Formación Java

Contenido

Introducción

Enlace a este documento compartido: https://bosonit.sharepoint.com/:w:/s/Desarrollo-Backend/EWzuGvONpSVAkX0ON6VU09IB8RiBXfwOQ5NOSJrqW9LK2w?e=EVy3eF

Requisitos para seguir esta formación.

- Conocimientos básicos de JAVA. Sentirse cómodo con programación orientada a objetos.
- Conocimientos del protocolo HTTP Y HTML.

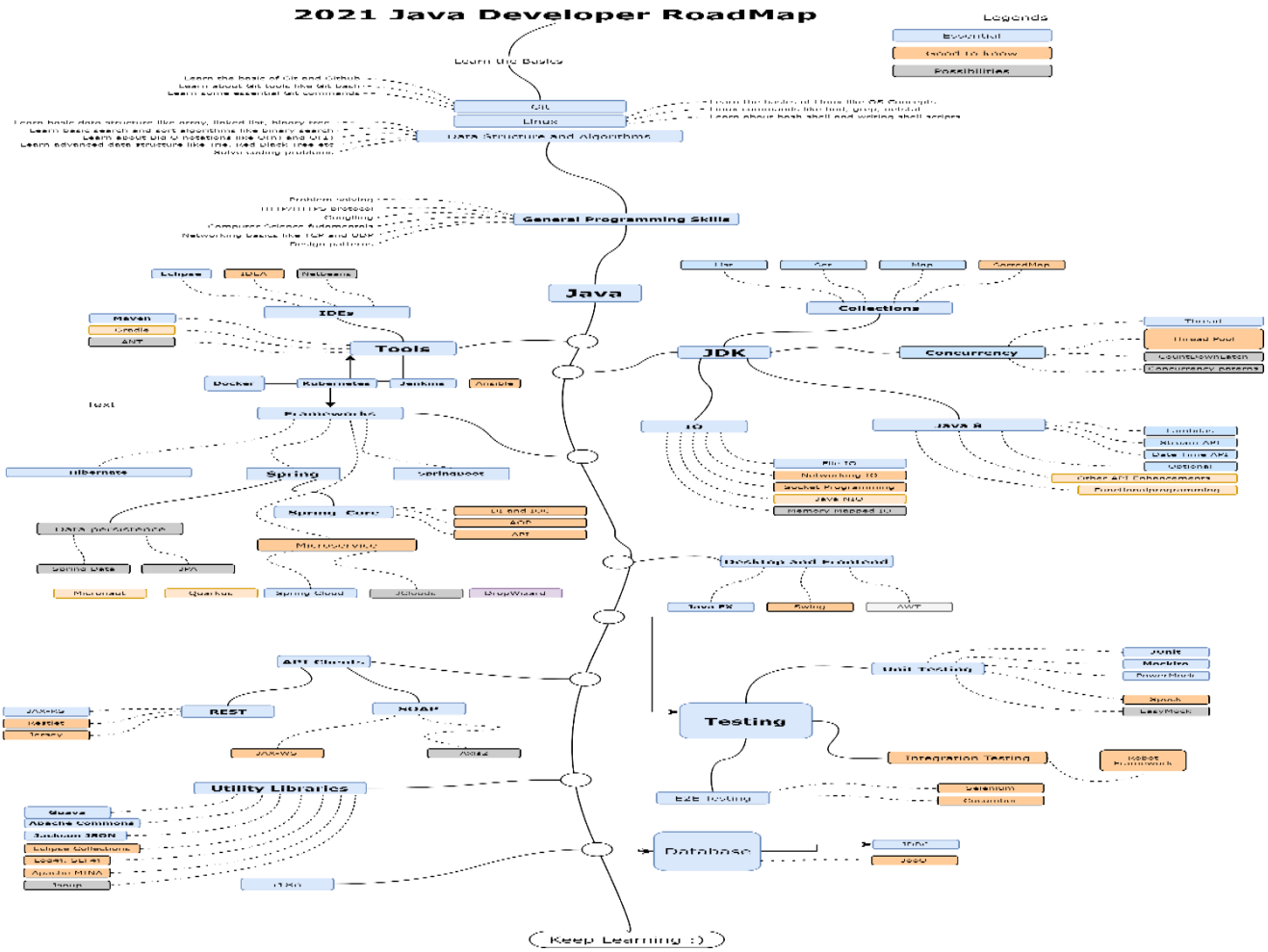
Se utilizará **IntelliJ Community** (https://www.jetbrains.com/es-es/idea/download/) como IDE.

Si no se dice lo contrario la versión de Java a usar será la 17.

Recomendable instalar el siguiente plugin de intellij (puede usarse otro) para trabajar con algún checkstyle que te ayude a homogeneizar tu estilo de programación: Google Java Style

The 2021 Java Programmer RoadMap

https://medium.com/javarevisited/the-java-programmer-roadmap-f9db163ef2c2



Básico

Libro sobre Java

https://docs.oracle.com/javase/tutorial/

Curso en Youtube: Curso Java. Presentación. Vídeo 1

Y muy importante:

SOLID: los 5 principios que te ayudarán a desarrollar software de calidad (profile.es)

Clean Code: reglas y principios - Adictos al trabajo

Java Moderno

JV1. Lambdas y Generics.

Español: https://www.ecodeup.com/entendiendo-paso-a-paso-las-expresiones-lambda-en-java/

https://docs.oracle.com/javase/tutorial/java/generics/types.html

https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html

JV2. Uso de Optional y Streams

Optional:

https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html

https://www.juanonlab.com/blog/es/uso-de-optional-en-java .

https://dzone.com/articles/using-optional-correctly-is-not-optional

Video: https://www.youtube.com/watch?v=1xCxoOuDZuU&ab_channel=Amigoscode

Streams:

https://www.oracle.com/lad/technical-resources/articles/java/processing-streams-java-se8.html

Cheatsheet sobre Streams: https://satishkathiriya99.medium.com/java-stream-api-cheatsheet-4e8c10d799dc

Ejercicio

Tiempo estimado resolución: 4 horas

Leer este fichero de texto separado por dos puntos (:). Tener en cuenta que es posible que falte el ultimo campo o que cualquier campo pueda estar vacío.

Los campos serán los siguientes:

Nombre:Población:Edad

Contenido fichero:

Jesús:Logroño:41
Andres:Madrid:19
Angel Mari:Valencia
Laura Saenz::23
Maria Calvo::38

Se deben meter todas las líneas en un objeto tipo: **List**<**Persona**>, donde **Persona** será una clase con estos campos: nombre, población y edad.

Usando **Streams**, mostrar en pantalla los datos de las personas que sean menores de 25 años. Usar **Optional (**la clase**)** para controlar si el valor de un campo está vacío (null)

Salida esperada;

Línea 1. Nombre: **Andres**. Población: **Madrid**. Edad: **19**

Línea 2: Nombre: Laura Sáenz. Población: Desconocida Edad:23

Importante: Para realizar este ejercicio **NO** se debe utilizar ningún bucle, excepto cuando estemos leyendo el archivo que se puede utilizar un **while**.

Solución al ejercicio:

Conocimientos base

Maven

Normalmente todos los proyectos se crean usando Maven. Leer estos articulos para saber que es Maven y como utilizarlo.

https://www.javiergarzas.com/2014/06/maven-en-10-min.html

https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

Ejercicio:

- Refactorizar el anterior proyecto (del punto JV2) usando Maven.
- Compilar nuestro proyecto con Maven desde IntelliJ. Esto nos deberá crear un fichero JAR, en el directorio 'target'. Conseguir ejecutar el programa java desde línea de comandos con la instrucción:
    - java –jar XXX.jar

