Sesión de Aprendizaje N° 10-B

Desarrollo Web BackEnd

Stack: Java - SPRING

JSP / Jakarta - SPRING

Spring Data / Hibernate / JPA

1. Servidor Web: Apache Tomcat

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation.

Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por Sun Microsystems).
Tomcat es desarrollado y actualizado por miembros de la *Apache Software Foundation* y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software License.

Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets conocido como ***Catalina***.

Las versiones más recientes son las 9.x, que implementan las especificaciones de Servlet 4.0 y de JSP 2.3.

La Estructura de Directorios de Tomcat

La distribución de Tomcat está dividida en los siguientes directorios:

- **bin:** ejecutables y scripts para arrancar y parar Tomcat.
- **common:** clases y librerías compartidas entre Tomcat y las aplicaciones web. Las clases se deben colocar en common/classes, mientras que las librerías en formato JAR se deben poner en common/lib.
- **conf:** ficheros de configuración.
- **logs:** directorio donde se guardan por defecto los logs.
- **server:** las clases que componen Tomcat.
- **shared:** clases compartidas por todas las aplicaciones web.
- **webapps:** directorio usado por defecto como raíz donde se colocan todas las aplicaciones web.
- **work y temp:** directorios para almacenar información temporal

Archivo de Configuración server.xml

Es el núcleo de la configuración de Apache Tomcat. Define la estructura y los parámetros clave del servidor. Aquí tienes algunos elementos importantes dentro de este archivo:

- <Server>: Es el elemento raíz que contiene toda la configuración del servidor.
- <Service>: Agrupa conectores y motores de procesamiento.
- <Connector>: Define cómo Tomcat acepta conexiones, incluyendo el puerto, el protocolo y el número de hilos.
- <Engine>: Es el motor de procesamiento de solicitudes dentro de un servicio.
- <Host>: Representa un host virtual dentro del motor.
- <Context>: Define configuraciones específicas para aplicaciones web.

Algunos parámetros clave dentro de server.xml incluyen:

- port: Especifica el puerto en el que Tomcat escucha (por ejemplo, `port="8080"`).
- maxThreads: Número máximo de hilos para manejar solicitudes simultáneas.
- connectionTimeout: Tiempo de espera para conexiones inactivas.
- redirectPort: Puerto al que se redirigen solicitudes HTTPS.

2. Gestión de Dependencias con Maven

Un **gestor de dependencias** es una herramienta que administra automáticamente las bibliotecas y módulos necesarios en un proyecto de software. Su función principal es descargar, instalar y actualizar estos recursos, asegurando compatibilidad y evitando conflictos de versiones.
Hay dos gestores en Java: *Maven* y *Gradle*.

Introducción a Gradle

Gradle también es una herramienta de software libre lanzada en 2007 ***pero sirve para gestionar proyectos Java, Groovy o Scala***, aunque ya soporta otros lenguajes.

Herramienta Maven

Maven es ***una herramienta de software libre para gestionar proyectos Java*** lanzada en 2002. Se basa ***en archivos xml***. La filosofía de Maven es la estandarización de las estructuras de aplicaciones y de su Configuración para la producción de software.

Url: https://maven.apache.org/

Con Maven se puede:

- Gestionar las dependencias del proyecto, para descargar e instalar módulos, paquetes y herramientas que sean necesarios para el mismo.
- Compilar el código fuente de la aplicación de manera automática.
- Empaquetar el código en archivos .jar o .zip.
- Instalar los paquetes en un repositorio (local, remoto)
- Generar documentación a partir del código fuente.
- Gestionar las fases del ciclo de vida de las ***build***: validación, generación de código fuente, procesamiento, generación de recursos, compilación, ejecución de test ...

En Maven se puede usar con multitud de elementos denominados ***plugins*** construidos por la comunidad de desarrolladores. Ejemplo: hay plugins que se pueden usar para funciones como desplegar deploy el proyecto en un servidor

Repositorio Maven

Maven también, proporciona un repositorio que contiene librerías de código, plugins y utilidades reutilizables. https://mvnrepository.com/
El repositorio de Maven (o repositorio central) tiene una estructura que permite descargar y almacenar los archivos JAR y WAR en versiones distintas.

