# Threads

## Creación de hilo:

* + - Para crear un hilo se necesita primero crear una clase que extienda de la interfaz Runnable e implementar el método run(). Posterior crear un objeto Thread y mandar en constructor el objeto Runnable que creamos.

## Detener hilos:

* + - join() - Esta función detiene un hilo hasta que otro termine. Se utiliza si un hilo padre depende de la finalización de otro hilo hijo. El hilo padre llama el método join() de su hijo.

## Sincronización de hilos:

* + - synchronized - Esta palabra reservada se puede utilizar en un método o en un bloque. Sirve para sincronizar los hilos para su ejecución. Dentro de un método o bloque sincronizado se pueden utilizar los métodos wait() con el que detendrá la ejecución de un hilo y le dará el foco a otro hilo que esté disponible y notify/notifyAll() que liberará los hilos que se hayan detenido con wait().

# Servlets

* + Es una clase que hereda de HttpServlet y contiene los métodos doGet, doPost.

**Ejemplo:**

public class LoginServlet **extends** HttpServlet **{**

* + Se debe utilizar la notación @WebServlet(“/id”) donde se especificará el ID del servlet y el que se utilizará para mandarlo llamar desde un jsp o HTML.

**Ejemplo:**

**LoginServlet.java**

@WebServlet**(**"/login"**)**

public class LoginServlet **extends** HttpServlet **{**

**index.jsp/index.html**

<form action=**"login"** method=**"post"**>

**Name:** <input type=**"text"** name=**"name"**>

<br>

**Password:** <input type=**"password"** name=**"password"**>

<br>

<input type=**"submit"** value=**"login"**>

</form>

* + Los beans que se usan en Java EE deben de implementar la interfaz Serializable.

**Ejemplo:**

public class Login **implements** Serializable **{**

* + Depende del método que utilicemos para para enviar la información de una forma, es el método que implementaremos en nuestro servlet.

**Ejemplo:**

**index.jsp/index.html**

<form action=**"login"** method=**"post"**>

**Name:** <input type=**"text"** name=**"name"**>

<br>

**Password:** <input type=**"password"** name=**"password"**>

<br>

<input type=**"submit"** value=**"login"**>

</form>

**LoginServlet.java**

@WebServlet**(**"/login"**)**

public class LoginServlet **extends** HttpServlet **{**

protected void doPost**(**HttpServletRequest request**,**

HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

...

**}**

**}**

* + Para llamar una página desde un Servlet se utiliza la clase RequestDispatcher el método forward(request, response).

**Ejemplo:**

protected void doPost**(**HttpServletRequest request**,**

HttpServletResponse response**)**

**throws** ServletException**,** IOException **{**

String name **=** request**.**getParameter**(**"name"**);**

String password **=** request**.**getParameter**(**"password"**);**

Login login **=** **new** Login**();**

login**.**setName**(**name**);**

login**.**setPassword**(**password**);**

boolean authenticated **=** login**.**authentication**();**

RequestDispatcher dispatcher **=** **null;**

**if(**authenticated**)** **{**

dispatcher **=** request**.**getRequestDispatcher**(**"welcome.jsp"**);**

**}** **else** **{**

dispatcher **=** request**.**getRequestDispatcher**(**"error.jsp"**);**

**}**

dispatcher**.**forward**(**request**,** response**);**

**}**

# Maven

* + Es un asistente inteligente creado por Apache que nos ayudan a crear proyectos basados en plantillas, descargar librerías y crear ejecutables. Está basado en XML por lo que la configuración del proyecto estará en un archivo llamado pom.xml. En este archivo describiremos el proyecto a construir, las dependencias que utilizaremos y como se empaquetará el proyecto.
  + Si se presenta problemas para crear proyectos usando archetypes se tiene verificar que el archivo en “~/.m2/settings.xml” contenga la configuración correcta de los proxies tanto de http como de https. Por último se tienen que borrar todo lo que tenemos en la carpeta “~/.m2/repository/” para que vuelva a cargar todas las dependencias que necesita el proyecto.
  + Para obtener las dependencias que debemos poner en el pom.xml de nuestro proyecto podemos hacerlo en “https://mvnrepository.com/”

# Gradle

* + Asistente inteligente que está basado en Groovy y usa Json. La configuración del proyecto estará en un archivo llamado build.gradle. En este archivo describiremos el proyecto a construir, las dependencias que utilizaremos y como se empaquetará el proyecto.