Hint: Para poder ejecutar nuestro JAR, hay que incluir este plugin en el fichero pom.xml.

```
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>2.4</version>
        <configuration>
          <archive>
            <manifest>
              <mainClass>com.bosonit.prueba.Prueba</mainClass>
            </manifest>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Esto es porque para que Java sepa donde esta nuestro main, dentro del fichero JAR, hay que especificarlo de algún modo. Este plugin indica en que clase está nuestra función main. En el ejemplo es la clase Prueba que está en el paquete "com.bosonit.prueba".

Recordar que Java deberá estar en el path de windows.

HTTP

HTTP es el protocolo utilizado mayoritariamente para comunicación entre microservicios, aparte de ser el protocolo utilizado en la web. Es importante tener conocimientos de este protocolo pues será la base de las comunicaciones.

- Un vistazo al protocolo HTTP: https://developer.mozilla.org/es/docs/Web/HTTP/Overview
- Partes de una petición HTTP aquí: https://developer.mozilla.org/es/docs/Web/HTTP/Overview#mensajes_http

- Códigos de respuesta HTTP. Estos son los códigos que un servidor puede devolver a nuestro programa. Por ejemplo, si nos devuelve 200, significará que todo ha ido bien. Más detalles sobre los códigos aquí: https://developer.mozilla.org/es/docs/Web/HTTP/Status

POSTMAN

Postman es un programa gratuito que nos permita realizar peticiones HTTP. Hay que tener en cuenta que con un navegador al poner en la URL una dirección solo podemos hacer peticiones tipo GET y no podemos poner cabeceras.

El programa se puede bajar desde aquí: https://www.postman.com/downloads/

Un video introductorio: Como usar postman, en Español

GIT

Git para Windows: https://git-scm.com/

Fundamentos de GIT: https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git

Tutorial: https://git-scm.com/docs/gittutorial

Video: https://www.youtube.com/watch?v=HiXLkL42tMU

PDF resumen con los comandos más usados de GIT: https://education.github.com/git-cheat-sheet-education.pdf

Ejercicio:

Tiempo estimado : 2 horas máximo.

Nota: No es necesario utilizar GIT en línea de comandos. Utilizar el IDE con el más comodo nos sintamos, si es necesario.

En el directorio 'PROYECTOGIT' (estará donde queramos en nuestro disco duro) crear los siguientes ficheros: 'fichero1.txt' y 'fichero2.txt'

- Editar fichero1.txt y escribir el texto "Primera línea de fichero1"
- Crear repositorio **GIT** , realizar un **commit**.
- Editar fichero2.txt y escribir el texto "Primera línea de fichero2"
- Realizar un nuevo commit.
- Crear un repositorio en github y conseguir subir estos dos ficheros. (git push)
- Volver al punto donde el fichero1 no tiene la línea "Primera línea de fichero1"
- Volver al estado final.

- Hacer un clon desde el repositorio de github en otro directorio de nuestro ordenador. Le llamaremos 'COPIA PROYECTOGIT".
- Modificar el fichero1 añadiendo la línea "Segunda línea de fichero1"
- Realizar un **commit** y **push**.

- Volver al directorio 'PROYECTOGIT'.
- Modificar el fichero1 añadiendo la línea "Tercera línea de fichero1" (no tendremos la segunda linea)
- Hacer un commit y push. **Dara un error.**
- Solucionar conflictos y subir cambios.

- En el directorio 'COPIA PROYECTOGIT" crear la rama 'cambio1' y cambiarse a ella.
- Modificar el fichero2 añadiendo la línea "Segunda línea de fichero2"
- Realizar un commit y un push.

- Volver al directorio ''PROYECTOGIT'.
- Hacer un pull (no es necesario) Volcar (mergear) los cambios de la rama "cambio1" en la rama principal (main).
- Subir los cambios al repositorio de Github

Librerías

Lombok

Tiempo: 1 ó 2 Hora.

https://projectlombok.org/

Video explicando lo que es y cómo configurarlo en IntelliJ: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EVBQhcHf2QdOt389QfsTzCkB0Xy1AQtOu3-7ZdM1IOf8Yw?e=OLwByi

MapStruct y ModelMapper

Librerías para conversión de unos objetos en otros (entidades a DTOS, por ejemplo):

- https://mapstruct.org/. Crea el código java para hacer las conversiones en tiempo de ejecución.
- http://modelmapper.org/ . Utiliza reflexión es más lenta pero más cómoda generalmente.

Guava

https://github.com/google/guava/wiki

Eclipse Collections

https://www.eclipse.org/collections/

Spring Boot Básico

BS0. Iniciación a Spring Boot

- Pagina inicio: https://spring.io/
- Pagina para crear proyectos: https://start.spring.io
- Creación aplicación Spring Boot tipo “Hola mundo” con un controlador que atiende una petición tipo GET. https://spring.io/guides/gs/rest-service/

Videos internos:

- https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EQ_kIexRRBJLmvb_UKJzP08Brtofr0f6-Y4NIl3hL148_w?e=4cQJmp (A rehacer!!)

Para hacer este ejercicio hay que tener conocimientos del protocolo HTTP. Mozilla tiene muy buenos artículos sobre HTTP: https://developer.mozilla.org/es/docs/Web/HTTP

Ejercicio

Descargar Postman: https://www.postman.com/downloads/

Crear aplicación que tenga estos endpoints.

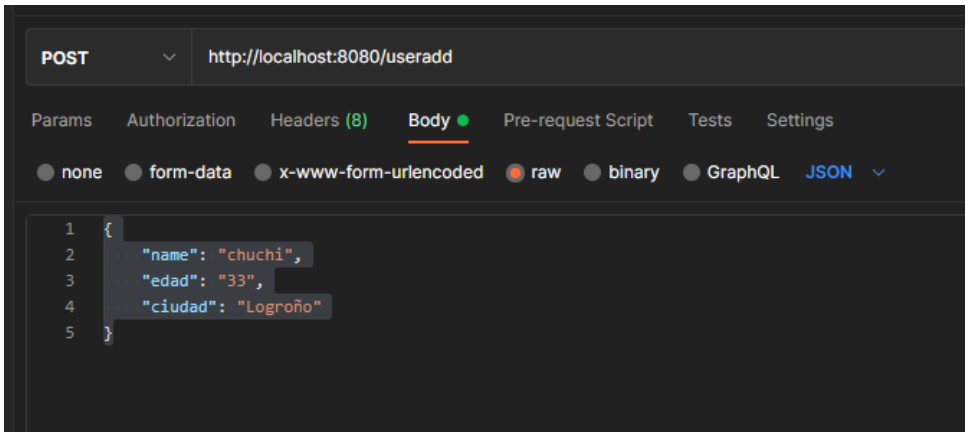
- /*user*/{nombre} (GET)

Devuelve un String poniendo “Hola” + el contenido de la variable nombre

- /useradd (POST)

EL objeto a recibir será en formato JSON.
https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON

Ejemplo de cómo mandar hacer una petición POST, mandando un objeto JSON utilizando Postman.



Recibe un objeto tipo Persona.

Persona será una clase con los campos: nombre, población y edad,

Devolver un objeto persona cuya edad sea la recibida más una.

BS1. Spring vs Spring Boot

https://javadesde0.com/diferencias-entre-spring-y-spring-boot/

https://dzone.com/articles/understanding-the-basics-of-spring-vs-spring-boot

Extra:

Patrón builder con spring:

Automatic Generation of the Builder Pattern with FreeBuilder | Baeldung

BS2. Inyección de dependencias.

Tiempo estimado: 4 horas.

- Video formación interno: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EUd5xUHsBCVIiNUnrX7N5hABHbJakmCezfv3SwCnC0idmg?e=TB57zb

Documentación:

- https://www.baeldung.com/spring-annotations-resource-inject-autowire
- https://www.vogella.com/tutorials/SpringDependencyInjection/article.html
- https://www.baeldung.com/spring-postconstruct-predestroy

Ejercicio

1) Crear aplicación con dos clases de controlador. Controlador1.java y Controlador2.java

En Controlador1, en la URL /controlador1/addPersona, tipo GET, en los headers tiene que recibir, el nombre, población y edad. (Mirar documentación sobre como pasar headers en https://www.baeldung.com/spring-rest-http-headers)

Utilizando una clase de servicio, se creará un objeto Persona. La llamada devolverá el objeto Persona creado.

En otra clase, crear el Controlador2, en la URL /controlador2/getPersona tiene que devolver el objeto Persona recibido en el Controlador1, multiplicando la edad por dos.

Importante: Utilizar Inyección de dependencias.

2) Al arrancar el programa crear una lista de objetos tipo Ciudad. Ciudad tendrá dos campos: nombre(String) y numeroHabitantes (int)

En controlador1, en la URL /controlador1/addCiudad, petición tipo POST, se añadirá una ciudad a la lista.

En controlador2, en la URL /controlador1/getCiudad, petición tipo GET, se devolverá la lista de las ciudades existentes.

Importante: Utilizar la etiqueta @Bean.

3) Crear 3 objetos Persona diferentes con funciones que tengan la etiqueta @Bean. La primera función pondrá el nombre a 'bean1', el segundo a "bean2" y el tercero a "bean3". Usar @Qualifiers

En un controlador con la URL /controlador/bean/{bean} dependiendo del parámetro mandado devolver cada uno de los Beans. Asi: /controlador/bean/bean1 devolverá el objeto cuyo nombre sea bean1 y así sucesivamente.

BS2-1. AOP - AspectJ

- Programación orientada a aspectos: https://picodotdev.github.io/blog-bitix/2020/02/programacion-orientada-a-aspectos-con-aspectj-spring-aop-y-la-clase-proxy/
- AOP en Spring Boot: https://programmerclick.com/article/57161522513/
- Intro to AspectJ: https://www.baeldung.com/aspectj

RS1. Avanzando con los controladores.

Duración: 4 Horas.

Programa ejemplo: https://spring.io/guides/gs/rest-service/

Cada alumno hará una aplicación como la del ejemplo. Incluyendo:

- Petición POST: mandando un objeto JSON en el body y recibiendo ese mismo objeto JSON en la respuesta (en el body).

- Petición GET: mandando parámetros en el path *(http://localhost:8080/user/{id})*

- Petición PUT: mandando Request Params (http://localhost:8080/post?var1=1&var2=2)

Enlaces:

https://www.baeldung.com/spring-pathvariable

https://www.baeldung.com/spring-request-param

https://www.baeldung.com/spring-rest-http-headers

Ejercicio

Realizar un CRUD sobre el objeto Persona. Para ello se creará una lista de Personas y se crearan 4 clases de controladores, una para cada una de las opciones:

- Añadir Persona. Petición POST. Body: { “name”:”Jesús” …. } http://URL/persona
- Modificar por id. Petición PUT. http://URL/persona/{id} - Body se manda los datos.

  Tener en cuenta que si no se manda un campo este está a nulo y NO queremos modificar a NULO los campos.
- Borrar (por id). Petición DELETE. http://URL/persona/{id}
- Consultar por id y por nombre. Petición GET http://URL/persona/{id} y http://URL/persona/nombre/{nombre}

El objeto Persona tendrá los campos: ‘nombre’, ‘edad’ y ‘población’.

Se deberá una clase de servicio que será la que inyectaremos en los controladores. Esa clase tendrá la lógica para el mantenimiento de los datos.

Usar **@RequestMapping(“persona”)** para que todas las llamadas sean tipo http://URL/persona/...

BS3. Ciclo de vida de Beans

- **Etiqueta @PostConstruct**

https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-lifecycle

Si precedemos una función con la etiqueta **@PostConstruct** Spring Boot ejecutara la función una vez haya instanciado el Bean. La clase donde este la etiqueta debe ser gestionado por Spring por lo cual deberá ser un **@Service**, un **@Component** o algo similar.

Por ejemplo:

