

Programación distribuida y concurrente 2021

Práctica 2

Materiales y recursos

Para el desarrollo de la actividad deberán utilizarse los siguientes materiales y recursos disponibles en la plataforma virtual:

- Clases teóricas.
- HTTP: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- HTTP status codes: https://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- Rest: https://en.wikipedia.org/wiki/Representational_state_transfer
- Rest Api - resource naming: <https://restfulapi.net/resource-naming/>
- <https://docs.oracle.com/javaee/7/tutorial/jaxrs002.htm>
- <https://javaee.github.io/tutorial/jsonp005.html>
- Jersey: <https://jersey.github.io/documentation/latest/index.html>

HTTP

- ¿Qué es el protocolo HTTP?

HTTP, de sus siglas en inglés: "Hypertext Transfer Protocol", es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML. Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (el cliente), normalmente un navegador Web. Así, una página web completa resulta de la unión de distintos sub-documentos recibidos, como, por ejemplo: un documento que especifique el estilo de maquetación de la página web (CSS), el texto, las imágenes, vídeos, scripts, etc.

Clientes y servidores se comunican intercambiando mensajes individuales (en contraposición a las comunicaciones que utilizan flujos continuos de datos). Los mensajes que envía el cliente, normalmente un navegador Web, se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.

HTTP es un protocolo basado en el principio de cliente-servidor: las peticiones son enviadas por una entidad: el agente del usuario (o un proxy a petición de uno). La mayoría de las veces el agente del usuario (cliente) es un navegador Web, pero podría ser cualquier otro programa, como por ejemplo un programa-robot, que explore la Web, para adquirir datos de su estructura y contenido para uso de un buscador de Internet.

Cada petición individual se envía a un servidor, el cual la gestiona y responde. Entre cada petición y respuesta, hay varios intermediarios, normalmente denominados proxies (en-US), los cuales realizan distintas funciones, como: gateways o caches.

- Mencionar los distintos métodos y verbos de HTTP

HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque estos también pueden ser sustantivos, estos métodos de solicitud a veces son llamados HTTP verbs. Cada uno de ellos implementa una semántica diferente, pero algunas características similares son compartidas por un grupo de ellos: ej. un request method puede ser safe, idempotent (en-US), o cacheable.

- El método GET solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
 - El método HEAD pide una respuesta idéntica a la de una petición GET, pero sin el cuerpo de la respuesta.
 - El método POST se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.
 - El modo PUT reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
 - El método DELETE borra un recurso en específico.
 - El método CONNECT establece un túnel hacia el servidor identificado por el recurso.
 - El método OPTIONS es utilizado para describir las opciones de comunicación para el recurso de destino.
 - El método TRACE realiza una prueba de bucle de retorno de mensaje a lo largo de la ruta al recurso de destino.
 - El método PATCH es utilizado para aplicar modificaciones parciales a un recurso.
- ¿Los request de tipo POST tienen un body? ¿Y los de tipo GET?

Verbos HTTP

VERBO	CUERPO REQUEST	CUERPO RESPONSE	SEGURO	IDEMPOTENTE	CACHEABLE
GET		X	X	X	X
HEAD			X	X	X
POST	X	X			X
PUT	X	X		X	
DELETE		X		X	
CONNECT	X	X			
OPTIONS	Opcional	X	X	X	
TRACE		X	X	X	
PATCH	X	X			X

- Códigos de estado de response HTTP: Describa los código: 200, 201, 400, 401, 403 y 500.

Código 200:

El código de respuesta de estado satisfactorio HTTP 200 OK indica que la solicitud ha tenido éxito. Una respuesta 200 es almacenable de forma predeterminada.

El significado de un éxito depende del método de solicitud HTTP:

- GET: El recurso ha sido recuperado y se transmite el mensaje al body.
- HEAD (en-US): Los encabezados de la entidad están en el body del mensaje.
- POST: El recurso que describe el resultado de la acción se transmite en el body del mensaje.
- TRACE: El body del mensaje contiene el mensaje de solicitud tal como lo recibió el servidor.