# Hibernate

* + Es una ORM (Object Relational Mapping).
  + Es recomendable instalar un plugin para que podamos utilizar Hibernate de una forma más cómoda desde nuestro IDE. Para instalarla tenemos que ir a Help🡪Install new software🡪Add. Para obtener el plugin correcto para la versión de Eclipse que tenemos es necesario obtenerla desde <https://download.jboss.org/jbosstools/> y elegir la versión que tenemos de Eclipse, /stable/updates; en nuestro caso: <https://download.jboss.org/jbosstools/oxygen/stable/updates/>.
  + Para configurar la conexión a la base de datos se puede hacer de 2 maneras:
    - Con un archivo “hibernate.cfg.xml” donde se tendrá los datos de conexión a la base de datos. Este archivo xml se creará en un paquete dentro de src/main que se llame “resources”. Para crearlo se puede hacer desde “New🡪Other🡪Hibernate🡪Hibernate Configuration File” (Esta opción sale gracias al plugin instalado para Eclipse).

**Ejemplo:**

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=**"hibernate.connection.driver\_class"**>

**com.mysql.jdbc.Driver**</property>

<property name=**"hibernate.connection.url"**>

**jdbc:mysql://localhost:3306/profesores**</property>

<property name=**"hibernate.connection.username"**>

**profesores**</property>

<property name=**"hibernate.connection.password"**>

**profesores**</property>

<property name=**"hibernate.dialect"**>

**org.hibernate.dialect.MySQLDialect**</property>

<property name=**"show\_sql"**>**true**</property>

<mapping class=**"com.javaee.Hibernate.model.SocialMedia"**/>

<mapping class=**"com.javaee.Hibernate.model.Course"**/>

<mapping class=**"com.javaee.Hibernate.model.Teacher"**/>

<mapping class=

**"com.javaee.Hibernate.model.TeacherSocialMedia"**/>

</session-factory>

</hibernate-configuration>

* + - Con una clase que contenga la anotación @Configuration dentro de un paquete que se llame “configuration”. Podemos utilizar también la anotación @EnableTransactionManagement para indicarle a Spring que todas las operaciones en la base de datos las harán de forma transaccional.

**Ejemplo:**

@Configuration

@EnableTransactionManagement

@PropertySource**(**value **=** **{** "classpath:application.properties" **})**

public class HibernateConfiguration **{**

@Autowired

private Environment env**;**

@Bean

public LocalSessionFactoryBean sessionFactory**()** **{**

LocalSessionFactoryBean sessionFactoryBean **=**

**new** LocalSessionFactoryBean**();**

sessionFactoryBean**.**setDataSource**(**dataSource**());**

sessionFactoryBean**.**setPackagesToScan**(**

"com.practices.teachers.model"**);**

sessionFactoryBean**.**setHibernateProperties**(**

hibernateProperties**());**

**return** sessionFactoryBean**;**

**}**

@Bean

public DataSource dataSource**()** **{**

DriverManagerDataSource dataSource **=**

**new** DriverManagerDataSource**();**

dataSource**.**setDriverClassName**(**

env**.**getRequiredProperty**(**"jdbc.driverClassName"**));**

dataSource**.**setUrl**(**env**.**getRequiredProperty**(**"jdbc.url"**));**

dataSource**.**setUsername**(**

env**.**getRequiredProperty**(**"jdbc.username"**));**

dataSource**.**setPassword**(**

env**.**getRequiredProperty**(**"jdbc.password"**));**

**return** dataSource**;**

**}**

public Properties hibernateProperties**()** **{**

Properties properties **=** **new** Properties**();**

properties**.**put**(**"hibernate.dialect"**,**

"org.hibernate.dialect.MySQLDialect"**);**

properties**.**put**(**"show\_sql"**,** "true"**);**

**return** properties**;**

**}**

// public HibernateTransactionManager transactionManager() {

// HibernateTransactionManager hibernateTransactionManager =

// new HibernateTransactionManager();

