# Pasos creación aplicación Web con modelo spring-jpa-hibernate y controlador con spring MVC

**New Dynamic Web Project y marcar Generate web.xml**

1. Convertir a Maven Project.

En pom.xml: añadir librerías Spring(**core, context, jdbc, web, orm y webMvc)** Y las de jstl, mysql, json simple e hibernate

<**dependencies**>

<!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->

<**dependency**>

<**groupId**>com.googlecode.json-simple</**groupId**>

<**artifactId**>json-simple</**artifactId**>

<**version**>1.1.1</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/javax.servlet/jstl -->

<**dependency**>

<**groupId**>javax.servlet</**groupId**>

<**artifactId**>jstl</**artifactId**>

<**version**>1.2</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<**dependency**>

<**groupId**>mysql</**groupId**>

<**artifactId**>mysql-connector-java</**artifactId**>

<**version**>5.1.47</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.hibernate</**groupId**>

<**artifactId**>hibernate-core</**artifactId**>

<**version**>5.4.3.Final</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-core</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-context</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-jdbc</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-web</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-orm</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-webmvc</**artifactId**>

<**version**>5.1.8.RELEASE</**version**>

</**dependency**>

</**dependencies**>

1. Activar característica JPA en proyecto (properties🡪project facets) para después usar el asistente de creación de entidades y crear dichas entidades (JPA tools🡪create entitites). **Recordar crear los constructores con parámetros**
2. Una vez creadas las entidades, desactivamos la característica JPA y borramos el persistence.xml
3. Crear archivo de configuración de Spring (springConfig.xml: new Spring Bean Configuration File) en carpeta WEB-INF y registrar toda la configuración (cambiar nombre BD, UP, etc)

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:tx=*"http://www.springframework.org/schema/tx"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.3.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.3.xsd"*>

<context:annotation-config/>

<context:component-scan base-package="**modelo.dao**"></context:component-scan>

<bean id=*"data"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"*

value=*"com.mysql.jdbc.Driver"*>

</property>

<property name=*"url"*

value=*"jdbc:mysql://localhost:3306****/bancabd****"*>

</property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"root"*></property>

</bean>

<!-- Creación del adaptador JPA-Hibernate -->

<bean id=*"adp"*

class=*"org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter"*>

<property name=*"databasePlatform"*

value=*"org.hibernate.dialect.MySQLDialect"*>

</property>

</bean>

<!-- Configuración JPA -->

<bean id=*"factory"*

class=*"org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"*>

<property name=*"persistenceUnitName"* value=*"bancaPU"*></property>

<property name=*"dataSource"* ref=*"data"*></property>

<property name=*"jpaVendorAdapter"* ref=*"adp"*></property>

<property name=*"packagesToScan"* value=*"entidades"*></property>

</bean>

<!-- Creación objeto Transaction Manager -->

<bean id=*"tm"*

class=*"org.springframework.orm.jpa.JpaTransactionManager"*>

<property name=*"entityManagerFactory"* ref=*"factory"*></property>

</bean>

<!-- Permitir utilizar el transaction manager de forma declarativa (Anotaciones) -->

<tx:annotation-driven transaction-manager=*"tm"*/>

</beans>

1. Interfaces e implementación de las clases del modelo, utilizando JPA y empleando las anotaciones Spring para inyección de EntityManager y aplicar transaccionalidad
2. Configurar web.xml: página inicio e incluir el registro del DispatcherServlet (tiene el listener que arranca Spring) con la referencia a los archivos de configuración mvcConfig y springConfig

<**servlet**>

<**servlet-name**>dispatcher</**servlet-name**>

<**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>

<**init-param**>

<**param-name**>contextConfigLocation</**param-name**>

<**param-value**>/WEB-INF/mvcConfig.xml,/WEB-INF/springConfig.xml</**param-value**>

</**init-param**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>dispatcher</**servlet-name**>

<**url-pattern**>/</**url-pattern**>

</**servlet-mapping**>

1. Crear mvcConfig en WEB-INF con Spring Configuration File (bean,context y mvc) y registrar la creación de los controladores y reglas de navegación dinámicas y estáticas