El resultado exitoso de un método PUT o uno DELETE (en-US) no es a menudo un 200 OK sino un 204 (en-US) No Content (o un 201 Created cuando el recurso es subido por primera vez).

Código 201:

El código de respuesta de estado de éxito creado HTTP 201 Created indica que la solicitud ha tenido éxito y ha llevado a la creación de un recurso. El nuevo recurso se crea efectivamente antes de enviar esta respuesta. y el nuevo recurso se devuelve en el cuerpo del mensaje, su ubicación es la URL de la solicitud o el contenido del encabezado de la Ubicación

El caso de uso común de este código de estado es el resultado de una solicitud metodo POST
Código 400:

La respuesta de código de estado del Protocolo de Transferencia de Hipertexto (HTTP) 400 Bad Request indica que el servidor no puede o no procesa la petición debido a algo que es percibido como un error del cliente (p. ej., sintaxis de petición malformada, solicitud inválida de enmarcado de mensajes, o enrutamiento engañoso de peticiones).

Código 401:

El código de error HTTP 401 indica que la petición (request) no ha sido ejecutada porque carece de credenciales válidas de autenticación para el recurso solicitado.

Este estatus se envía con un WWW-Authenticate encabezado que contiene información sobre cómo autorizar correctamente.

Es similar al estatus 403, pero en este caso , la autenticación si es posible.

Código 403:

El código de error de respuesta HTTP 403 Forbidden indica que el servidor ha entendido nuestra petición, pero se niega a autorizarla.

Este estado es similar a 401, pero en este caso, re-autenticarnos no provocará ninguna diferencia.

El acceso está permanentemente prohibido y vinculado a la lógica de la aplicación (como un error de contraseña incorrecta).

Código 500:

El código de respuesta 500 Error Interno del Servidor del Protocolo de Transferencia de Hipertexto (HTTP) indica que el servidor encontró una condición inesperada que le impide completar la petición.

Este código es una respuesta genérica. Usualmente, esto indica que el servidor no puede encontrar un mejor código de respuesta del tipo 5xx. En ocasiones, los administradores del servidor registran respuestas como el código de estado 500 con más detalles sobre la petición en aras de evitar que el error vuelva a ocurrir en el futuro.

- ¿Qué posibles valores de Media Type puede tomar el header Content Type de HTML?

Content-Type es la propiedad de cabecera (header) usada para indicar el media type del recurso. Dice al cliente que tipo de contenido será retornado.

Datos Discretos:

Tipo	Descripción	Ejemplo de subtipos típicos
text	Representa cualquier documento que contenga texto y es teóricamente legible por humanos	text/plain, text/html, text/css, text/javascript
image	Representa cualquier tipo de imagen. Los videos no están incluidos, aunque las imágenes animadas (como el gif animado) se describen con un tipo de imagen.	image/gif, image/png, image/jpeg, image/bmp, image/webp
audio	Representa cualquier tipo de archivos de audio	audio/midi, audio/mpeg, audio/webm, audio/ogg, audio/wav
video	Representa cualquier tipo de archivos de video	video/webm, video/ogg
application	Representa cualquier tipo de datos binarios.	application/octet-stream, application/pkcs12, application/vnd.mspowerpoint, application/xhtml+xml, application/xml, application/pdf

Datos multiparte: Es una forma de representar un documento compuesto de datos discretos.

JSON

- Defina brevemente **JavaScript Object Notation**

El objeto JSON contiene métodos para analizar JavaScript Object Notation (JSON) y convertir valores a JSON. No puede ser llamado o construido, y aparte de estas dos propiedades, no tiene funcionalidad interesante por sí mismo. JSON es una sintaxis para serializar objetos, arreglos, números, cadenas, booleanos y nulos. Está basado sobre sintaxis JavaScript pero es diferente a ella: algo JavaScript no es JSON, y algo JSON no es JavaScript.