// hibernateTransactionManager.setSessionFactory(

// sessionFactory().getObject());

//

// return hibernateTransactionManager;

// }

**}**

* + Beans
    - Anotaciones para los beans (utilizar las anotaciones de la paquetería javax.persistence):
      * @Entity - Para indicar que la clase será una entidad.
      * @Table(name = “<nombre de la tabla>”) - Para indicar el nombre de la tabla que estará mapeada en la clase.
      * @Column(name = “<nombre de la columna>”) - Para indicar el nombre de la columna que mapeará el atributo.
      * @Id - Para indicar que ese atributo será llave primaria.
      * @GeneratedValue(strategy = GenerationType.IDENTITY) - Para indicar que el campo será autoincrementable.
      * @JsonIgnore – Para que en la respuesta no se enviar ese campo. Útil para evitar redundancias cuando hay referencias con otros objetos y esos objetos a su vez tienen referencia al primero, si no se incluyera esta anotación mandará error porque se vuelve una respuesta infinita.
      * @OneToOne - Para indicar una relación uno a uno entre 2 entidades, puede ser unidireccional para que una entidad tenga un objeto de la entidad relacionada o bidireccional para que las 2 entidades tengan objetos que se relacionan entre sí. Se puede utilizar especificaciones adicionales como cascade = CascadeType.ALL para hacer que cuando se afecte una entidad también afecte la entidad relacionada.

**Ejemplo:**

@OneToOne**(**cascade **=** CascadeType**.**ALL**)**

* + - * @OneToMany y @ManyToOne - Para indicar relación uno a muchos de 2 entidades. Se tiene que establecer cuál de las 2 entidades será la owner y cual la no owner.
        + Entity Owner

@OneToMany**(**mappedBy **=** “entidad”**,**

cascade **=** CascadeType**.**ALL**,**

fetch **=** FetchType**.**EAGER**)**

* + - * + Entity No Owner. Se utilizarán 2 anotaciones:

@ManyToOne**(**optional **=** **true,** fetch **=** FetchType**.**EAGER**)**

@JoinColumn**(**name **=** "primary\_key"**)**

* + - * @ManyToMany - Para indicar relación muchos a muchos de 2 entidades. Se tiene que establecer cuál de las 2 entidades será la owner y cual la no owner.
        + Entity Owner

@ManyToMany**(**cascade **=** CascadeType**.**ALL**,**

joinColumns **=** **{**@JoinColumn**(**name **=**

“campo\_tabla\_owner”**)},**

inverseJoinColumns **=** **{**@JoinColumn**(**name **=**

“campo\_tabla\_hijo”**)})**

* + - * + Entity No Owner

@ManyToMany**(**mappedBy **=** “entidad”**,**

cascade **=** CascadeType**.**ALL**)**

* + - Una vez teniendo las entidades creadas y mapeadas, tenemos que indicarle con que beans trabajaremos:
      * Si se está haciendo con archivo de configuración en XML, se agregan al archivo de configuración “cfg.xml” dentro de las etiquetas <session-factory>

<mapping class=**"com.javaee.Hibernate.model.SocialMedia"**/>

<mapping class=**"com.javaee.Hibernate.model.Course"**/>

<mapping class=**"com.javaee.Hibernate.model.Teacher"**/>

<mapping class=**"com.javaee.Hibernate.model.TeacherSocialMedia"**/>

* + - * Si se está haciendo desde una clase de configuración entonces indicar en que paquetería estarán nuestros beans:

sessionFactoryBean**.**setPackagesToScan**(**

"com.practices.teachers.model"**);**

# Spring

## DAO

* + - Se crea una clase abstracta en el paquete “dao” para obtener la sesión de la base de datos.