<!-- creación de controladores -->

<**mvc:annotation-driven**/>

<**context:component-scan** base-package=*"controlador"*></**context:component-scan**>

<!-- reglas de navegación -->

<!--registro de objeto encargado de la navegación-->

<**bean** id=*"resolver"*

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<**property** name=*"prefix"* value=*"/"*></**property**>

<**property** name=*"suffix"* value=*".jsp"*></**property**>

</**bean**>

<!-- navegaciones estáticas -->

<**mvc:view-controller** path=*"/toMenu"* view-name=*"menu"*/>

<**mvc:view-controller** path=*"/toAltaCurso"* view-name=*"altacurso"*/>

Poner aquí para acceso a los archivos de estilos de la página?

<mvc:resources mapping=”/asset/\*\*” location=”/asset/”><mvc:resources>

1. Implementar los controladores como clases estándares (spring MVC xxxAction.java), anotadas con @Controller. Los métodos controladores se anotan con @PostMapping o @GetMapping en función del método HTTP que provoca su ejecución
   1. **Get pq viene de un enlace**
   2. **Post si vienen de formulario**
   3. **@RequestMapping que engloba post y get,**

Si se usa etiquetas MVC form:

*@RequestMapping*(value="**doPrepararAlta"**)

public String prepararAlta(HttpServletRequest request, **Model mod**) {

//Crear instancia antes de pedir los datos, para que Spring pueda volcar en ella, los parámetros enviados desde la petición post

Alumno al=new Alumno();

mod.addAttribute("alumno", al);

//recupera la lista de cursos y la guarda en un

//atributo de petición para que pueda estar accesible

request.setAttribute("cursos", daoCursos.getCursos());

return "altaalumno";

}

*@PostMapping*(value="doAltaAlumno")

public String altaAlumno(*@ModelAttribute*("alumno") Alumno alumno) {

//comprobar que el alumno no existe

if(daoAlumnos.buscarAlumno(alumno.getDni())==null) {

daoAlumnos.altaAlumno(alumno);

return "menu";

}else {

//no se escribe mensaje de error, eso es cosa del cliente

//le redirijimos al controlador de acción para que vuelva a preparar la página para pintarla

//para redirigir a un controlador de acción

return "forward:/doPrepararAlta";

}

}

1. Implementación de las vistas

* Jstl (no necesario importar beans)
* Usar etiquetas SpringMVC para formularios: Si formulario de muchos campos

[<%@**taglib uri=*"http://www.springframework.org/tags/form" prefix=”f” %***](mailto:%25@taglib%20uri=%22http://www.springframework.org/tags/form%22%20prefix=f%20%25)***>***

<**f:form** action="doAltaAlumno" method="post" modelAttribute="alumno">

* Carpeta asset

**Si BD entidades relacionadas:**

* Poner filtro para impedir que hibernate cierre la sesión al localziar la entidad principal y así hibernate pueda ejecutar la siguiente línea pq hace una carga lazy de las entidades relacionadas. Sino, daría error: **Error LazyInitialization.** En web.xml

**<filter>**

**<filter-name>OpenEntityManagerInViewFilter</filter-name>**

**<filter-class>org.springframework.orm.jpa.support.OpenEntityManagerInViewFilter</filter-class>**

**</filter>**

**<filter-mapping>**

**<filter-name>OpenEntityManagerInViewFilter</filter-name>**

**<url-pattern>/\*</url-pattern>**

**</filter-mapping>**

* Como el filtro se ejecuta antes que el servlet, no se activa spring, con lo que hay que volver a activar spring con un listener en web.xml

**<context-param>**

**<param-name>contextConfigLocation</param-name>**

**<param-value>/WEB-INF/springConfig.xml</param-value>**

**</context-param>**

**<listener>**

**<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>**

**</listener>**

* Y en web.xml se quita del servlet-dispatcher la línea **:** /WEB-INF/springConfig.xml

1. Blindaje errores: comprobar que no exista en la BD algo con el mismo campo, etc. Como lo del merge: