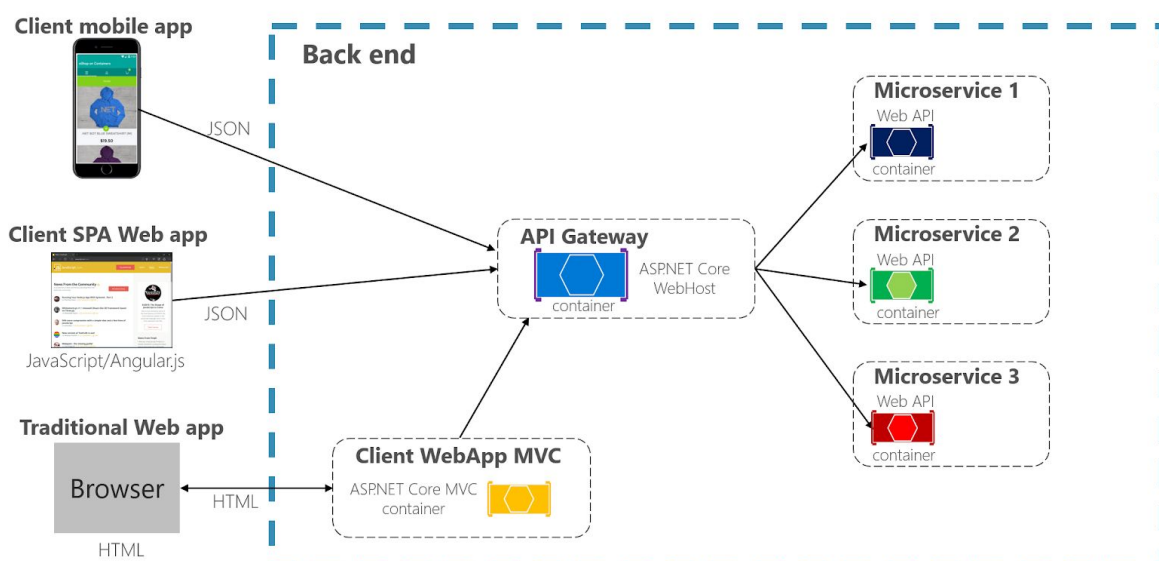
Donde la notación con el dólar, indica el uso de una variable global ya definida. En este caso, con la notación “DEFAULT:new”, se está indicando que primeramente se buscará la variable definida por defecto “PORT” para la propiedad server. En caso de no existir, los dos puntos indicar un operador lógico “OR” que indica qué puerto ha de usarse en caso de no existir uno por defecto ya definido. Esto nos sirve para parametrizar la configuración de nuestra aplicación.

4.8 Spring Cloud Gateway

Gateway es un servicio basado en Spring Boot y Spring Cloud. Resumidamente, se encarga de resolver las peticiones realizadas a la aplicación de forma centralizada y distribuirla entre los servicios de una aplicación compuesta por microservicios.

Como su nombre indica, hace de “portero” en la conexión entre las peticiones externas y los microservicios que componen nuestra aplicación.

Using a single custom **API Gateway service**



En nuestro desarrollo hemos implementado el Gateway Zuul para mantenernos dentro del stack de Spring. Este gateway ha sido desarrollado por la empresa Netflix.



Para la implementación del Gateway, como para otros servicios ya creados, primero se ha generado un proyecto inicializado desde la web spring-initializr. En este caso, es necesario añadir las dependencias referentes a la librería spring cloud y el artefacto gateway:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-zuul</artifactId>
</dependency>
```

A estas dependencias, debemos sumarles las nombradas anteriormente para que nuestro servicio solicite su configuración al servidor config de forma ordenada y, además, que se registre en nuestro Service Discovery.