**Ejemplo:**

public abstract class AbstractSession **{**

@Autowired

private SessionFactory sessionFactory**;**

protected Session getSession**()** **{**

**return** sessionFactory**.**getCurrentSession**();**

**}**

**}**

* + - Se necesitan hacer Interfaces en el paquete “dao” con acciones a realizar en la base de datos y una clase que implemente de esa interfaz por cada Bean. Estas clases deben extender de la clase abstracta para poder obtener la sesión de la conexión.
    - Anotaciones para los DAO:
      * @Repository – A todas las implementaciones de DAO se debe indicar que es de la capa de repositorio agregando esta anotación.
      * @Transactional – Para indicar que todas las acciones serán de forma transaccional.
    - Para crear sesión en la base de datos (si no se hace con la clase abstracta):

Configuration configuration **=** **new** Configuration**();**

configuration**.**configure**();**

SessionFactory sessionFactory **=** configuration**.**buildSessionFactory**();**

Session sesión **=** sessionFactory**.**openSession**();**

* + - Para realizar insert en la base de datos:

public void saveTeacher**(**Teacher teacher**)** **{**

Long idNewTeacher **=** **(**Long**)** getSession**().**save**(**teacher**);**

**if** **(**idNewTeacher **!=** **null)** **{**

System**.**out**.**println**(**"Id del profesor nuevo: " **+** idNewTeacher**);**

**}** **else** **{**

System**.**out**.**println**(**"Error al guardar profesor"**);**

**}**

**}**

* + - Para realizar consulta por id:

public Teacher findTeacherById**(**Long idTeacher**)** **{**

**return** getSession**().**get**(**Teacher**.**class**,** idTeacher**);**

**}**

* + - Para realizar consulta sin filtros:

@SuppressWarnings**(**"unchecked"**)**

public List**<**Teacher**>** findAllTeachers**()** **{**

**return** getSession**().**createQuery**(**"from Teacher"**).**list**();**

**}**

* + - Para realizar consulta con filtros:

public TeacherSocialMedia findSocialMediaByIdAndNickname**(**

Long idSocialMedia**,** String nickname**)** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Object**[]>** objects **=** getSession**().**createQuery**(**""

**+** "from TeacherSocialMedia tsm "

**+** "join tsm.socialMedia sm "

**+** "where sm.idSocialMedia = :idSocialMedia "

**+** "and tsm.nickname = :nickname"**)**

**.**setParameter**(**"idSocialMedia"**,** idSocialMedia**)**

**.**setParameter**(**"nickname"**,** nickname**)**

**.**list**();**

**if** **(**objects**.**size**()** **>** 0**)** **{**

**for** **(**Object**[]** objects2 **:** objects**)** **{**

**for** **(**Object object **:** objects2**)** **{**

**if** **(**object **instanceof** TeacherSocialMedia**)** **{**

**return** **(**TeacherSocialMedia**)** object**;**

**}**

**}**

**}**

**}**

**return** **null;**

**}**

* + - Para realizar update a la base de datos:

public void updateTeacher**(**Teacher teacher**)** **{**

getSession**().**update**(**teacher**);**

**}**

* + - Para realizar delete a la base de datos:

public void deleteTeacher**(**Long idTeacher**)** **{**

Teacher teacher **=** findTeacherById**(**idTeacher**);**

**if** **(**teacher **!=** **null)** **{**

Iterator**<**TeacherSocialMedia**>** i **=**

teacher**.**getTeacherSocialMedias**().**iterator**();**

**while** **(**i**.**hasNext**())** **{**

TeacherSocialMedia teacherSocialMedia **=** i**.**next**();**

i**.**remove**();**

getSession**().**delete**(**teacherSocialMedia**);**

**}**

teacher**.**getTeacherSocialMedias**().**clear**();**

getSession**().**delete**(**teacher**);**

**}** **else** **{**

System**.**out**.**println**(**"Error al borrar profesor"**);**

**}**

**}**

## Services

* + - Se crean interfaces en paquete “service” con los mismos métodos que las interfaces DAO por cada servicio (que será lo mismo que por cada Bean).
    - Se crea clase que implementa cada interfaz en paquetería “service” mandando llamar los métodos de DAO.