Spring.io

- Realice un investigación y describa brevemente el framework Spring:
<https://docs.spring.io/spring/docs/current/spring-framework-reference/index.htm>

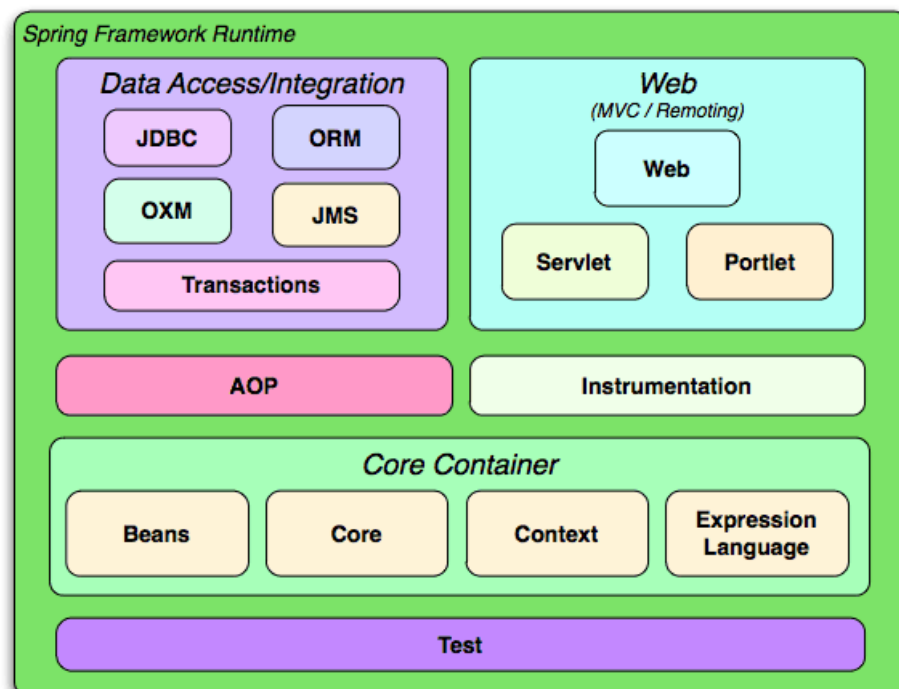
!

Spring Framework es un *framework* Open Source que facilita la creación de aplicaciones de todo tipo en Java, Kotlin y Groovy.

Si bien es cierto que, por lo que es más conocido es por la inyección de dependencias, Spring Framework está dividido en diversos módulos que podemos utilizar, ofreciéndonos muchas más funcionalidades:

- Core container: proporciona inyección de dependencias e inversión de control.
- Web: nos permite crear controladores Web, tanto de vistas MVC como aplicaciones REST.
- Acceso a datos: abstracciones sobre JDBC, ORMs como Hibernate, sistemas OXM (*Object XML Mappers*), JSM y transacciones.
- Programación orientada a Aspectos (AOP): ofrece el soporte para aspectos.
- Instrumentación: proporciona soporte para la instrumentación de clases.
- Pruebas de código: contiene un *framework* de *testing*, con soporte para JUnit y TestNG y todo lo necesario para probar los mecanismos de Spring.

Estos módulos son opcionales, por lo que podemos utilizar los que necesitemos sin tener que llenar nuestro classpath con clases que no vamos a usar.



Razones para usar Spring

Aunque no sea una característica única de Spring, el uso de inyección de dependencias facilita la programación contra interfaz, permitiendo a los distintos componentes depender únicamente de interfaces y produciendo así un código menos acoplado. No solo eso, también permite implementar el patrón *singleton* de una forma extremadamente sencilla (por defecto, las dependencias que inyectamos son singletons).