```
@Component
public class MiComponente
{
	@PostConstruct
	funcion ejecutate()
	{
	    System.out.println("Línea a ejecutar cuando se haya instanciado esta clase");
	}
}
```

- **CommandLineRunner**

También se puede implementar la interface **CommandLineRunner** en una clase y crear la función “run”

https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/CommandLineRunner.html

Opcionalmente a cualquier clase con la etiqueta **@Configuration** (recordar que **@SpringBootApplication** implementa la etiqueta **@Configuration**) se le puede poner una función que devuelva un @Bean del tipo **CommandLineRunner** para que se ejecute el contenido de esa etiqueta.

Ejercicio.

Tiempo estimado 4 horas.

1) Realizar programa con tres funciones que se deberán ejecutar al arrancar el programa. Una mostrará el texto "Hola desde clase inicial", otra escribirá "Hola desde clase secundaria" y la tercera función pondrá "Soy la tercera clase". Se deberá utilizar @Postconstruct en la primera función y la interface **CommandLineRunner** en los dos siguientes. ¿En qué orden se muestran los mensajes? ¿Por qué?

Por ejemplo:

```
@SpringBootApplication
public class Main {

	@Bean
	CommandLineRunner ejecutame()
	{
	   return p ->
	   {
		System.out.println("Linea a ejecutar cuando arranque");
	   };
	}
}
```

2) Modificar la tercera función para que imprima los valores pasados como parámetro al ejecutar programa.

BS4. Configuración de aplicación: Propiedades en Spring Boot y Perfiles.

- https://docs.spring.io/spring-boot/docs/1.5.6.RELEASE/reference/html/boot-features-external-config.html
- https://www.baeldung.com/configuration-properties-in-spring-boot

Propiedades en fichero application.properties.

Relación de propiedades que se pueden poner en el fichero application.properties: https://docs.spring.io/spring-boot/docs/current/reference/html/appendix-application-properties.html

https://www.baeldung.com/spring-cloud-bootstrap-properties

Video: Spring Boot Tutorial 2020 | @ConfigurationProperties Example

- Uso de **@Value**

https://www.arquitecturajava.com/spring-boot-properties-y-value/

https://www.baeldung.com/spring-value-annotation

- Cogiendo valores de variables de entorno del sistema operativo

En el fichero **application.properties** si se pone una variable como esta:

var1={VAR_SISTEMA1:Valor_Defecto}

var1 tendrá el valor que exista en la variable VAR_SISTEMA del S.O. (en windows: set VAR_SISTEMA=valor_del_Sistema). Si no está definida esa variable en el S.O. el valor tomado será el que está a continuación de los dos puntos, en nuestro ejemplo: “Valor Defecto”.

Recordar que al ejecutar un programa java con el parámetro –D se puede especificar un variable de entorno del sistema. Por ejemplo:

```
java –DVAR_SISTEMA=mi_valor –jar app.jar
```

Si definimos una variable en un fichero “.java” de este modo:

```
@VALUE(“${VAR_SISTEMA:valor_defecto}”)
String valor;
```

La variable “valor” contendrá “valor_defecto” si no está definida en **application.properties** ni en el S.O.

En el caso de que este definida en **application.properties** tendrá el valor especificado en el fichero.

En el caso de que este definida en el S.O. tendra el valor definido en el SO. Independientemente de si está definida en el fichero **application.properties** o no.

Ejercicio:

1) Realizar aplicación que tenga en el fichero 'application.properties' los siguientes valores.

VAR1=CONTENIDO DE VALOR1

My.VAR2=100

Conseguir leer las propiedades desde una un controlador y devolverlo.

GET /valores/ -> return "valor de var1 es: "+var1+ "valor de my.var2 es: "+var2;

Intentar leer el valor 'VAR3', que no existe en application.properties. Deberá asignar a la variable el texto 'var3 no tiene valor"

GET /var3/ -> return "valor de var3 es: "+var3;

Poner la variable 'var3' dentro del S.O. Lanzar aplicación y ver como la presenta.

2) Hacer el ejercicio anterior, pero cambiando el fichero **application.properties** por "`application.yml`" (adaptando el formato) Deberían obtenerse los mismos resultados que anteriormente.

Yml es abreviatura de YAML. . https://geekflare.com/es/yaml-introduction/

Uso de fichero de propiedades externos.

Se pueden especificar otros ficheros con propiedades además de en **applicacion.properties.** En este artículo se explica cómo hacerlo.

https://www.baeldung.com/properties-with-spring

BS4-1 Perfiles.

Video explicativo: https://bosonit.sharepoint.com/:v:/s/Desarrollo-Backend/EWhUnAOo0hJCoUw-nqlBqTwBIKyoXoW664ypjgS2MQbU4g?e=ca7nnP

Spring siempre leerá el fichero **boostrap.properties** primero (si existiera), después leerá **applicacion.properties** y después todos los ficheros application-PERFIL.properties según los perfiles elegidos. En este artículo se explica cómo como usar perfiles.

http://www.profesor-p.com/2019/02/28/perfiles-en-spring-boot/

Recordar que la manera más fácil de especificar que perfil o perfiles están activos para una aplicación es establecer la variable de entorno: ***spring.profiles.active***

Ejercicio:

Tiempo realización: 4-5 horas.

Hacer programa Restful escuchando en el puerto 8081.

Poner los valores "url" y "password" como propiedades del fichero "application.properties". Devolverá el valor de estas variables, al hacer una petición GET a http://localhost:8081/parametros

Crear una clase de configuración que lea del fichero “miconfiguracion.properties” y coja los valores de las variables “valor1” y “valor2”. Imprimir todas las variables al arrancar el programa. Mostrarlas en el en el endpoint http://localhost:8081/miconfiguracion

Crear un interfaz llamado “Perfiles” con una función llamada “public void miFuncion()” que escribirá un texto. Crear la clase ‘perfil1’ y la clase ‘perfil2’ que implementen ese interfaz y anotarla con @Component. Cada una de las implementaciones tendrá la variable ‘perfil’ que tendrá el valor ‘perfil1’ y ‘perfil2’.

Crear endpoint en http://localhost:8081/perfil que devuelva el valor de la variable ‘perfil’. Dependiendo del perfil con que se lance la aplicación, Spring debe usar una clase y por lo tanto devolverá diferente texto.

BS5. Logging en Spring Boot

Video interno explicando que es el logging y para que se utiliza: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EYuAWi53zTRBnEa1FQRo0zABw77oWoa_HvdpoFTbPAVaIA?e=qDgaGh

https://www.baeldung.com/spring-boot-logging

https://howtodoinjava.com/spring-boot2/logging/logging-with-lombok/

http://www.profesor-p.com/springboot/logging-en-spring-boot/

Ejercicio:

Duración 3 horas máximo.

Crear programa que escriba los logs tipo **Error** en un fichero y los de tipo **Warning** o superiores solo a consola.

Recordar que todas las configuraciones posibles están en esta URL: https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

Parte opcional:

Los ficheros se deberán llamar ‘spring-logging-NN.log', donde NN es el numero consecutivo. Hacer que genere un fichero de log nuevo cada vez que se inicie la aplicación y que como máximo haya 5 ficheros.

Por ejemplo, el primer fichero será: spring-logging.log, el segundo spring-logging-01.log, etc.

Cada vez que se genere uno nuevo, se renombrara spring-logging.log a spring-logging-01.log, spring-logging-01.log a spring-logging-02.log y así sucesivamente. spring-logging-05.log, si existe será borrado y spring-logging-04.log será renombrado a spring-logging-05.log

Configurar el sistema de logs para que si un fichero excede los 5K de longitud también deberá rotar.

Aclaración importante: Para hacer este ejercicio completamente habría que crear un fichero ‘logback-spring.xml” o “logback.xml”.

Como crear ese fichero esta fuera del ámbito del curso solo se requiere conocer muy por encima las posibilidades que tenemos a la hora de configurar nuestro sistema de logging.

Existe documentación en la página oficial del proyecto Logback (https://logback.qos.ch/) pero NO es necesario hacerlo. Estableciendo propiedades en el fichero application.properties como “logging.file.name” o "logging.logback.rollingpolicy.max-file-size" normalmente valdra para configurar nuestro sistema de log. (ver https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#application-properties.core.logging.file.name)

Sí se requiere utilizar los diferentes niveles de log (warning, info, etc) y sacar los warning o inferiores por consola y los de error dejarlos en un fichero.

NO estar más de 3 horas con este ejercicio.

DB0 Buenas prácticas uso de DTos.

Los objetos DTOs (Data Transfer Object) los usaremos habitualmente cuando trabajemos con bases de datos.

- ¿Qué es un DTO? https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/
- Mas explicaciones sobre DTOs https://www.arquitecturajava.com/data-transfer-object-dto-un-concepto-clave/

Cómo hacer un patch con Spring:

https://www.baeldung.com/spring-rest-json-patch

DB1 Acceso a base de datos en Spring Boot Data. JPA (Hibernate).

ORM: Object Relation Mapping.

Videos internos:

- https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EaZ2Qm-hCMNKglX4p-RAl64BEPHMPHTBeV6yYt6n_zNdVA?e=11SwTb
- https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/ESWqBVmbxdxMjrp2r0gO8tABGx1Rr2pe0JGxVVhkeD0FPA?e=L8kOL3 (Aclarando conceptos)

Video sobre básico de JPA+Hibernate (Makigas):
https://www.youtube.com/watch?v=CfGDZmPj3Qw&list=PLTd5ehIj0goPcnQs34i0F-Kgp5JHX8UUv

Spring Boot JPA Tutorial (Ingles): https://www.youtube.com/watch?v=8SGI_XS5OPw

Artículos:

- http://www.profesor-p.com/2018/08/25/jpa-hibernate-en-spring/

- https://es.wikipedia.org/wiki/Java_Persistence_Query_Language

Crear querys basadas en el nombre de la función: https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

JPQL. Uso de anotación @Query: https://www.baeldung.com/spring-data-jpa-query

QBE (Opcional): http://www.profesor-p.com/spring/data/qbe/

Configuración para diferentes bases de datos:

- **H2**. https://www.baeldung.com/spring-boot-h2-database

Ejemplo de configuración en fichero "**application.properties**" para conexión a base datos H2 cuyos datos se guardarán en fichero.