**Ejemplo:**

@Service**(**"courseService"**)**

@Transactional

public class CourseServiceImpl **implements** CourseService **{**

@Autowired

private CourseDao \_courseDao**;**

@Override

public void saveCourse**(**Course course**)** **{**

\_courseDao**.**saveCourse**(**course**);**

**}**

...

**}**

* + - Anotaciones para los Services:
      * @Service(“[identificador]”)
      * @Transactional

## Controllers

* + - Se crea un controlador por cada entidad a manejar.
    - Las funciones dentro del controlador deben retornar un ResponseEntity<E>.

**Ejemplo:**

@Controller

@RequestMapping**(**"/v1"**)**

public class TeacherController **{**

@Autowired

TeacherService \_teacherService**;**

@Autowired

SocialMediaService \_socialMediaService**;**

//POST

@RequestMapping**(**value **=** "/teacher"**,**

method **=** RequestMethod**.**POST**,**

headers **=** "Accept=application/json"**,**

consumes **=** "application/json"**)**

public ResponseEntity**<?>** createTeacher**(**

@RequestBody Teacher teacher**,**

UriComponentsBuilder uriBuilder**)** **{**

**if(**teacher**.**getName**()** **==** **null** **||** teacher**.**getName**().**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher is incomplete"**).**build**();**

**}**

\_teacherService**.**saveTeacher**(**teacher**);**

Teacher teacher2 **=** \_teacherService**.**findTeacherByName**(**

teacher**.**getName**());**

**return** ResponseEntity

**.**created**(**uriBuilder

**.**path**(**"/v1/teacher/{id}"**)**

**.**buildAndExpand**(**teacher2**.**getIdTeacher**())**

**.**toUri**())**

**.**build**();**

**}**

**}**

* + - Anotaciones para la clase Controller:
      * **@Controller** – Con esta anotación decimos que la clase es un controlador.
      * **@RequestMapping(“/”)** – Para indicar la ruta en la que escuchará el controlador.
    - Anotaciones para las funciones dentro de los controladores
      * **@RequestMapping(value=”[ruta]”, method=RequestMethod.[metodo], headers = ”Accept=application/json”)** – Para indicar la ruta en la que escuchará la función.
      * **@ResponseBody** – Para indicar que esa función responderá algo al cuerpo de la página.
      * **@PathVariable** – Para obtener un valor de la URL mandado como variable.

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher/{id}"**,**

method **=** RequestMethod**.**GET**,**

produces **=** "application/json"**)**

public ResponseEntity**<**Teacher**>** getTeacherById**(**

@PathVariable**(**"id"**)** Long idTeacher**)** **{**

**if(**idTeacher **==** **null** **||** idTeacher **<=** 0L**)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "idTeacher is incorrect"**)**

**.**build**();**

**}**

Teacher teacher **=** \_teacherService**.**findTeacherById**(**idTeacher**);**

**if(**teacher **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher not found"**).**build**();**

**}**

**return** ResponseEntity**.**ok**(**teacher**);**

**}**

* + - * **@RequestBody** – Para obtener un objeto mandado al servicio.

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher"**,**

method **=** RequestMethod**.**POST**,**

headers **=** "Accept=application/json",

consumes **=** "application/json"**)**

public ResponseEntity**<?>** createTeacher**(**

@RequestBody Teacher teacher**,**

UriComponentsBuilder uriBuilder**)** **{**

**if(**teacher**.**getName**()** **==** **null** **||** teacher**.**getName**().**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher is incomplete"**).**build**();**

**}**

\_teacherService**.**saveTeacher**(**teacher**);**

Teacher teacher2 **=** \_teacherService**.**findTeacherByName**(**

teacher**.**getName**());**

**return** ResponseEntity

**.**created**(**uriBuilder

**.**path**(**"/v1/teacher/{id}"**)**

**.**buildAndExpand**(**teacher2**.**getIdTeacher**())**

**.**toUri**())**

**.**build**();**

**}**

* + - * **@RequestParam –** Para obtener parámetros desde la URL (?name=linkedIn).