Hay cientos de tecnologías que Spring permite integrar. Desde bibliotecas que implementan opentracing hasta las que nos generan métricas para nuestra aplicación, pasando por serialización/deserialización a JSON y XML, seguridad con OAuth2 o programación reactiva entre otras. Spring aumenta la productividad y reduce la fricción al ofrecernos abstracciones sobre implementaciones de tecnologías concretas. Un ejemplo claro es el de spring-data, que nos permite definir el acceso a base de datos con interfaces Java. Esto lo consigue *parseando* el nombre de los métodos y generando la consulta con la sintaxis específica para el *driver* que utilicemos. Por ejemplo, cambiar nuestra aplicación de MySQL a PostgreSQL es tan sencillo como cambiar el *driver*. Spring se encarga de la sintaxis de forma transparente. Spring nos permite desactivar estos "comportamientos mágicos" en caso de ser necesario, por lo que podemos tomar el control cuando necesitemos más granularidad. Siguiendo con el ejemplo de spring-data, este control sería necesario si tenemos que realizar consultas mucho más complejas que un `SELECT * BY name`. En esos casos, entre otras opciones, podemos anotar nuestro método con `@Query` y escribir la consulta que deseemos.

Por lo general, Spring no obliga a implementar ni extender nada, lo que nos permite escribir código que es "agnóstico" del *framework*. De esta forma, desarrolladores con cero o muy poco conocimiento de Spring pueden realizar su trabajo sin mayores complicaciones.

Spring es de código abierto y tiene una gran comunidad detrás.

- Spring Web:

- ¿Cuál es el propósito de la anotación `@Service`?

Esta anotación se usa en una clase. `@Service` marca una clase Java que realiza algún servicio, como ejecutar lógica de negocios, realizar cálculos y llamar a API externas. Esta anotación es una forma especializada de la anotación `@Component` destinada a ser utilizada en la capa de servicio.

- Para que se usa la anotación `@Autowired`

Esta anotación se aplica a campos, métodos de "setters" y constructores. La anotación `@Autowired` inyecta la dependencia del objeto implícitamente. Cuando se usa `@Autowired` en los campos y se pasan los valores de los campos con el nombre de la propiedad, Spring asignará automáticamente los campos con los valores que se pasan.

- Spring-boot: Explique brevemente el propósito de este proyecto de Spring

Spring Boot es una infraestructura ligera que elimina la mayor parte del trabajo de configurar las aplicaciones basadas en Spring. En este tutorial, aprenderá cómo utilizar los iniciadores, los criterios y la estructura de archivos ejecutable JAR de Spring Boot para crear rápidamente aplicaciones basadas en Spring que simplemente se ejecutan.

El objetivo de Spring Boot es proporcionar un conjunto de herramientas para construir rápidamente aplicaciones de Spring que sean fáciles de configurar.

- Spring Data JPA

- ¿Para qué se usa la anotación `@Repository` y qué es la interface `JpaRepository`?

Esta anotación se utiliza en clases Java que acceden directamente a la base de datos. La anotación `@Repository` funciona como un marcador para cualquier clase que cumpla la función de repositorio u Objeto de acceso a datos.

Esta anotación tiene una función de traducción automática. Por ejemplo, cuando ocurre una excepción en el `@Repository`, hay un controlador para esa excepción y no es necesario agregar un bloque try-catch.

`JpaRepository` extiende `PagingAndSortingRepository` que a su vez extiende `CrudRepository`.

Sus principales funciones son:

`CrudRepository` principalmente proporciona funciones CRUD, `PagingAndSortingRepository` proporciona métodos para realizar registros de clasificación y clasificación, `JpaRepository` proporciona algunos métodos relacionados con JPA, como vaciar el contexto de persistencia y eliminar registros en un lote.

Debido a la herencia mencionada anteriormente, `JpaRepository` tendrá todas las funciones de `CrudRepository` y `PagingAndSortingRepository`.

JAXRS - Jersey

- ¿Con qué anotaciones se declaran los distintos métodos HTTP?