```
# Poner endpoint para consola H2. Desde ella podremos manejar las tablas, realizar select,insert, etc.
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Parametros conexión a base de datos

spring.datasource.url=jdbc:h2:file:~/test09
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
hibernate.dialect=org.hibernate.dialect.HSQLDialect
spring.jpa.hibernate.ddl-auto=create-drop

# Mostrar las sentencias SQL ejecutadas por JPA.
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

- **PostgreSQL**: https://mkyong.com/spring-boot/spring-boot-spring-data-jpa-postgresql/
- **MySQL**: http://cristianruizblog.com/spring-boot-mysql/

EJ2 CRUD con validación

Tiempo estimado: 8 horas.

Ver este video interno antes de hacer este ejercicio:

https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EaJHzzo-YWRMsI__9GgtaKoBz-xhw3sKqU6I4IOaYda4kg?e=hIYoeL

Crear CRUD de la siguiente tabla:

```
table persona
{
  id_persona int [pk, increment]
  usuario string [not null max-length: 10 min-length: 6]
```

```
 password string  [not null]
 name string [not null]
 surname string
 company_email string  [not null ]
 personal_email string [not null]
 city string [not null]
 active boolean  [not null]
 created_date date  [not null]
 imagen_url string
 termination_date date
}
```

Realizar validaciones necesarias con lógica en java (no usar etiqueta @Valid)

if (usuario==null) {throw new Exception("Usuario no puede ser nulo"); }

if (usuario.length()>10) { throw new Exception("Longitud de usuario no puede ser superior a 10 caracteres)

…

EI ID autoincremental se puede hacer con estas simples instrucciones:

```
@GeneratedValue
private int id;
```

Poner 3 endpoints en la búsqueda.

- Buscar por ID
- Buscar por nombre de usuario (campo usuario)
- Mostrar todos los registros.

Usar DTOS, interfaces y clases de servicio.

BP-1 Buenas prácticas. Arquitectura hexagonal

Video presentación (clase interna): https://bosonit.sharepoint.com/:v:/s/Desarrollo-Backend/EVFPEzAYDYNFoD1VsvAcEGEB_-x5AE6P9nJ1VD9K0OKpbA?e=fbOGqE

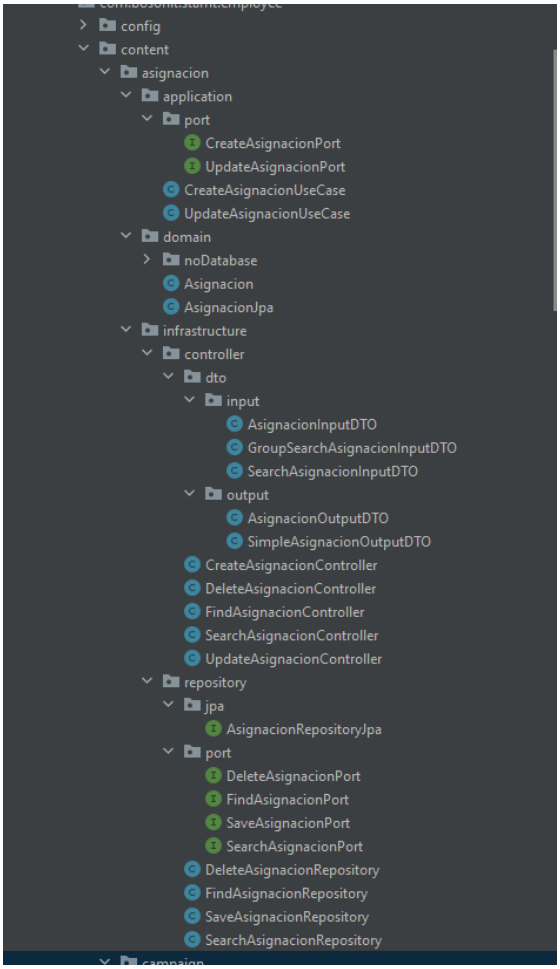
DDD -> Una introducción a Domain Driven Design | The talking bit (franiglesias.github.io)

Arquitectura Hexagonal. Introducción Arquitectura Hexagonal - DDD

Video explicando casos reales (case interna) : https://bosonit.sharepoint.com/:v:/s/Desarrollo-Backend/ERZ2EV-EoWJMhk_0DKZGJlsBKRKbNdLap_r9hWDtiPNAIA?e=snlzgm

Articulo: https://reflectoring.io/spring-hexagonal/

Esta es una pantalla de cómo están estructurados los ficheros en Staffit para un CRUD de la tabla (entity) 'asiginacion'



- En infrastructure/controller estarán los controladores (puntos de entrada de las APIS).
- Estos harán referencia a los interfaces del directorio ‘application/port’. En la imagen serían los ficheros CreateAsignacionPort y UpdateAsignacionPort. En algunos casos llamaran a los interfaces de ‘repository/port’ , en la imagen serían DeleteAsignacionPort, FindAsingacionPort, etc.
- AsignacionRepositoriyJPA es una interface que extiende de JPARepository.
- Los ficheros terminados en ‘useCase’ en el directorio ‘application’ son las implementaciones de los interfaces situados en la el directorio ‘application/port’. Estas clases tendrán referencia a los interfaces de ‘repository/port’.
- Los ficheros terminados en Repository serán los que accederán a la base de datos, es decir tendrán referencia a la entity, en este caso a AsignacionJPA a traves de la interface ‘AsignacionRepositoryJPA’.
- Los DTOs estarán en ‘infrastructure/controller/dto’, estando separados en los directorios input y ouptut según sean de entrada o salida respectivamente.

Los controladores en un CRUD, por norma seguirán estar reglas:

- Añadir. Petición POST Url: /asignacion. Recibirá el objeto AsignacionInputDTO, devolverá el objeto AsignacionOutputDTO.
- Editar. Petición PUT: URL /asignacion/{id} Recibirá el objeto AsignacionInputDTO, devolverá el objeto AsignacionOutputDTO.
- Borrar. Peticion DELETE. URL: /asignacion/{id}. No recibe ni devuelve nada.

- Buscar por ID: GET URL: /asignacion/{id} devolverá un AsignacionOutputDTO
- Buscar todos los elementos. GET URL: /asignacion. Devolvera un

  LIST< AsignacionOutputDTO>

- Buscar por nombre. GET URL: /asignacion/nombre/{nombre}. Devolvera un

  LIST< AsignacionOutputDTO>

A tener en cuenta:

- Los controladores deben recibir solo DTOS tipo INPUT y devolver DTOS tipo OUTPUT. No debe haber ninguna referencia a las entidades o repositorios en los controladores.
- Dentro de lo posible se crearán constructores y/o funciones en los DTOS y/o entities para realizar las transformaciones necesarias entre un objeto y otro.
- Se debe comprobar que un ID existe al intentar borrarlo o editarlo. En el caso de que no exista devolver un 406. Not Acceptable (en-US)
- Si al buscar por ID este no existe se deberá devolver un 404.
- Al buscar todos los registros o por unos criterios dados (nombre, por ejemplo) no se encuentra nada, devolver una lista vacía.

Ejercicio.

Refactorizar el anterior CRUD, utilizando buenas prácticas.

No es necesario crear la carpeta repository. Por hacer más simple el ejercicio se pueden poner todos los servicios en application.

Recordar que todas las dependencias (autowired) deben ser hacía interfaces y nunca hacía clases.

BS6 SPEL. Validación datos de entrada.

Uso de etiqueta @Valid https://www.baeldung.com/spring-boot-bean-validation

Validaciones personalizadas: https://www.baeldung.com/spring-mvc-custom-validator

Uso de interfaces para validar. https://levelup.gitconnected.com/constraint-validation-in-spring-boot-microservices-b89805e9c540

BS7 Gestionando devolución códigos HTTP

En SpringBoot para gestionar lo devuelto por un controlador se utilizará **ResponseEntity**

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html

Documento iniciación al protocolo HTTP **https://javadesde0.com/responseentity-y-http-response/**

Códigos HTTP: https://developer.mozilla.org/es/docs/Web/HTTP/Status

Ejemplos de Uso:

- https://www.baeldung.com/spring-response-entity
- https://technicalsand.com/using-responseentity-in-spring/#5-how-to-use-responseentity

BS8 Personalizando respuestas.

http://www.profesor-p.com/2018/11/20/personalizar-codigo-http-en-spring-boot/

Para juntar la validación (la etiqueta @Valid) y la personalización de las respuestas mirar este articulo:

https://medium.com/@salithachathuranga94/validation-and-exception-handling-in-spring-boot-51597b580ffd

Resumiéndolo lo que hay que hacer es poner el @Valid correspondiente al recibir nuestro Dto

```
@PostMapping(path = "/users")
    public ResponseEntity<User> saveUser(@RequestBody @Valid
UserRequestDTO userRequest) {
        return new ResponseEntity<>(userService.saveUser(userRequest),
HttpStatus.CREATED);
    }
```