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher"**,**

method **=** RequestMethod**.**GET**,**

produces **=** "application/json"**)**

public ResponseEntity**<**List**<**Teacher**>>** getTeachers**(**

@RequestParam**(**value **=** "name"**,** required **=** **false)**

String name**)** **{**

List**<**Teacher**>** lstTeachers **=** **new** ArrayList**<>();**

**if(**name **==** **null** **||** name**.**isEmpty**())** **{**

lstTeachers **=** \_teacherService**.**findAllTeachers**();**

**if(**lstTeachers**.**isEmpty**())** **{**

**return** ResponseEntity**.**noContent**()**

**.**header**(**"error"**,** "Teachers not found"**)**

**.**build**();**

**}**

**return** ResponseEntity**.**ok**(**lstTeachers**);**

**}** **else** **{**

Teacher teacher **=** \_teacherService**.**findTeacherByName**(**name**);**

**if(**teacher **==** **null)** **{**

**return** ResponseEntity**.**noContent**()**

**.**header**(**"error"**,** "Teacher not found"**).**build**();**

**}**

lstTeachers**.**add**(**teacher**);**

**return** ResponseEntity**.**ok**(**lstTeachers**);**

**}**

**}**

* + - Clases útiles que se utilizan dentro de las funciones:
      * UriComponentBuilder – Esta clase se puede poner como parámetro de una función para poder generar una URI y mandarla llamar en el cliente.

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher"**,**

method **=** RequestMethod**.**POST**,**

headers **=** "Accept=application/json",

consumes **=** "application/json"**)**

public ResponseEntity**<?>** createTeacher**(**

@RequestBody Teacher teacher**,**

UriComponentsBuilder uriBuilder**)** **{**

**if(**teacher**.**getName**()** **==** **null** **||** teacher**.**getName**().**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher is incomplete"**).**build**();**

**}**

\_teacherService**.**saveTeacher**(**teacher**);**

Teacher teacher2 **=** \_teacherService**.**findTeacherByName**(**

teacher**.**getName**());**

**return** ResponseEntity

**.**created**(**uriBuilder

**.**path**(**"/v1/teacher/{id}"**)**

**.**buildAndExpand**(**teacher2**.**getIdTeacher**())**

**.**toUri**())**

**.**build**();**

**}**

* + - Guardar una imagen en el servidor:

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher/{id\_teacher}/image"**,**

method **=** RequestMethod**.**POST**,**

headers **=** "Accept=multipart/form-data"**,**

consumes **=** "multipart/form-data"**,**

produces **=** "image/jpeg"**)**

public ResponseEntity**<**byte**[]>** uploadTeacherImage**(**

@PathVariable**(**"id\_teacher"**)** Long idTeacher**,**

@RequestParam**(**"file"**)** MultipartFile multipartFile**)** **{**

**if(**idTeacher **==** **null** **||** idTeacher **<=** 0L**)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "idTeacher is incorrect"**).**build**();**

**}**

**if(**multipartFile **==** **null** **||** multipartFile**.**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "File is incorrect"**).**build**();**