Con el `@`, ejemplos :

- `@GET`: Indica que el método anotado corresponde a una petición HTTP GET.
- `@POST`: Indica que el método anotado corresponde a una petición HTTP POST.
- `@HeaderParam`: Enlaza una cabecera http al parámetro de un método.
- `@HttpMethod`: Asocia un método con el nombre de un método HTTP.
- `@Path`: Identifica la URI de una clase o método que sirve las peticiones.
- `@ProduceMime`: Define el/los tipo(s) MIME que los métodos producen.
- `@QueryParam`: Enlaza un parámetro de la petición HTTP con un parámetro del método java anotado.
-

- Describa el propósito de las anotaciones

- `@Path`: Identifica la URI de una clase o método que sirve las peticiones.
- `@Produces`: Se utiliza para especificar el tipo MIME de las representaciones que un recurso puede proporcionar y enviar al cliente, por ejemplo "text/plain"
- `@Consumes`: Se utiliza para especificar el tipo MIME de las representaciones que un recurso puede consumir cuando ésta es enviada desde el cliente
- `@PathParam`: Es un tipo de parámetro que puede extraerse para utilizarse en la clase del recurso. Los parámetros del path de la URI se extraen de la URI de la petición, y los nombres de los parámetros se corresponden con los nombres de las variables de las plantillas del path de la URI especificados en la anotación `@Path` a nivel de clase
- `@QueryParam`: Es un tipo de parámetro que puede extraerse para utilizarse en la clase del recurso. Los parámetros de la *query* de la URI se extraen de los parámetros de *query* de la URI de la petición
- `@HeaderParam`: Enlaza una cabecera http al parámetro de un método.

- Explique el uso de la clase `javax.ws.rs.core.Response` y sus métodos estáticos `ok` y `status`. ¿Qué patrón de diseño implementa?

DTO

- Explique brevemente el patrón de diseño **Data Transfer Object**.

El patrón DTO tiene como finalidad crear un objeto plano (POJO) con una serie de atributos que puedan ser enviados o recuperados del servidor en una sola invocación, de tal forma que un DTO puede contener información de múltiples fuentes o tablas y concentrarse en una única clase simple.

- Investigue el uso de la librería Model Mapper

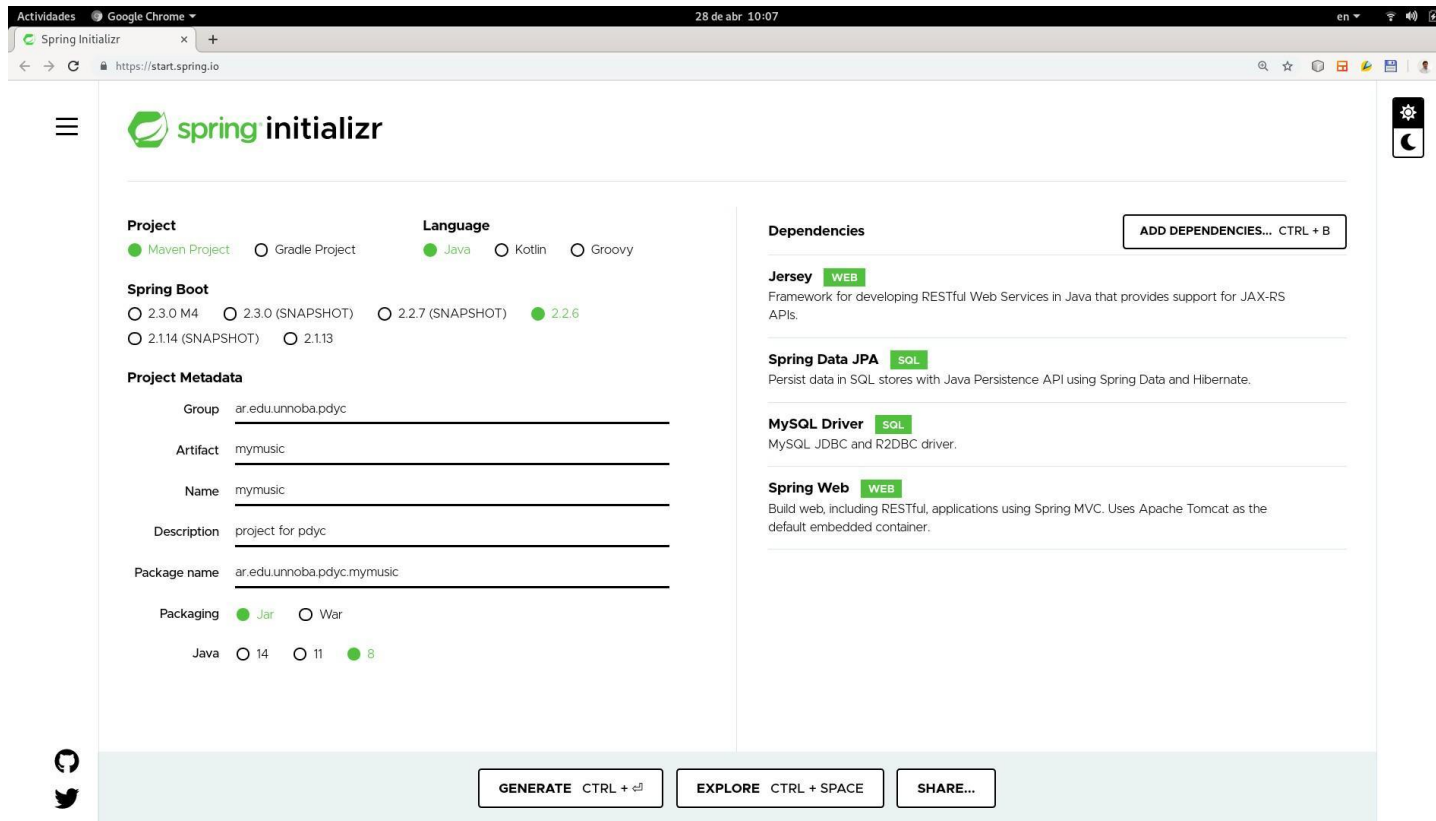
<http://modelmapper.org/getting-started/>