Cuando se ejecuta cualquiera de los comandos maven, por ejemplo, si ejecutamos:

```
> mvn install
```

Maven irá verificando todas las fases del ciclo de vida desde la primera hasta la del comando, ejecutando solo aquellas que no se hayan ejecutado previamente.

¿Qué es el archivo pom.xml?

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).

Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc... ()

Archivo pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>web2</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>20</maven.compiler.source>
        <maven.compiler.target>20</maven.compiler.target>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    </properties>

</project>
```

Integración con IDE

Maven sustituye el entorno integrado de desarrollo (IDE por sus siglas en inglés), por tanto la integración con diferentes IDEs es muy importante. Existen plugins de Maven para crear archivos de configuración del IDE a partir de los POMs. Actualmente se soportan:

- Eclipse
- Netbeans
- IntelliJ IDEA
- JDeveloper 11G (11.1.1.3)

Comandos Maven

Utilice el siguiente comando de Maven para compilar e implementar la aplicación web de plantilla en el servidor Tomcat. Una vez implementado, puede acceder a él a través de la url http://localhost:8080/my-webapp/:

Desplegar un proyecto:

```
$ mvn clean install tomcat7:deploy
```

Redesplegar un proyecto:

```
$ mvn clean install tomcat7:redeploy
```

Quitar un proyecto

```
$ mvn clean install tomcat7:undeploy
```

3. ¿Qué es Jakarta Server Page?

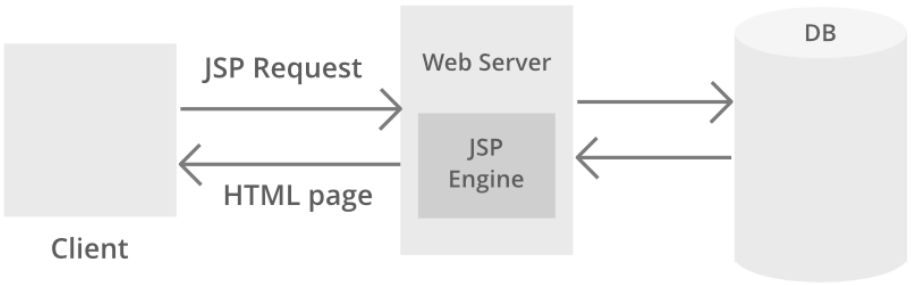
El JSP *es Java Server Pages es un lenguaje script del lado del servidor que permite la creación de un método dinámico independiente de la plataforma para construir aplicaciones Web. JSP tiene acceso a toda la familia de las API de Java, incluyendo la API JDBC para acceder a bases de datos.*

Nota: La fundación eclipse ha impulsado el nuevo JSP Jakarta Server Page.

Información de Jakarta y JSP

Link de Jakarta: https://jakarta.ee/

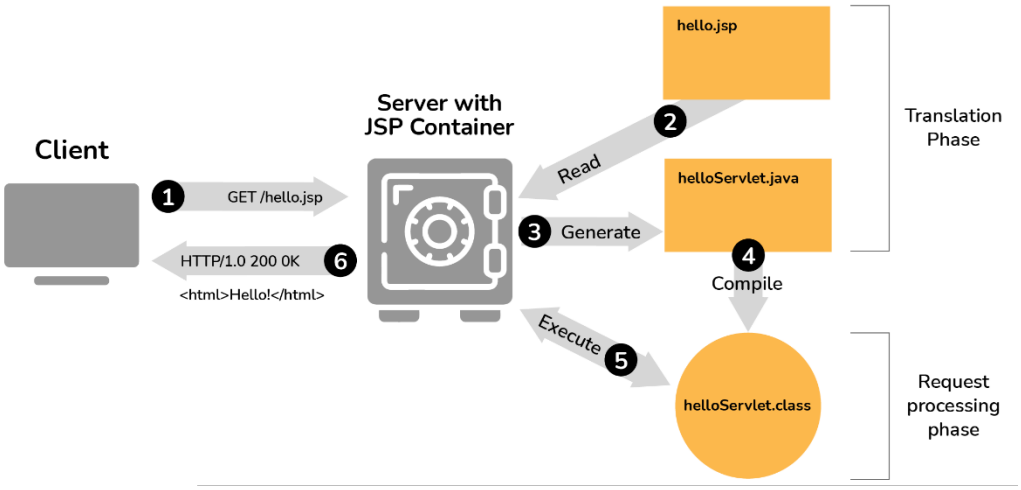
Jakarta Server Pages: https://jakarta.ee/specifications/pages/3.0/



Solo requiere de archivos JSP y el servidor web necesita un *motor JSP*, es decir, un contenedor para procesar páginas de JSP. El *contenedor JSP* se encarga de interceptar las solicitudes de páginas del JSP para proporcionar el entorno de tiempo de ejecución y otros servicios que un JSP necesita.

Procesamiento del JSP

Los siguientes pasos explican cómo el servidor web crea la página web usando JSP.



4. Sintaxis JSP

Los elementos de JSP se han descrito a continuación.

Scriptlet

Un scriptlet puede contener cualquier número de declaraciones de JAVA, declaraciones variables, de métodos, o expresiones que sean válidas en el lenguaje de scripting de la página.

Sintaxis