**}**

Teacher teacher **=** \_teacherService**.**findTeacherById**(**idTeacher**);**

**if(**teacher **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher not found"**).**build**();**

**}**

**if** **(**teacher**.**getAvatar**()** **!=** **null** **||**

**!**teacher**.**getAvatar**().**isEmpty**())** **{**

String fileName **=** teacher**.**getAvatar**();**

Path path **=** Paths**.**get**(**fileName**);**

File f **=** path**.**toFile**();**

**if(**f**.**exists**())** **{**

f**.**delete**();**

**}**

**}**

**try** **{**

Date date **=** **new** Date**();**

SimpleDateFormat sdf **=** **new** SimpleDateFormat**(**

"yyyy-MM-dd-HH-mm-ss"**);**

String dateName **=** sdf**.**format**(**date**);**

String fileName **=** String**.**valueOf**(**idTeacher**)**

**+** "-pictureTeacher-" **+** dateName **+** "."

**+** multipartFile**.**getContentType**().**split**(**"/"**)[**1**];**

teacher**.**setAvatar**(**TEACHER\_UPLOADED\_FOLDER **+** fileName**);**

byte**[]** bytes **=** multipartFile**.**getBytes**();**

Path path **=** Paths**.**get**(**TEACHER\_UPLOADED\_FOLDER **+** fileName**);**

Files**.**write**(**path**,** bytes**);**

\_teacherService**.**updateTeacher**(**teacher**);**

**return** ResponseEntity**.**ok**().**contentType**(**MediaType**.**IMAGE\_JPEG**)**

**.**body**(**bytes**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Error to upload image"**).**build**();**

**}**

**}**

* + - Consultar una imagen del servidor:

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher/{id\_teacher}/image"**,**

method **=** RequestMethod**.**GET**,**

produces **=** "image/jpeg"**)**

public ResponseEntity**<**byte**[]>** getTeacherImage**(**

@PathVariable**(**"id\_teacher"**)** Long idTeacher**)** **{**

**if(**idTeacher **==** **null** **||** idTeacher **<=** 0L**)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "idTeacher is incorrect"**).**build**();**

**}**

Teacher teacher **=** \_teacherService**.**findTeacherById**(**idTeacher**);**

**if(**teacher **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher not found"**).**build**();**

**}**

**try** **{**

String fileName **=** teacher**.**getAvatar**();**

Path path **=** Paths**.**get**(**fileName**);**

File f **=** path**.**toFile**();**

**if(!**f**.**exists**())** **{**

**return** ResponseEntity**.**noContent**()**

**.**header**(**"error"**,** "Imagen not found"**).**build**();**

**}**

byte**[]** image **=** Files**.**readAllBytes**(**path**);**

**return** ResponseEntity**.**ok**().**contentType**(**MediaType**.**IMAGE\_JPEG**)**

**.**body**(**image**);**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Error to show image"**).**build**();**

**}**

**}**

* + - Borrar una imagen del servidor:

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher/{id\_teacher}/image"**,**

method **=** RequestMethod**.**DELETE**)**

public ResponseEntity**<?>** deleteTeacherImage**(**

@PathVariable**(**"id\_teacher"**)** Long idTeacher**)** **{**

**if(**idTeacher **==** **null** **||** idTeacher **<=** 0L**)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "idTeacher is incorrect"**).**build**();**

**}**

Teacher teacher **=** \_teacherService**.**findTeacherById**(**idTeacher**);**

**if(**teacher **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher not found"**).**build**();**

**}**

**if(**teacher**.**getAvatar**()** **==** **null** **||** teacher**.**getAvatar**().**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"error"**,** "Teacher does't have image assigned"**)**

**.**build**();**

**}**

String fileName **=** teacher**.**getAvatar**();**

Path path **=** Paths**.**get**(**fileName**);**

File file **=** path**.**toFile**();**

**if(**file**.**exists**())** **{**

file**.**delete**();**

**}**

teacher**.**setAvatar**(**""**);**

\_teacherService**.**updateTeacher**(**teacher**);**

**return** ResponseEntity**.**noContent**().**build**();**

**}**

* + - Guardar en tabla intermedia de relación muchos a muchos:

**Ejemplo:**

@RequestMapping**(**value **=** "/teacher/socialMedia"**,**

method **=** RequestMethod**.**PATCH**,**

headers **=** "Accept=application/json"**)**

public ResponseEntity**<?>** assignTeacherSocialMedia**(**

@RequestBody Teacher teacher**,**

UriComponentsBuilder uriBuiler**)** **{**

//Teacher.idTeacher's validation

**if(**teacher**.**getIdTeacher**()** **==** **null** **||** teacher**.**getIdTeacher**()** **<=** 0**)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"ERROR"**,** "We need id\_teacher"**)**

**.**body**(**"We need id\_teacher"**);**

**}**

//Validation if teacher don't exist in DB.