Las clases DTO son clases usadas como contenedores de datos sin ninguna lógica o con muy poca, se construyen con datos copiados de otras clases. Un uso de estas clases DTO es para evitar emplear el uso del patrón *Open Session in View* ya que aunque ofrece algunos beneficios también tiene algunos inconvenientes. La librería ModelMapper permite realizar los copiados de datos de un objeto origen a una nueva instancia destino de otra clase.

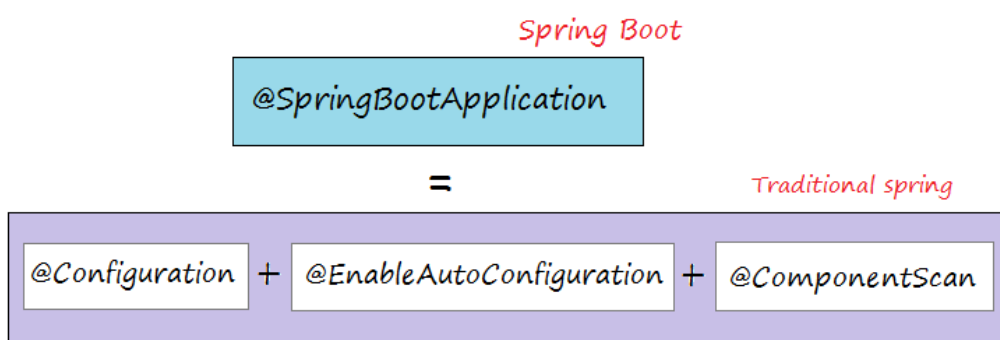
ModelMapper es una librería Java para copiar o *mapear* propiedades de un tipo de objeto a otro tipo de objeto, permitiendo copiar también los datos de las referencias a los objetos que contengan. Soporta diferentes convenciones, copiados explícitos, conversiones y proveedores para construir los objetos destino e integraciones con diferentes librerías, una de ellas jOOQ.

Inicialización de un nuevo proyecto Spring

- Antes de comenzar es necesario disponer de una instancia de un motor de base de datos (MySQL, Postgre) instalado y disponer un schema o base de datos (vacío, sin tablas).
- Completar el siguiente formulario en <https://start.spring.io/> para generar la estructura básica de un proyecto maven con las dependencias de Spring Boot, Jersey, Spring Web, Spring Data JPA y el driver JDBC para su base de datos.