En nuestro DTO pondremos las validaciones que deseemos:

```
@Data
@Builder
public class UserRequestDTO {

    @NotBlank(message = "Invalid Name: Empty name")
    @NotNull(message = "Invalid Name: Name is NULL")
    @Size(min = 3, max = 30, message = "Invalid Name: Must be of 3 -
30 characters")
    String name;

    @Email(message = "Invalid email")
    String email;

    @NotBlank(message = "Invalid Phone number: Empty number")
    @NotNull(message = "Invalid Phone number: Number is NULL")
    @Pattern(regexp = "^\\d{10}$", message = "Invalid phone number")
    String mobile;

    @Min(value = 1, message = "Invalid Age: Equals to zero or Less
than zero")
    @Max(value = 100, message = "Invalid Age: Exceeds 100 years")
    Integer age;

}
```

Por fin, tendremos que poner en la clase que controla los errores un código parecido a este:

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, List<String>>>
handleValidationErrors(MethodArgumentNotValidException ex) {
        List<String> errors = ex.getBindingResult().getFieldErrors()

.stream().map(FieldError::getDefaultMessage).collect(Collectors.toList
());
        return new ResponseEntity<>(getErrorsMap(errors), new
HttpHeaders(), HttpStatus.BAD_REQUEST);
    }

    private Map<String, List<String>> getErrorsMap(List<String>
errors) {
        Map<String, List<String>> errorResponse = new HashMap<>();
        errorResponse.put("errors", errors);
        return errorResponse;
    }

}
```

Ejercicio:

Tiempo estimado: 4 Horas.

Crear dos tipos de excepción al CRUD anteriormente realizado:

- **NotFoundException** que generara un código HTTP 404. Se lanzará cuando no se encuentren registros en un findById o si al borrar o modificar un registro este no existe.
- **UnprocesableException** que devolverá un 422 (**UNPROCESSABLE** ENTITY) cuando la validación de los campos no cumpla los requisitos establecidos, al modificar o insertar un registro.

Ambas excepciones deberán devolver un objeto llamado **CustomError** con los campos:

Date timestamp;

Int HttpCode;

String mensaje; // Devolverá el mensaje de error.

DB1-1 Generación autoincreméntales

Técnicas para generación de campos autoincreméntales en JPA:

- https://www.adictosaltrabajo.com/2019/12/26/hibernate-uso-de-generationtype-y-otras-anotaciones/

Video interno sobre como generar campos autoincrementales tipo String: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EYeQlAjk1CZAjLNHZgMOtrIBMdkRcqPD5f8ofsrrHFazNQ?e=wqu2dq

Clase de la que se habla en el video que sirve para generar autoincreméntales de tipo String:

```
public class StringPrefixedSequenceIdGenerator extends SequenceStyleGenerator {

  public static final String VALUE_PREFIX_PARAMETER = "valuePrefix";

  public static final String VALUE_PREFIX_DEFAULT = "";

  private String valuePrefix;

  public static final String NUMBER_FORMAT_PARAMETER = "numberFormat";

  public static final String NUMBER_FORMAT_DEFAULT = "%d";

  private String numberFormat;

  @Override

  public Serializable generate(SharedSessionContractImplementor session, Object object)
throws HibernateException {

    return valuePrefix + String.format(numberFormat, super.generate(session, object));

  }

  @Override

  public void configure(Type type, Properties params, ServiceRegistry serviceRegistry) {

    super.configure(LongType.INSTANCE, params, serviceRegistry);

    valuePrefix =    ConfigurationHelper.getString(VALUE_PREFIX_PARAMETER, params,
VALUE_PREFIX_DEFAULT);

    numberFormat =      ConfigurationHelper.getString(NUMBER_FORMAT_PARAMETER, params,
NUMBER_FORMAT_DEFAULT);

  }

}
```

Ejemplo de definición de campo autoincremental en una entidad:

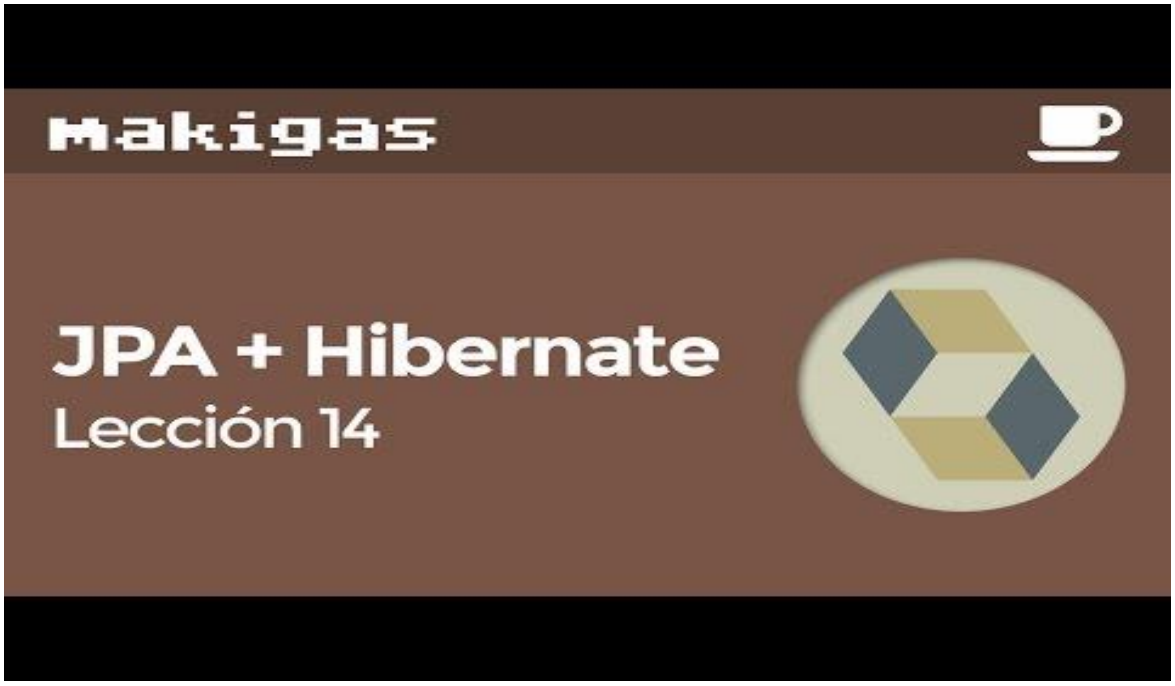
```
@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "ausencias_seq")

@GenericGenerator(

    name = "ausencias_seq",

    strategy = "com.bosonit.staffit.shared.sequences.StringPrefixedSequenceIdGenerator",

    parameters = {

      @Parameter(name = StringPrefixedSequenceIdGenerator.INCREMENT_PARAM, value = "1"),

      @Parameter(name = StringPrefixedSequenceIdGenerator.VALUE_PREFIX_PARAMETER, value =
"AUS"),

      @Parameter(name = StringPrefixedSequenceIdGenerator.NUMBER_FORMAT_PARAMETER, value =
"%08d")

    })

@Column(name = "id")

private String id;
```

Más documentación sobre campos autoincreméntales tipo **String :**
https://www.baeldung.com/hibernate-identifiers#3-sequence-generation

DB2 Relación entre entidades en JPA

Video de formación:

- Java: JPA + Hibernate – 14. OneToMany: Anotaciones



Resumen de etiquetas en Hibernate: https://thorben-janssen.com/key-jpa-hibernate-annotations/#Enumerated

Relaciones entre tablas: Many-To-One , One-To-Many ,One -to-one:

- **¡Error! Referencia de hipervínculo no válida.**
- https://thorben-janssen.com/best-practices-many-one-one-many-associations-mappings/
- https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/
- https://thorben-janssen.com/hibernate-tips-same-primary-key-one-to-one-association/

EJ3-1 CRUD con relaciones entre tablas

Tiempo: 12 horas.

Al ejercicio anterior añadir las siguientes tablas:

table student

{

```
  id_student string [pk, increment]
  id_persona string [ref:-  persona.id_persona] -- Relación OneToOne con la tabla persona.
  num_hours_week int   [not null] -- Número de horas semanales del estudiante en formación
  coments string
  id_profesor string [ref: > profesor.id_profesor] -- Un estudiante solo puede tener un profesor.
  branch string [not null] -- Rama principal delestudiante (Front, Back, FullStack)
}
table profesor
{
  id_profesor string [pk, increment]
  id_persona string [ref:- persona.id_persona] -- Relación OneToOne con la tabla persona.
  coments string
  branch string [not null] -- Materia principal que imparte. Por ejemplo: Front
}
table estudiante_asignatura
{
  id_asignatura String [pk, increment]
  id_student string [ref: > student.id_student] -- Un estudiante puede tener N asignaturas
  asignatura string  -- Ejemplo: HTML, Angular, SpringBoot...
 coments string
 initial_date date [not null], -- Fecha en que estudiante empezó a estudiar la asignatura
 finish_date date  -- Fecha en que estudiante termina de estudiar la asignatura
}
```

--- Ejemplo de entidades --

```
@Entity
@Table(name = "estudiantes")
@Getter
@Setter
public class Student {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   Integer id_student;
   @OneToOne
```

```
    @JoinColumn(name = "id_persona")
    Persona persona;
    @Column(name = "horas_por_semana")
    Integer num_hours_week;
    @Column(name = "comentarios")
    String coments;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "id_profesor")
    Profesor profesor;
    @Column(name = "rama")
    String branch;
    @OneToMany
    List<Alumnos_Estudios> estudios;
}