```
<% code fragment %>
```

Puedes escribir el equivalente XML de la sintaxis anterior de la siguiente manera.

```
<jsp:scriptlet>
   code fragment
</jsp:scriptlet>
```

Ejemplo:

```
<%
        out.println("Hola desde JSP");
%>
```

Declaraciones del JSP

Una declaración declara una o más variables o métodos que puede utilizar en el código Java más adelante en el archivo JSP. Debe declarar la variable o el método antes de usarlo en el archivo JSP.

Sintaxis

```
<%! declaration; [ declaration; ]+ ... %>
```

Puedes escribir el equivalente XML de la sintaxis anterior de la siguiente manera.

```
<jsp:declaration>
   code fragment
</jsp:declaration>
```

Ejemplo:

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP Expression

Un elemento de expresión de JSP contiene una expresión del lenguaje scripting que se evalúa, se convierte en elemento html, e insertada donde la expresión aparece en el archivo JSP.

Sintaxis

```
<%= expression %>
```

Puedes escribir el equivalente XML de la sintaxis anterior de la siguiente manera.

```
<jsp:expression>
   expression
</jsp:expression>
```

Ejemplo:

```
<html>
   <head><title>A Comment Test</title></head>

   <body>
      <p>Today's date:
<%= (new java.util.Date()).toLocaleString()%>
      </p>
   </body>
</html>
```

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

<%-- This is JSP comment --%>

5. Directivas y Acciones del JSP

Directivas

Una directiva del JSP afecta a la estructura general de la clase de servicio. Generalmente tiene la siguiente forma.

```
<%@ directive attribute="value" %>
```

Hay tres tipos de etiqueta directiva.

<%@ page ... %>

Define los atributos dependientes de la página, como el lenguaje de scripting, la página de error y los requisitos de amortiguación.

<%@ include ... %>

Incluye un archivo durante la fase de traducción.

<%@ taglib ... %>

Declara una biblioteca de etiquetas, que contiene acciones personalizadas, usadas en la página

Acciones

Las acciones de JSP utilizan construcciones en la sintaxis XML para controlar el comportamiento del motor de servicio. Puede insertar dinámicamente un archivo, reutilizar los componentes de JavaBeans, reenlacar al usuario en otra página o generar HTML para el plugin Java.

Sólo hay una sintaxis para el elemento Acción, ya que se ajusta al estándar XML.

```
<jsp:action_name attribute="value" />
```

6. Objetos implícitos de JSP

JSP soporta nueve variables definidas automáticamente, que también se llaman objetos implícitos. Estas variables son .

No.	Objeto & Descripción
1	**Request** Este es el objeto **HttpServletRequest** asociado a la solicitud.
2	**Response** Este es el objeto **HttpServletResponse** asociado con la respuesta al cliente.
3	**Out** Este es el objeto **PrintWriter** utilizado para enviar salida al cliente.
4	**Sesión** Este es el objeto **HttpSession** asociado a la solicitud.
5	**Aplicación** Este es el objeto **ServletContext** asociado al contexto de la aplicación.
6	**Config** Este es el objeto **ServletConfig** asociado a la página.
7	**pageContext** Esto encapsula el uso de características específicas del servidor como **JspWriters** de mayor rendimiento.
8	**Page** Esto es simplemente sinónimo de **esto**, y se utiliza para llamar a los métodos definidos por la clase de servicio traducido.
9	**Exception** El objeto **Excepción** permite que los datos de excepción sean accedidos por JSP designado.

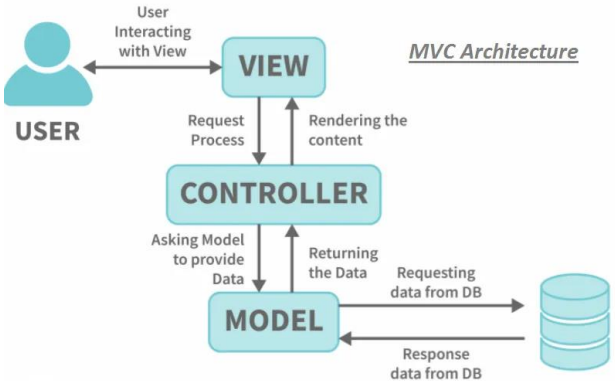
Ejemplo