```
server:
  port: ${GATEWAY_PORT:9090}

spring:
  application:
    name: api-gateway
  cloud:
    config:
      discovery:
        enabled: true
        service-id: config-server
```

4.9 Spring Cloud Config Server

Spring, ofrece una implementación del llamado Config Server muy útil y sencilla. En nuestro caso, se ha creado un proyecto con la herramienta antes mencionada “spring-initializr”, con las dependencias correspondientes a Spring Cloud, que es la librería padre que nos ofrece Spring Cloud Config Server:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

Una vez hecho esto, tendremos un proyecto sencillo con una clase main que se ejecutará el contexto de spring. A esta clase, habrá que añadirle obligatoriamente la siguiente anotación: **@EnableConfigServer**

Además, para este servicio es necesario que implementemos una configuración concreta en el archivo “bootstrap.yml”, que debe ser la siguiente:

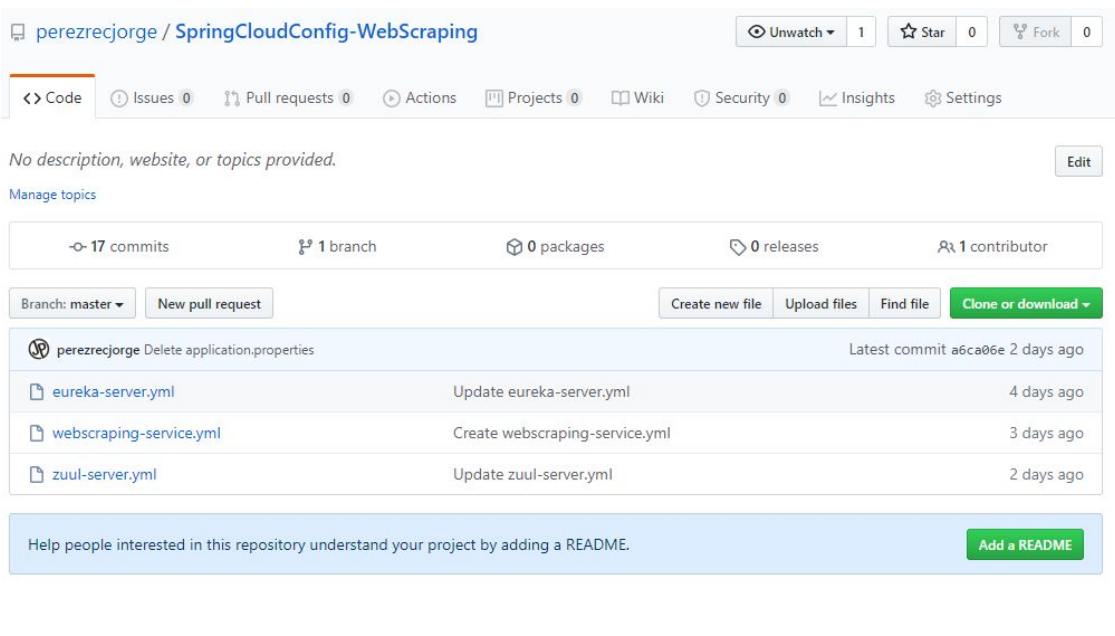
```
server:
  port: ${CONFIG_PORT:8888}

spring:
  application:
    name: config-server
  cloud:
    config:
      server:
        git:
          uri: ${CONFIG_REPO_URI:https://github.com/perezrecjorge/config-repo}
          username: ${CONFIG_REPO_USERNAME:perezrecjorge}
          password: ${CONFIG_REPO_PASSWORD: }