Teacher teacherSaved **=** \_teacherService**.**findTeacherById**(**

teacher**.**getIdTeacher**());**

**if(**teacherSaved **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"ERROR"**,** "Teacher not found"**)**

**.**body**(**"Teacher not found"**);**

**}**

//Teacher.teacherSocialMedias validation

**if(**teacher**.**getTeacherSocialMedias**()** **==** **null** **||**

teacher**.**getTeacherSocialMedias**().**isEmpty**())** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"ERROR"**,** "We need teacherSocialMedia"**)**

**.**body**(**"We need teacherSocialMedia"**);**

**}**

Iterator**<**TeacherSocialMedia**>** i **=**

teacher**.**getTeacherSocialMedias**().**iterator**();**

**while(**i**.**hasNext**())** **{**

TeacherSocialMedia teacherSocialMedia **=** i**.**next**();**

//Teacher.teacherSocialMedias.socialMedia validation

**if(**teacherSocialMedia**.**getSocialMedia**()** **==** **null** **||**

teacherSocialMedia**.**getSocialMedia**()**

**.**getIdSocialMedia**()** **==** **null** **||**

teacherSocialMedia**.**getNickname**()** **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"ERROR"**,**

"We need id\_social\_media and nickname"**)**

**.**body**(**"We need id\_social\_media and nickname"**);**

**}**

//Validation if socialMedia don't exist in DB

SocialMedia socialMediaSaved **=**

\_socialMediaService**.**findSocialMediaById**(**

teacherSocialMedia**.**getSocialMedia**()**

**.**getIdSocialMedia**());**

**if(**socialMediaSaved **==** **null)** **{**

**return** ResponseEntity**.**badRequest**()**

**.**header**(**"ERROR"**,** "IdSocialMedia not found"**)**

**.**body**(**"IdSocialMedia not found"**);**

**}**

teacherSocialMedia**.**setSocialMedia**(**socialMediaSaved**);**

teacherSocialMedia**.**setTeacher**(**teacherSaved**);**

//Validation if teacherSocialMedia exist in DB

TeacherSocialMedia teacherSocialMediaSaved **=**

\_socialMediaService

**.**findSocialMediaByIdTeacherAndIdSocialMedia**(**

teacher**.**getIdTeacher**(),**

teacherSocialMedia**.**getSocialMedia**()**

**.**getIdSocialMedia**());**

**if(**teacherSocialMediaSaved **==** **null)** **{**

teacherSaved**.**getTeacherSocialMedias**().**add**(**

teacherSocialMedia**);**

**}** **else** **{**

List**<**TeacherSocialMedia**>** teacherSocialMedias **=**

**new** LinkedList**<>();**

teacherSocialMedias**.**addAll**(**

teacherSaved**.**getTeacherSocialMedias**());**

**for** **(**int j **=** 0**;** j **<** teacherSocialMedias**.**size**();** j**++)** **{**

TeacherSocialMedia teacherSocialMedia2 **=**

teacherSocialMedias**.**get**(**j**);**

**if(**teacherSocialMedia**.**getTeacher**().**getIdTeacher**()** **==**

teacherSocialMedia2**.**getTeacher**()**

**.**getIdTeacher**()** **&&**

teacherSocialMedia**.**getSocialMedia**()**

**.**getIdSocialMedia**()** **==**

teacherSocialMedia2**.**getSocialMedia**()**

**.**getIdSocialMedia**())** **{**

teacherSocialMedia2**.**setNickname**(**

teacherSocialMedia**.**getNickname**());**

**}**

teacherSocialMedias**.**set**(**j**,** teacherSocialMedia2**);**

**}**

teacherSaved**.**getTeacherSocialMedias**().**clear**();**

teacherSaved**.**getTeacherSocialMedias**().**addAll**(**

teacherSocialMedias**);**

**}**

**}**

\_teacherService**.**updateTeacher**(**teacherSaved**);**

**return** ResponseEntity**.**ok**(**teacherSaved**);**

**}**