```
<%! int day = 3; %>
<html>
   <head><title>IF...ELSE Example</title></head>

   <body>
      <% if (day == 1 || day == 7) { %>
         <p> Today is weekend</p>
      <% } else { %>
         <p> Today is not weekend</p>
      <% } %>
   </body>
</html>
```

7. MVC de JSP

MVC – Modelo Vista Controlador

Es una arquitectura que separa el desarrollo en capas: presentación (Vista), lógica (Controlador) y datos (Modelo). En esto, el flujo comienza desde la capa de vista, donde la solicitud se eleva y procesa en la capa del controlador. Esto se envía a la capa de modelo para insertar datos y recuperar el mensaje de éxito o fracaso.



Ejemplo:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<form action="ControllerServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br> <br>
<input type="submit" value="login">
</form>
```

8. Spring Framework

Spring es un *robusto framework* para el *desarrollo de aplicaciones empresariales* en el *lenguaje de programacion Java*.
Fue creado por *programadores para programadores*, con la finalidad de *estandarizar el trabajo, resolver, agilizar y manejar los problemas y complejidades* que van apareciendo en el mundo de la programación.

Spring framework es muy extenso y su comunidad crece día a día para ayudar al desarrollo de aplicaciones web.
Link de Spring Framework: https://spring.io/
Documentación Spring Framework: https://docs.spring.io/spring-framework/docs/current/reference/html/

Terminología Spring

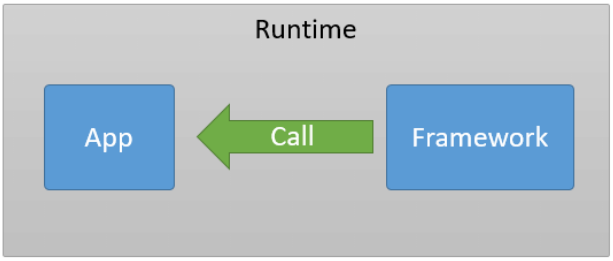
Bean
El término ***"bean" (o componente)*** se utiliza para referirse a cualquier componente manejado por Spring
Los "bean" ***son clases en forma de JavaBeans***

- Sin args en el constructor.
- Métodos getter y setter para los atributos

Atributos de los "beans" pueden ser valores simples o probablemente referencias a otros "beans"

Inversion de Control (IOC)

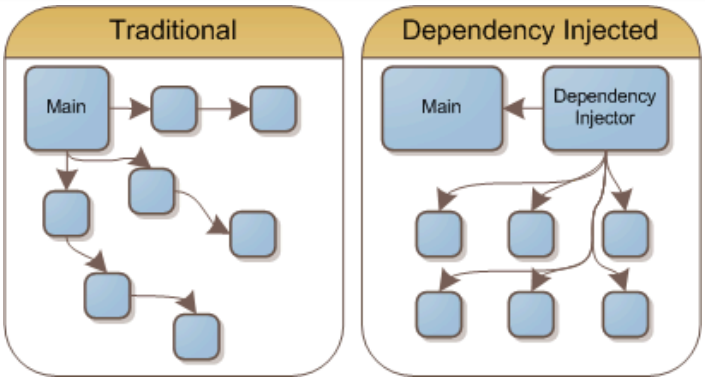
Inversión de control se usa para referirse al ***cambio en el flujo de ejecución y vida de los objetos con respecto a la programación tradicional***, básicamente es de invertir la forma en que se controla la aplicación, lo qué antes ***dependía del programador***, por ejemplo: el orden en que se llaman los métodos, ahora depende ***completamente del framework***, todo esto con la finalidad de crear aplicaciones más complejas y automáticas.