 @Entity
@Table(name = "estudios")
@Getter
@Setter

public class Alumnos_Estudios {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    Integer id_study;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "profesor_id")
    Profesor profesor;
    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "id_student")
    Student student;
    @Column(name = "asignatura")
    String asignatura;
    @Column(name = "comentarios")
    String comment;
    @Column(name = "initial_date")
    Date initial_date;
    @Column(name = "finish_date")
    Date finish_date;
```

La relación gráfica entre las tablas la tenéis en este enlace:
https://dbdiagram.io/d/60938b29b29a09603d139d64

La relación de la tabla student y profesor con Persona es one-to-one

La relación entre estudiante y profesor es tipo one-to-many. Un estudiante puede tener un solo profesor y un profesor puede tener N estudiantes.

Un estudiante puede estar en N estudios por lo cual la relación es many-to-many.

Realizar CRUD sobre TODAS las tablas. Cuando se busque por ID la persona, si es estudiante, debe devolver los datos de 'estudiante', 'profesores' y los estudios realizados por el estudiante. Si la persona es profesor, sacara los datos de la entidad profesor relacionada, los estudiantes a su cargo, y para cada estudiante los estudios realizados

Como se puede apreciar los índices son tipo String y autoincreméntales. Recordar que en el punto DB1-1 tenéis como crear campos autoincreméntales del tipo String.

- **Primera parte ejercicio**: Realizar CRUD de estudiante con endpoints similares a los de persona. De momento, no poner relación con profesor (id_profesor)

  Modificar endpoint "/estudiante/{id}" para que acepte el QueryParam 'ouputType'. Por defecto deberá tener el valor 'simple' (es decir si no se manda cogerá como valor el literal 'simple').

  En el caso de que el valor de 'outputType' sea 'simple', la consulta devolverá solo los datos del estudiante. En el caso de que sea 'full' devolverá los datos del estudiante y de la persona asociada.

  Ejemplo:

  GET "http://localhost:8080/{id}?outputType=full"

  {

  id_student : 1,

  num_hours_week : 40,

  coments: "No puede trabajar por las tardes",

  id_persona: 1,

  user: 'francisco',

  password: "secreto",

  Name: "Francisco",

  Surname: "perez",

  company_email: "francisco@bosonit.com",

  personal_email: "francisco@gmail.com",

  city : "zaragoza",

  Active: true

  created_date: '2021-10-03'

  imagen_url: "http://pictures.com/imagen1.png",

  termination_date: null

  }

  Como se puede observar habrá que crear diferentes DTOs de salida.

- **Segunda parte ejercicio**: Realizar CRUD de las tablas restantes.

  Tener en cuenta que UNA persona solo puede ser o profesor o estudiante.

  En TODOS los endpoints de búsqueda de la entidad persona (por ID, por nombre o todas las personas), aceptar un parámetro que indique si debe devolver solo los datos de la persona o también los de estudiante o profesor si fuera alguno de ellos.

- **Tercera parte:** Crear endpoint en 'estudiante_asignatura' que permita buscar por id de estudiante. Este endpoint sacara todas las asignaturas que tiene un estudiante.

  Realizar chequeos lógicos: ¿Borrar persona si tiene un estudiante/profesor asignado? ¿Borrar asignatura si tiene estudiantes asignados?

- **Cuarta parte**: Realizar endpoints en estudiante para asignarle una o más asignaturas. Crear otro endpoint para desasignar una o más asignaturas (deberá poder recibir una lista de IDs de asignaturas).

BS9. Clase RestTemplate. Usos con Feign.

Video interno: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EeT1XZCUccROjTsgvzxUhTUBd1lpuNa1JwKbEXtKMV1i6g?e=NMVZfY

En la URL https://github.com/chuchip/feignExample tenéis el programa del que se habla en el video anterior.

Artículos: https://www.adictosaltrabajo.com/2017/09/26/spring-cloud-feign-declarative-rest-client/

RestTemplate: http://www.profesor-p.com/2-spring/8-restful/

Feign: http://www.profesor-p.com/2019/01/03/microservicios-distribuidos-con-eureka/

Video sobre uso de Feign en Staffit: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EX3UykYv1mJFuZDo-ANQGaIBpxVZRcAGv_HJrwoXSPtntA?e=zvewAf (opcional)

Ejercicio:

En el CRUD anteriormente realizado incluir en la entidad Persona el siguiente endpoint.

@RestControler("/profesor/{id})
ProfesorOutputDto GET getProfesor(@PathVariable int id)

Este endpoint deberá llamar al de la entidad profesor que devuelve el profesor por id, en el puerto 8081 usando RestTemplate.

Usando Feign:

Crear la llamada anterior usando feign.

Para hacer la prueba deberemos tener dos instancias de nuestro programa lanzado. Una corriendo en el puerto 8080 y otra en el puerto 8081. La prueba consistirá en llamar a

GET localhost:8080/persona/profesor/1 y que esta llamada llame a localhost:8081/profesor/1 devolviéndonos los datos del profesor.

BS10 Swagger y HalExplorer

Duración: 1 Hora.

https://www.baeldung.com/swagger-2-documentation-for-spring-rest-api

https://www.baeldung.com/spring-rest-hal

BS11 Spring DevTools

Utilizando dependencia devtools en SpringBoot: https://www.arquitecturajava.com/spring-boot-devtools-y-recarga-de-aplicaciones/

BS12 CORS

¿Qué es CORS? https://developer.mozilla.org/es/docs/Web/HTTP/CORS

Implementar en Spring seguridad CORS.

https://spring.io/guides/gs/rest-service-cors/

https://www.arquitecturajava.com/spring-rest-cors-y-su-configuracion/

Ejercicio:

Tiempo estimado: 2 horas.

Permitir realizar peticiones entre dominios en el ejercicio EJ3-1 para la ruta '/person'.

Para probar ir a la página: https://codepen.io/de4imo/pen/VwMRENP .

Actualizar el back para que funcionen estas llamadas:

Alta: Tipo. POST - Ruta: http://localhost:8080/addperson

Objeto mandado:

'usuario': document.getElementById('username').value,

'password': document.getElementById('passwd').value,

'name': document.getElementById('name').value,

'surname': document.getElementById('surname').value,

'company_email': document.getElementById('emailcomp').value,

'personal_email': document.getElementById('emailpers').value,

'city': document.getElementById('city').value,

'active': document.getElementById('active').checked,

'created_date': document.getElementById('created_date').value,

```
    'imagen_url': document.getElementById('imagen_url').value,

    'termination_date': document.getElementById('finish_date').value,

  })
```

Consulta: http://localhost:8080/getall

List<Person> (Mismos campos que en el add)

Docker

Clase de Docker:

https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EYsDkLNWJXFKk3VzlK1LiGgBDiNxgCIsqckO9ZNgSLKkiw?e=Z8MnSW

https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EUwjpnppiUhJuFMfrJZNcCIBSrkqMyFE9DcTw41yHohyKQ?e=wKwabv

Ejercicio:

Crear imagen Docker que incluya una aplicación Spring Boot que se conecte a una base de datos PostgreSQL.

Levantar Docker con servidor de PostgreSQL que no sea accesible desde nuestro Windows, es decir, no mapear el puerto de postgres a nuestro host ("-p 5432:5432"). El usuario para conectarse será 'postgres' y el password: 'contrasena'. Se creará la base de datos 'test'

Deberemos meter en un Docker la aplicación de Spring Boot y el servidor de PostgreSQL deberá estar en otro Docker, dentro de la misma red virtual para que ambos contenedores Docker se vean entre ellos.

Desde Windows deberemos podernos hacer peticiones con postman a la aplicación en Spring Boot.

Pistas:

- Poner el driver de PostgreSQL y configurar fichero application.properties para conectarte a Postgresql. Mirar este articulo como guía: https://dzone.com/articles/bounty-spring-boot-and-postgresql-database
- Crear una red interna: docker network create mynetwork
- Ejecutar postgres especificando que este en una red definida por nosotros:

> docker run --network mynetwork --name postgres_test -ePOSTGRES_USER=postgres -e POSTGRES_PASSWORD=contrasena -e POSTGRES_DB=test postgres

Fichero DockerFile de ejemplo para la aplicación de Spring Boot.