eureka:
  client:
    service-url:
      defaultZone: ${EUREKA_URI:http://localhost:${EUREKA_PORT:8761}/eureka}
    register-with-eureka: true
```

Donde podremos definir distintos repositorios asociados a los perfiles de la aplicación Spring. Para que cada servicio pueda encontrar su archivo de configuración, estos archivos deben de tener como nombre, el nombre del servicio y extensión “.yml” o “.properties”.

Se puede usar cualquier plataforma git; GitHub, GitLab, etc. En nuestro caso, se ha creado un repositorio en GitHub donde se contienen todos estos archivos:



The screenshot shows a GitHub repository page for 'perezrecjorge / SpringCloudConfig-WebScraping'. The repository has 1 commit, 1 branch, 0 packages, 0 releases, and 1 contributor. The commit history shows four commits: 'Delete application.properties' (2 days ago), 'Update eureka-server.yml' (4 days ago), 'Create webscraping-service.yml' (3 days ago), and 'Update zuul-server.yml' (2 days ago). There is a button to 'Add a README'.

4.10 Angular



Es un framework front-end de desarrollo para JavaScript creado por Google. El objetivo de Angular es facilitar el desarrollo de aplicaciones web SPA y proporcionar herramientas para trabajar de manera sencilla y óptima los elementos de una web.

Una Aplicación web SPA es una web de una sola página, lo que significa que la navegación entre las distintas secciones y páginas de la aplicación, así como la carga de los datos, se realiza de manera dinámica, de forma asíncrona llamando al servidor, y sin refrescar la página.

La arquitectura de angular está basada en componentes que realizan tareas concretas y colaboran entre sí para conseguir la funcionalidad deseada.

Podemos diferenciar dos categorías básicas de componentes:

- **Views:** Estos componentes se comunican con los servicios y son los encargados de pintar el layout de la página, o trozo de página, normalmente una página se renderiza con la composición de varios de estos componentes. Otra funcionalidad de estos componentes es de proporcionar datos a componentes más genéricos y actuar sobre los eventos que ocurren en los mismos.

- **Componentes:** Componentes reutilizables que se suelen encargar de mostrar información y/o interactuar con el usuario. Un ejemplo de estos componentes son los botones, listados, alertas, etc.

Web: <https://angular.io/>

4.10.1 Descarga de AngularCLI e iniciar proyecto

Para poder iniciar nuestro proyecto Angular en Windows, necesitamos descargar el Cliente de Angular de la siguiente página: <https://cli.angular.io/>

Posteriormente, abrir CMD (con permisos de administrador) y dirigirnos a la ruta donde queramos crear nuestro proyecto y ejecutar: `ng new my-proyecto-angular`

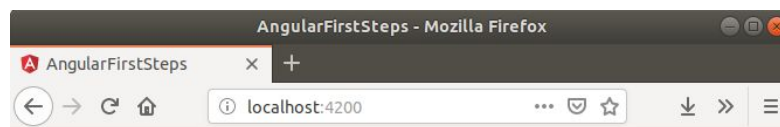
Ahora vamos al directorio en el que hemos creado el proyecto:

```
cd my-proyecto-angular
```

Y para ejecutar el proyecto Angular, ejecutaremos el siguiente comando dentro de la ruta de tu proyecto:

```
ng serve --open
```

Usando la opción `— open` (o simplemente `-o`) nos abrirá automáticamente la aplicación en el navegador en “http://localhost: 4200/”



Welcome to angular-first-steps!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)

5. DESPLIEGUE DE LA APP

Para poner en funcionamiento nuestro proyecto es muy sencillo, solo tenemos que seguir los siguientes pasos:

1. Ejecutar “register-server”
2. Ejecutar “config-server”
3. Ejecutar “api-gateway”
4. Ejecutar “scraping-service”
5. Abrir CMD como administrador
6. (CMD) Dirigirnos la carpeta raíz de nuestro proyecto Angular
7. (CMD) Ejecutar el comando “ng serve”

Y listo, nuestra APP estará completamente operativa, para probarla nos dirigimos a <http://localhost:4200/>

6. CONCLUSIÓN

Desde el comienzo de mi idea y de la investigación, como alumno, he sabido que se trataba de un proyecto muy ambicioso, por ello, más que centrado en el desarrollo, este proyecto se trata de un tema de investigación centrado en la capa más alta del diseño software: la arquitectura.

Esto, sin duda me ha ayudado a crecer como desarrollador, ya que ha sido un salto inmenso respecto a las enseñanzas aportadas en el centro educativo debido a la necesidad de entender una serie de tecnologías y conceptos tan actuales, que en algunos casos sus definiciones son etéreas y poco accesibles.

Respecto a las conclusiones de la investigación, centradas en la tecnología, posiblemente estas sean las mayores ventajas por las que me atrevería a decir que este patrón de arquitectura es recomendable:

- Integrar y escalar con aplicaciones de terceros es muy sencillo
- El mantenimiento es más simple y barato — con los microservicios se puede hacer mejoras de un módulo a la vez, dejando que el resto funcione normalmente.
- Las soluciones desarrolladas con arquitectura de microservicio permiten la mejora rápida y continua de cada funcionalidad.
- Versátil — los microservicios permiten el uso de diferentes tecnologías y lenguajes

Por otro lado, creo que también existen contras, y pienso que los mayores son los siguientes:

- Debido a que los componentes están distribuidos, las pruebas globales son más complicadas.
- Los microservicios requieren desarrolladores experimentados con un nivel muy alto de experiencia.
- Se requiere un control exhaustivo de la versión.
- La arquitectura de microservicios puede ser costosa de implementar debido a los costos de licenciamiento de aplicaciones de terceros.

7. REPOSITORIO GITHUB

[Repositorio GitHub TFG](#)