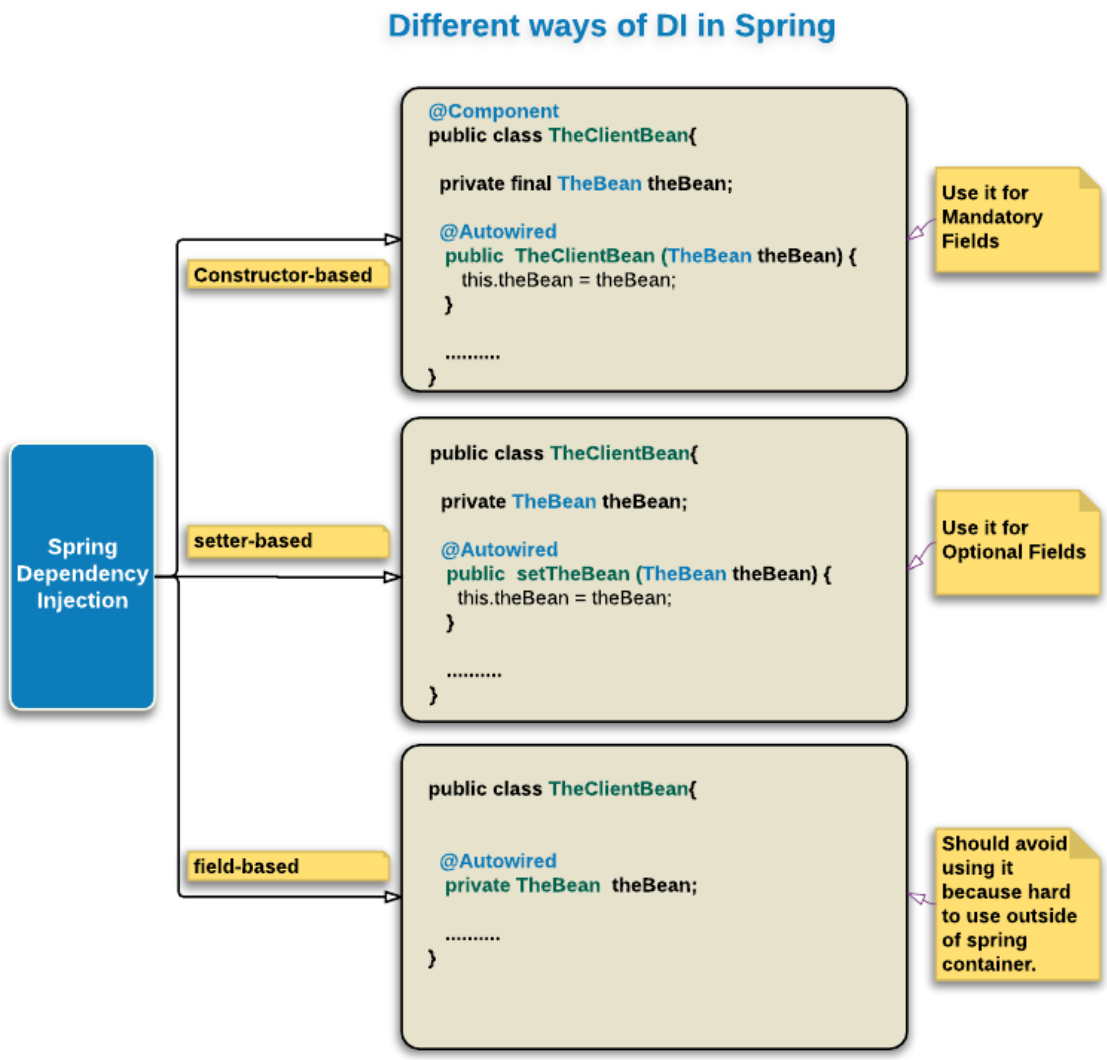


Principio Hollywood

Inyección de dependencia (DI) en Spring

La inyección de dependencia, es un tipo de inversion de control, donde *el manejo de las propiedades de un objeto, son inyectadas a través de un constructor, un setter, un servicio*, etc. Creando de esta manera un control diferente para un comportamiento más conveniente en nuestra aplicación.





Sabiendo esto, una de las caracteristicas principales de spring framework es el Spring container.