```
FROM openjdk:11
EXPOSE 8081
COPY /target/*.jar /usr/local/lib/spring.jar
ENTRYPOINT ["java","-jar","/usr/local/lib/spring.jar"]
```

Jenkins

Introducción: Ver los primeros 5 minutos.

Jenkins Full Course | Jenkins Tutorial For Beginners | Jenkins Tutorial | Simplilearn

Nivel Avanzado

JVA2. Java 17

Uso de records.

- https://docs.oracle.com/en/java/javase/16/language/records.html
- https://www.arquitecturajava.com/java-record-class-y-jdk-14/

Ejercicio:

Coger el ejercicio EJ2 y refactorizarlo usando records.

DBA1. Querys avanzadas. CriteriaBuilder.

Duración: 8 Horas

https://thorben-janssen.com/hibernate-tips-select-pojo-criteria-query/

http://www.profesor-p.com/2019/02/14/querys-avanzadas-con-jpa-en-spring-boot/

https://en.wikibooks.org/wiki/Java_Persistence/Criteria

Joins

https://thorben-janssen.com/hibernate-tip-left-join-fetch-join-criteriaquery/

SubQuerys

https://thorben-janssen.com/hibernate-tip-subquery-criteriaquery/

Delete and Update

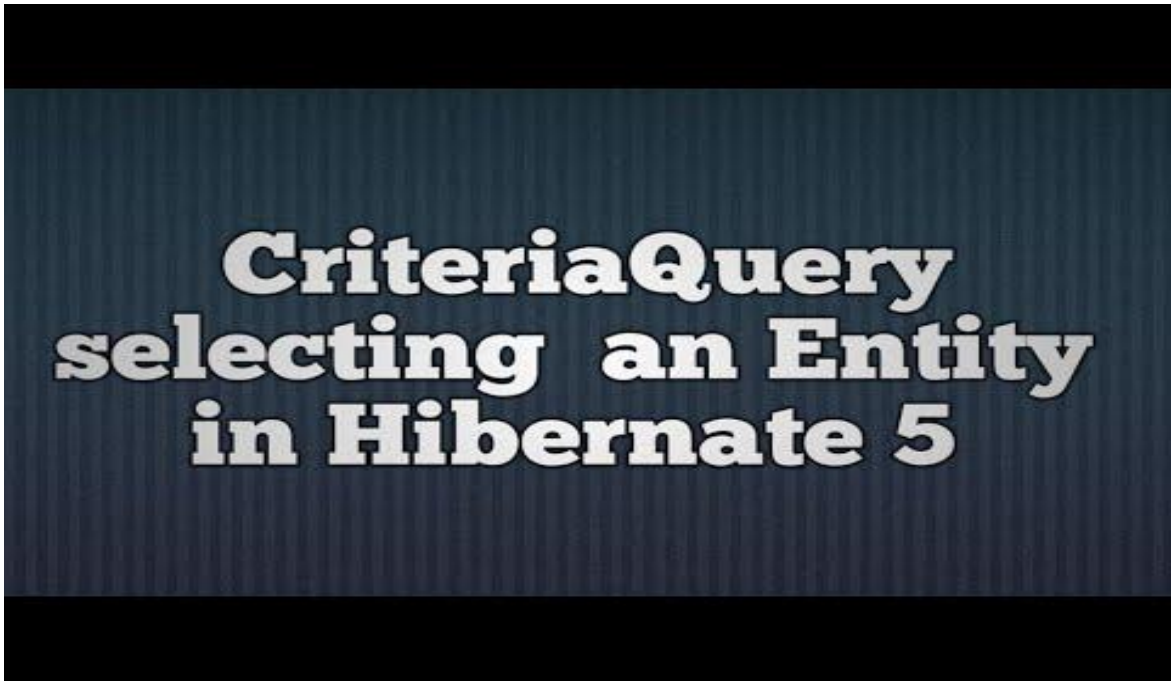
https://thorben-janssen.com/criteria-updatedelete-easy-way-to/

Uniones entre dos tablas sin relación:

https://stackoverflow.com/questions/30639132/joining-tables-without-relation-using-jpa-criteria

Videos (en ingles):

Hibernate CriteriaQuery APIs Tutorials



Ejercicio

1) Sobre el ejercicio anterior de mantenimiento de Personas. Crear endpoint que permita buscar aquellas personas cuyo 'user', 'name', 'surname' o 'fecha creación' superior a una dada y/o inferior a una dada. Incluir otro campo que indicará si se quiere ordenar el resultado por 'user' o 'name'. Tener en cuenta que cualquiera de esos campos puede ser mandados, pero ninguno es obligatorio.

2) Incluir paginación. Es decir que se pueda mostrar N elementos a partir de la página dada. Así si se define que el tamaño de la página es 10 y que se quiere mostrar la página 1, se mostrara del registro 11 al 20 (ambos inclusive). Por defecto el tamaño de la página será 10 y el número de página será obligatorio mandarlo.

DBA2. Acceso a Mongo con MongoRepository y MongoTemplate.

Duración: 10 Hora

Si no conocen Mongo hacer este curso que llevará 10 horas:
https://university.mongodb.com/courses/M001/about

MongoTemplate (equivalente a JDBC Template)

https://www.concretepage.com/spring-5/spring-data-mongotemplate#MongoTemplate

https://www.appsdeveloperblog.com/spring-boot-and-mongotemplate-tutorial-with-mongodb/

MongoRepository (equivalente a JPA)

https://blog.marcnuri.com/spring-data-mongodb-implementacion-de-un-repositorio-a-medida

Ejercicio

Realizar CRUD de personas, utilizando MongoTemplate.

SA1 Websockets

Documentación:

- ¿Que son los websockets ? https://refactorizando.com/websockets-spring-boot/
- Ejercicio práctico: https://spring.io/guides/gs/messaging-stomp-websocket/
- Más documentación:
    - https://www.baeldung.com/websockets-spring
    - https://www.baeldung.com/spUring-websockets-sendtouser

SA2 Subiendo y bajando ficheros.

Ejemplo oficial: https://spring.io/guides/gs/uploading-files/

Diferentes maneras de recibir un fichero: https://www.baeldung.com/sprin-boot-multipart-requests

Ejercicio.

1)

- Permitir subir un fichero incluyendo como metadato la categoría. Guardar el fichero y en una tabla el metadato, nombre de fichero, fecha de subida, etc. Devolver Entidad 'Fichero' con los datos, incluyendo un ID único.
- Descargar fichero, buscándolo por diferentes métodos (id y nombre de fichero).

2) Crear programa con estos endpoints.

- Petición POST. /upload/{tipo} (@RequestParam("file") MultipartFile file,

RedirectAttributes redirectAttributes).

Solo aceptara ficheros con la extensión indicada en la URL

- Petición GET /setpath?path={directorio_a_guardar}

El programa al arrancar permite mandar un parámetro que es el directorio donde debe guardar los ficheros. Ejemplo: java –jar MIPROGRAMA.jar "/DIRECTORIO_A_GUARDAR". Si no se especifica esta variable ponerlo en el directorio donde se lanza java.

SA6. Uso de caches en Spring Boot. Diferentes tipos de cache.

Duración: 1 Hora

http://www.profesor-p.com/2019/05/12/cacheando-datos-en-spring-boot/

https://www.arquitecturajava.com/introduccion-a-spring-cache/

SA7. Spring Actuator

https://danielme.com/2019/04/09/spring-boot-actuator/

https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html

https://www.baeldung.com/spring-boot-actuators

Testing.

TE1 Testing en SpringBoot. Mockito y JUNIT.

Duración: 12 Horas

https://www.youtube.com/watch?v=EOkoVm3rtNQ&list=PLTd5ehIj0goML37B7s9I9iN2zhJCfxJBC&t=4s

https://www.baeldung.com/injecting-mocks-in-spring

Usando mockito:

https://www.springboottutorial.com/spring-boot-unit-testing-and-mocking-with-mockito-and-junit

https://mkyong.com/spring-boot/spring-rest-integration-test-example/

Uso de TestRestTemplate:

https://howtodoinjava.com/spring-boot2/testing/testresttemplate-post-example/

Diferentes métodos para hacer tests

http://www.profesor-p.com/spring/testeando-tu-aplicacion-web/

Video explicando la aplicación: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/Ef1jvelR8mdCiYbNVLctA-cBSVfuzIfFZul5LWg3qFk3Hw?e=N7FE7Z

Clases internas:

https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/ERvsMejeU9BIkN2Tlbs6SYUBC5_pcsv_9denbbHfb7b2XQ?e=Lh466L

SonarQube:

https://www.sonarqube.org/success-download-community-edition/

Seguir este manual para su correcta configuración:

https://docs.sonarqube.org/latest/setup/get-started-2-minutes/

EJTE-1 Ejercicio:

Hacer testing de ejercicio CRUD básico.

Ejecutar SonarQube sobre el ejercicio y que de un 70% de cobertura del código

TE2. Testing en SpringBoot. Cucumber (Opcional)

Duración: 2 Horas

https://pandemoniodigital.es/testing/2021/01/11/cucumber-spring-boot.html

https://thepracticaldeveloper.com/cucumber-tests-spring-boot-dependency-injection/#adding-dependency-injection-to-cucumber-tests

Spring Security.

Ver estos videos:

https://www.youtube.com/watch?v=CBA3I9TkzpQ&list=LL&index=95

https://www.youtube.com/watch?v=LG4GjdPXqz0&t=5s

SE1 Seguridad básica. Conceptos.

Videos (ingles):
https://bosonit.sharepoint.com/recursos/bosonitUni/Formacin/Forms/AllItems.aspx?id=%2Frecursos%2FbosonitUni%2FFormacin%2FCursos%2Fspring%20security

Seguridad en Spring MVC: http://www.profesor-p.com/2018/10/17/seguridad-web-en-spring-boot/

Clase interna con ejemplos prácticos de seguridad en Spring utilizando JWT:
https://bosonit.sharepoint.com/:v:/s/Desarrollo-Backend/EfX8lzbeUwVIlRVBAdLDP-4BcI7viXp0f-wYnO7j5IQ8MQ?e=VcjNUi

SE2 Oauth2.

Flujo Oauth2.

- https://www.digitalocean.com/community/tutorials/una-introduccion-a-oauth-2-es
- https://programacionymas.com/blog/protocolo-oauth-2

Ejemplo práctico con SpringBoot:

http://www.profesor-p.com/2018/10/18/securizando-servicios-rest-con-oauth2-en-springboot/

S3 JWT.

- ¿Qué es JWT? https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/
- Tutorial de ayuda si no se ha entendido bien
  https://www.youtube.com/watch?v=VVn9OG9nfH0
- Página referencia: https://jwt.io/
- Aplicaciones de ejemplo:
  - https://www.toptal.com/spring/spring-security-tutorial
  - https://blog.softtek.com/es/autenticando-apis-con-spring-y-jwt

    - https://savantdesilva.medium.com/enhancing-rest-api-security-with-spring-security-framework-and-jwt-ed138cac0806

- Exposición de compañeros de 2 posibles maneras de implementarlo:

  https://bosonit.sharepoint.com/:v:/s/Desarrollo-Backend/ETGJzIppIQBPpsiNpgHqjTsBds8FOsqRfoUEWhSdko9G-A?e=j9zrZH

EJS3-1 Ejercicio:

Añadir al CRUD del EJ3-1 (CRUD con relaciones entre tablas) personas seguridad.

Se añadirá a la tabla personas el campo

admin boolean [not null]

Después se creará el endpoint /login que pedirá usuario y password devolviendo un token JWT firmado si los datos son válidos comprobando contra la tabla 'personas'.

Si el usuario es admin permitirá acceder a todos los endpoints, si no lo es, solo a los endpoints de consulta.

Tener en cuenta que en este ejercicio NO estamos usando OAUTH2.

Mensajería.

ME1 Kafka

Duración: 4 Hora