- Descargar el proyecto generado y descomprimirlo.
- Posicionarse sobre el directorio del proyecto e inicializar el versionado con Git.
- Cree el archivo .gitignore para evitar versionar archivos no necesarios (archivos de configuración que use internamente el IDE) y la carpeta /target que es donde maven genera el código compilado de la aplicación.
- Crear el commit inicial que contenga todas la estructura y archivos iniciales.
- Cree y cambia a la rama develop.
- Abrir con el IDE (Netbeans) el proyecto generado.
- Identifique y analice el contenido del archivo pom.xml
- ¿Cuál es el propósito de la clase generada por el asistente que tiene la notación @SpringBootApplication?



@EnableAutoConfiguration, esta anotación le dice a Spring Boot que adivine cómo desea configurar Spring en función de la dependencia jar agregada

`@Configuration` Añote una clase con `@Configuration` para indicar que esta clase es una clase de configuración

`@ComponentScan` le dice a Spring qué clases empaquetadas anotadas serán escaneadas automáticamente por Spring y cargadas en el contenedor de beans.

- ¿Para qué se utiliza el archivo `src/main/resources/application.properties`
Las properties son un modo conveniente de proporcionar pares clave-valor que podremos utilizar desde nuestra propia aplicación. De hecho, lo que podemos definir mediante una property es una clave, a la que le asignaremos un valor, y en nuestra aplicación podemos usar esa clave. Facilita escribir directamente sobre el archivo `application.properties` las variables que se vayan a utilizar. por ejemplo el puerto, la base de datos y datos de autenticación para la base de datos.
- Ejecute el proyecto y determine cuál es la causa por la cual falla el deploy.
- Busque qué configuración debe agregar para poder resolver el inconveniente.
- Cuando logre resolver el inconveniente, realice un nuevo commit para versionar los cambios que generan el error inicial.
- Realice un nuevo commit para versionar los cambios
- Defina ahora la clase `JerseyConfig` que extienda de `ResourceConfig` para poder registrar los distintos resources para que puedan ser accedidos. Tenga en cuenta la anotación `@Component` (`org.springframework.stereotype.Component`).
- Realice un nuevo commit para versionar los cambios
- Cambie nuevamente a la rama master y realice un merge con la rama develop.

Entrega

La actividad inicia el día 20 de Abril, se publicará en ED y finaliza el 04 de Mayo, con el envío de la resolución del problema.

La entrega podrá ser grupal con un máximo de 2 personas por grupo y se deberá entregar subiendo a ED un único archivo adjunto en formato “.zip”.

Referencias:

<https://developer.mozilla.org/es/docs/Web/HTTP/Overview>
<https://developer.mozilla.org/es/docs/Web/HTTP/Methods>
<https://www.ramoncarrasco.es/es/content/es/kb/157/resumen-de-verbos-http>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/200>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/201>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/400>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/401>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/403>
<https://developer.mozilla.org/es/docs/Web/HTTP/Status/500>
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/JSON
https://developer.mozilla.org/es/docs/Web/HTTP/Basics_of_HTTP/MIME_types
<https://developer.mozilla.org/es/docs/Web/HTTP/Headers/Content-Type>
<https://www.campusmvp.es/recursos/post/que-son-spring-framework-y-spring-boot-tu-primer-programa-java-con-este-framework.aspx>
<https://mvinovaciontecnologica.wordpress.com/2020/02/06/guia-de-anotaciones-de-spring-framework/>
<https://developer.ibm.com/es/languages/java/tutorials/j-spring-boot-basics-perry/>
<https://www.it-swarm-es.com/es/java/cual-es-la-diferencia-entre-las-interfaces-crudrepository-y-jpaRepository-en-spring-data-jpa/1070175142/>
<https://www.adictosaltrabajo.com/2008/04/05/jersey-rest/>
<https://es.wikipedia.org/wiki/JAX-RS>
<http://www.jtech.ua.es/j2ee/2010-2011/restringido/servc-web/sesion04-apuntes.html>
<https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/>

investigué también para hacer andar el proyecto.