El **Spring Container** es un **IoC Container** (*Contenedor de Inversion de control*), es parte del framework spring, que *se encargan de realizar inyección de dependencia* sin la necesidad de realizar la definición en java (lenguaje POO) a través de archivos de metadata, por ejemplo un xml.

Este **IOC container** se encargará de *crear los objetos, relacionarlos, configurarlos y manejar su ciclo de vida completo*, desde la creación de estos hasta su destrucción, es por esto que se usa el término *Inversion of control* en spring.

Ejemplo de Clases en Spring

Ejemplo de como funciona el *Spring container*.

Imaginemos que tenemos un *proyecto* llamado *Universidad*, con una *clase Estudiante.java*

Una POO convencional sería lo siguiente*:*

```java
package com.universidad;

public class EstudianteBean{
   private String nombre;
   private String apellido;
   private String dni;

   public String getNombre(){
   return nombre;
   }

   public void setNombre(String nombre){
   this.nombre = nombre;
   }

   public String getApellido(){
   return apellido;
   }

   public void setApellido(String apellido){
   this.apellido = apellido;
   }

   public String getDni(){
   return dni;
   }

  public void setDni(String dni){
  this.dni = dni;
  }

  public void mostrarEstudianteActual(){
  System.out.println(nombre+" "+apellido+", es el estudiante actual");

  }

}
```

C

Nuestro proyecto **Universidad** *utilizando Spring framework*, la configuración se realiza mediante un *archivo xml* que se encarga del control de ciertos objetos (Spring IOC Container). Este archivo, llamado *springconfig.xml* debería ser así.

```
<!DOCTYPE beans PUBLIC "-//PRING//DTD// BEAN 2.0//ES"
"http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
    <bean id="estudiante" class="com.universidad.EstudianteBean"
>       <property name="nombre" value="Juan">
        <property name="apellido" value="Perez">
        <property name="dni" value="95.445.123">
    </bean>
</beans>
```

La etiqueta *"bean"* tiene un *atributo* **class** con la *dirección en mi proyecto* y el nombre de la *clase EstudianteBean.java* que nos representa un archivo, y el "id" tiene un nombre para identificarlo. La etiqueta *"property"* apunta a las propiedades de la clase y "value" con un valor directo. El Spring Container puede acceder y modificar objetos de la clase EstudianteBean a través de sus metodos get y set.

Spring permite (a través de archivos de configuración XML), crear un catálogo para cada beans, ejecutarlo y manipularlos desde otras clases sin necesidad de instanciarlos una y otra vez. Utiliza el patrón Factory para generar instancias.

¿Cuándo Spring empieza IoC?

Spring, implementa el control sobre los objetos en el proyecto Universidad de la siguiente manera:

```
package com.universidad;

import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.Resource;

public class UniversidadLogic{

  public static void main(String[] args){
      Resource recurso = new ClassPathResource("springconfig.xml");
      BeanFactory factory = new XmlBeanFactory(recurso);

      EstudianteBean estudianteBean = (EstudianteBean)factory.getBean("estudiante"
);
      estudianteBean.mostrarEstudianteActual();
   }
}
```

Las librerías de spring para la creación de objetos utiliza BeanFactory se encargan del mapeo y manejo de los objetos, mientras que las relacionadas con Resource y ClassPathResource se encargan de apuntar al archivo xml que tiene el catalogo de Beans.

En el ejemplo, puede parecer muy tedioso el manejo de objetos con Spring, pero las instancias de Resource y BeanFactory son procesos que se jecutan una vez en grandes proyectos que manejan muchas clases y objetos esto permitirá un manejo eficaz de nuestras aplicaciones.