Empezando con Kafka: https://kafka.apache.org/quickstart

Video interno hablando sobre Kafka: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EVUn9U3JUpZNkkEk8P-yhsIBP8-jqKNvhzwSjyS0Nc7Gdw?e=McCctk

Videos de introducción: https://www.confluent.io/learn/kafka-tutorial/ (INGLES)

https://www.paradigmadigital.com/dev/comunicacion-microservicios-apache-kafka/

http://www.profesor-p.com/2019/01/24/mensajeria-con-kafka-y-spring-boot/

https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-1/

https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-2-2/

https://kafka.apache.org/documentation/

Videos: https://kafka.apache.org/28/documentation/streams/

Programa para controlar Kafka: https://www.kafkatool.com/

ME2 RabbitMQ

Video interno explicando conceptos y programa de ejemplo: https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EXuZwVA5W2FEv3Bkq6sTlkcB5IrdPN91pq8WqrgNhCrLJA?e=OAUC8d

Instalación: https://www.rabbitmq.com/download.html

https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html

RabbitMQ : Message Queues for beginners

Exchanges:

https://www.cloudamqp.com/blog/part4-rabbitmq-for-beginners-exchanges-routing-keys-bindings.html

Spring Boot: https://docs.spring.io/spring-amqp/reference/html/

Ejemplos programas en SpringBoot:

https://github.com/chuchip/rabbitMQTest (programa utilizado en el video inicial)

https://spring.io/guides/gs/messaging-rabbitmq/

https://www.rabbitmq.com/tutorials/tutorial-four-spring-amqp.html

EJKA-1:

Usando Kafka, montar dos aplicaciones que una mande mensajes y la segunda le responda con otros mensajes.

Spring Cloud.

Teoría. Conceptos.

Paquetes disponibles: https://spring.io/projects/spring-cloud

Hay que tener en cuenta que la mayoría de los paquetes de Netflix han sido declarados obsoletos.

En este artículo se habla de las nuevas tecnologías usadas (a fecha del 2022).
https://piotrminkowski.com/2020/05/01/a-new-era-of-spring-cloud/

Spring Cloud Config Server.

- Spring Cloud Config server utilizando un servidor GIT externo: http://www.profesor-p.com/2018/12/10/servidor-configuraciones-usando-spring-cloud/
- Alternativa usando **Spring Cloud Consul**: https://cloud.spring.io/spring-cloud-consul/reference/html/

Feign y Ribbon y Hyxtris.

Importante: Ribbon está obsoleto, en su lugar ahora se utiliza 'Spring Cloud Load Balancer'. Mirar este documento: https://spring.io/guides/gs/spring-cloud-loadbalancer/

Clase interna. Duración: 2 Hora

http://www.profesor-p.com/2019/01/03/microservicios-distribuidos-con-eureka/

Eureka Namming Server. Consul

Duración: 2 Hora

Eureka (deprecated): http://www.profesor-p.com/2019/01/03/microservicios-distribuidos-con-eureka/

Consul: https://www.paradigmadigital.com/dev/spring-cloud-consul-1-2-descubrimiento-microservicios/

Spring Cloud Gateway

- Página oficial: https://spring.io/projects/spring-cloud-gateway
- Ejemplo de uso: http://www.profesor-p.com/spring/gateway/

Zuul (Obsoleto)

Duración: 1 Hora

http://www.profesor-p.com/2019/03/16/zuul-para-redirigir-peticiones-rest-en-spring-boot/

Sleuth

Duración: 1 Hora

https://spring.io/projects/spring-cloud-sleuth

https://www.paradigmadigital.com/dev/trazabilidad-distribuida-spring-cloud-sleuth-zipkin/

Ejercicio FINAL entorno cloud

https://bosonit.sharepoint.com/:w:/s/Desarrollo-Backend/EbXPdIbD2VhNkxhoDFVlEqkBDrULBoBrvd7CaxuFXFp_RQ?e=jcUKhx

video explicación ejemplo

Spring Boot - Proyecto Final

Avanzado segunda parte

DBA3. Acceso a base de datos. Configuración de conexión. Uso de JDBCTemplate.

https://howtodoinjava.com/spring-boot2/datasource-configuration/

https://www.journaldev.com/17053/spring-jdbctemplate-example

https://docs.spring.io/spring-data/jdbc/docs/current/reference/html/#reference

http://www.profesor-p.com/2018/08/22/acceso-a-base-de-datos-con-jdbc-spring/

Baeldung: https://www.baeldung.com/spring-jdbc-jdbctemplate

Video: https://www.youtube.com/watch?v=CJjHdchLY9Y&t=32s

Ejercicio

Realizar CRUD de personas, utilizando JDBCTemplate.

DBA4. Conceptos avanzados JPA

Duración: 4 Hora

Mejorando rendimientos select:

Jpa JOIN: https://www.arquitecturajava.com/jpa-join-fetch-uso/

JPA EntityGraph:

- https://refactorizando.com/entity-graph-jpa-spring-boot/
- https://www.arquitecturajava.com/un-ejemplo-de-jpa-entity-graph/

Audit:

https://springbootdev.com/2018/03/13/spring-data-jpa-auditing-with-createdby-createddate-lastmodifiedby-and-lastmodifieddate/

Listener (events)

https://www.arquitecturajava.com/jpa-entity-listener/

https://vladmihalcea.com/hibernate-event-listeners/

Varios:

https://www.baeldung.com/tag/jpa/

SA3 SPEL. Spring Expresion Language.

Duración: 1 Hora.

https://docs.spring.io/spring-framework/docs/4.3.12.RELEASE/spring-framework-reference/html/expressions.html

https://www.baeldung.com/spring-expression-language

SA4 Crear anotaciones en Spring Boot.

Ejemplos:

- https://fullstackdeveloper.guru/2021/06/15/how-to-create-a-custom-annotation-in-spring-boot/
- https://www.adictosaltrabajo.com/2015/08/10/crear-anotaciones-propias-en-java/

Guía: https://www.baeldung.com/java-custom-annotation

SA5 Eventos en Spring

Recibiendo y generando eventos en Spring.

Video:

- https://www.youtube.com/watch?v=4oBqCtdRIYo

Artículos:

- https://danielme.com/2019/06/02/spring-sistema-de-eventos/
- https://www.baeldung.com/spring-events

Node:

Leer documento https://bosonit.sharepoint.com/:w:/s/Desarrollo-Backend/EVB69vF-6LVIlVd9sXc-XPwBEoG6LoJ6iVq6bMb3gETGsw?e=AHDCnR

Opcional:

Spring reactivo

- Spring **WebFlux (Api Reactiva)**
  - **WebClient. (**http://www.profesor-p.com/webclient/)
  - **Proyecto Reactor.**

  Tiempo estimado: 6 Horas.

Spring Batch

Página del proyecto: https://spring.io/projects/spring-batch

Video: Spring Framework en español - Ejemplo práctico con Spring Batch - 4SoftwareDevelopers

Video sobre integración y control de los batchs en la empresa:
https://www.youtube.com/watch?v=bhFBtNiZYYY&t=2150s

Ejemplos prácticos:

https://spring.io/guides/gs/batch-processing/

https://gustavopeiretti.com/como-crear-una-aplicacion-con-spring-batch/

Ejercicios: Hacer un programa que lea de un csv y lo guarde en una base de datos H2. El proceso se tiene que ejecutar automaticamente cada 5 minutos.

https://spring.io/guides/gs/scheduling-tasks/

Extra: Usar Spring Cloud Data Flow: https://www.adictosaltrabajo.com/2021/01/11/primeros-pasos-con-spring-cloud-data-flow-ejecucion-local/

Microservicios

- Docker.
    - Docker Compose.
    - Docker Swarm (Conceptos)

    - Kubernate(Conceptos)
- Integración Continua.
    - Jenkins.
        - GitHub/GitLab - Jenkins.

Bibliografía:

Contenido del curso de Spring Boot dado por Pivotal.

https://d1fto35gcfffzn.cloudfront.net/education/Spring-Core-Training.pdf

Staffit

Explicación de qué es Staffit, como configurarlo por primera vez y partes que lo componen

https://bosonit.sharepoint.com/:v:/s/FullStackEnero2020-Back1er_Trimestre2021/EfHvKaxEzJZDlawvYrQ79LkBEWePj27Ax-YfPg2cMPT4Fg?e=PCNpdI

Para más información sobre como compilar los módulos de Staffit:

https://bosonit-my.sharepoint.com/:v:/p/jesus_javier/EV9YcI7UqPtGh6MGa7h5RfABoGriqYbED0YXz_Fr-dBqEg?e=sDol7q

El repositorio de staffit está en: http://192.168.10.210/staffit/back/java/cloud/ aunque en los videos habla de una copia que se utiliza para pruebas.

Opcional:

J2EE

- Crear aplicación RESTFUL – 8 Horas.
- Crear aplicación JSP/JSF con manejo sesiones – 16 Horas.

JSP:
https://www.youtube.com/watch?v=coK4jM5wvko&list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk

JSF: Curso completo:
https://www.youtube.com/playlist?list=PLU8oAlHdN5BktAXdEVCLUYzvDyqRQJ2lk A partir del 228 al 245

Ejemplo: https://www.youtube.com/watch?v=dZnXn-ecwp8

- JBOSS (WildFly) - 6 Horas
- GlassFish. (Jakarta EE) - 6 Horas.

Herramientas:

Grafana: https://grafana.com/

ElasticSearch: https://www.elastic.co/es/elasticsearch/

Cloud:

Google App Engine. (https://cloud.google.com/appengine)

Patrones de diseño:

Patrones de diseño en Java ¿Qué son y en qué consisten? ▷➞ VidaBytes ▷➞ | VidaBytes ▷➞