9. SPRING-BOOT

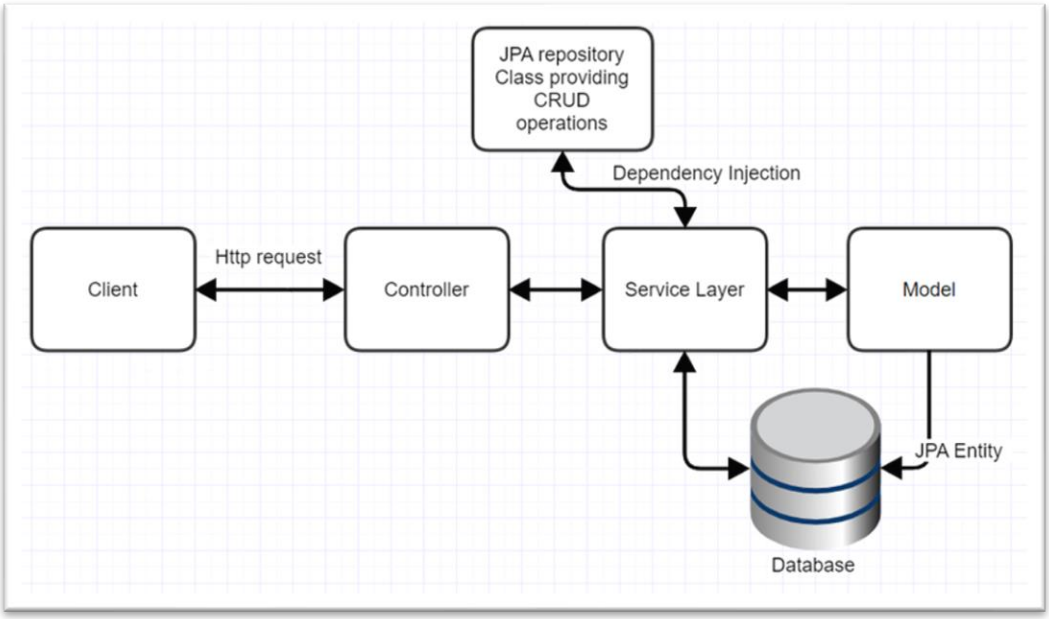
Spring Boot es una *herramienta que permite crear proyecto con Spring Framework*, *eliminando la complejidad de las configuraciones*, *instalación de dependencias* además *integrando un servidor de pruebas* para desplegar la aplicación en desarrollo.

Documentación:
https://spring.io/projects/spring-boot

Entre las características de la herramienta Spring Boot se encuentran:

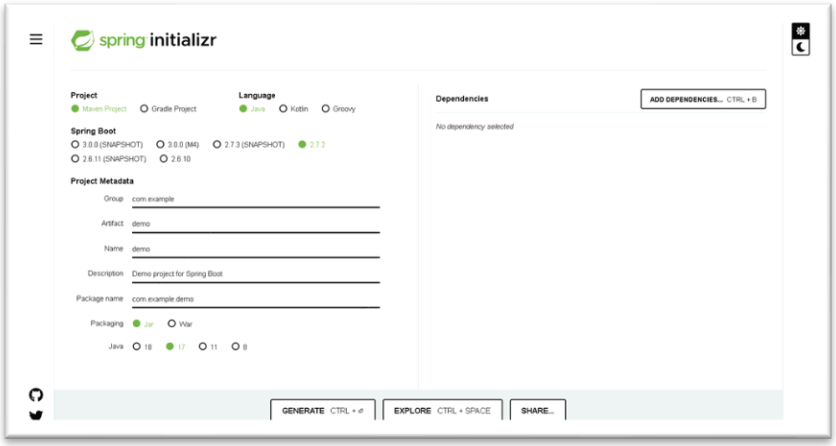
- Permite hacer una *configuración automática* para que Spring Framework funcione.
- Ofrece *métricas*.
- Brinda *comprobación de Helth (vida)* de una aplicación con *Spring Framework*.
- Ofrece ***un servidor incorporado*** para evitar la complejidad en el despliegue (deployment) de una aplicación Spring.

Arquitectura de una Aplicación con Spring Framework



Página de Inicialización de Spring-Boot

https://start.spring.io/



Segunda Unidad: Desarrollo JSP JAVA
GUÍA DE LABORATORIO N° 09:
INSTALACIÓN DE JDK JAKARTA Y TOMCAT

Docente: Jaime Suasnabar Terrel

Fecha: …….. **Duración:** 90 minutos

Instrucciones: Instalar JDK Jakarta, Intelli J Idea y Tomcat.

OBJETIVOS
- Conocer el funcionamiento de Java y Open JDK
- Instalar Open JDK
- Instalar Intelli J Idea.
- Instalar Servidor Web Tomcat

PROCEDIMIENTO

EJERCICIO 01. INSTALACIÓN DE JDK OPEN SOURCE Y SERVIDOR WEB APACHE TOMCAT

1. Instalación de JDK para Jakarta

Descargar los binarios de la página oficial de OpenJDK:

- Página principal: https://openjdk.java.net/
- Binario: https://jdk.java.net/19/

Se descomprime el archivo zip descargado.

Clic derecho sobre el archivo y Extraer todo.

- Seleccionamos la ubicación donde queremos que quede los binarios del JDK. En mi caso voy a utilizar: c:\java.
- Se ha creado una carpeta llamada jdk-19.0.2. Debemos recordar esta ruta porque la configuraremos en las variables de entorno.

Creación de las variables de entorno.

- Se etc debe crear la variable de entorno JAVA_HOME y se debe actualizar la variable de entorno PATH.
- La primera se utilizará para que java sepa dónde se encuentra la instalación del JDK y la segunda es para poder ejecutar los comandos de java (como javac, java,) desde cualquier lugar.

Probar Java y JDK.

- Ahora abre una terminal. Puedes comprobar la versión de Java y del compilador de Java ejecutando:

```
>java --version
```

- Y:

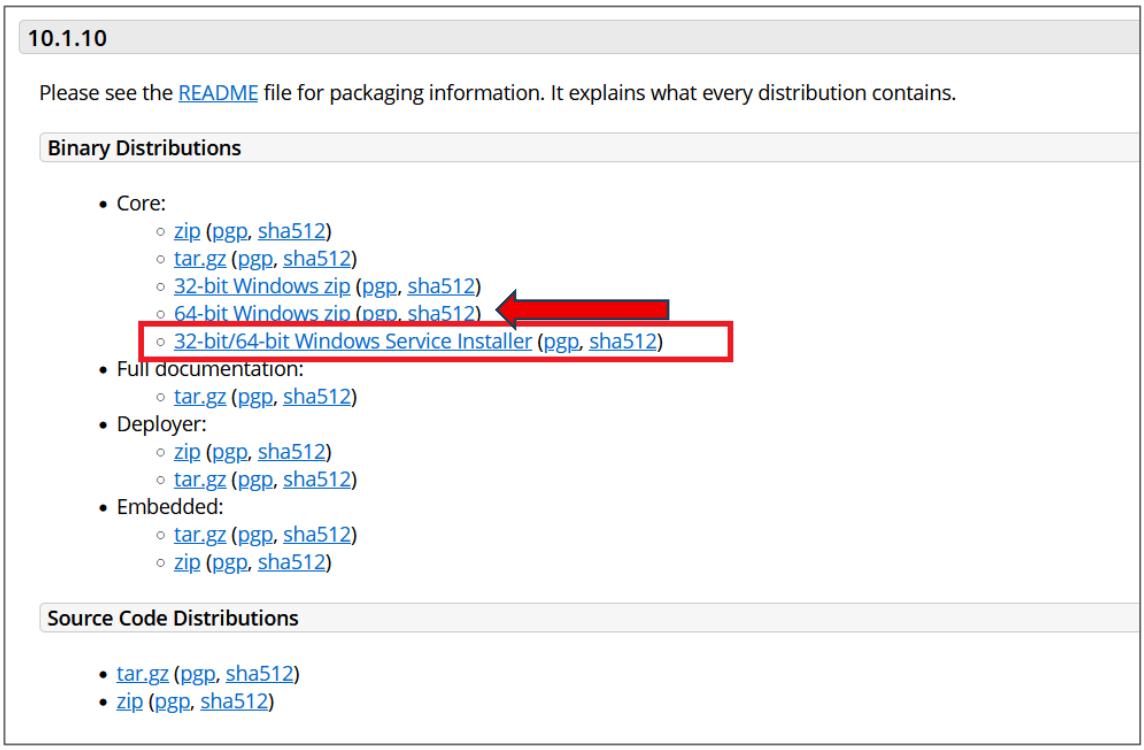
```
javac --version
```

2. TOMCAT – Instalación

Ingresar a la web: http://tomcat.apache.org/

Descargar el instalador estable más reciente

- Haz clic en Tomcat 10 en la barra lateral izquierda. Puedes encontrar esta opción debajo del título "Descargas" (Downloads) en un menú de navegación del lado izquierdo.
- Descarga 32-bit/64-bit Windows Service Installer debajo de "Core". Puedes encontrar esta opción en la sección "Distribuciones binarias" (Binary Distributions) de la parte inferior.



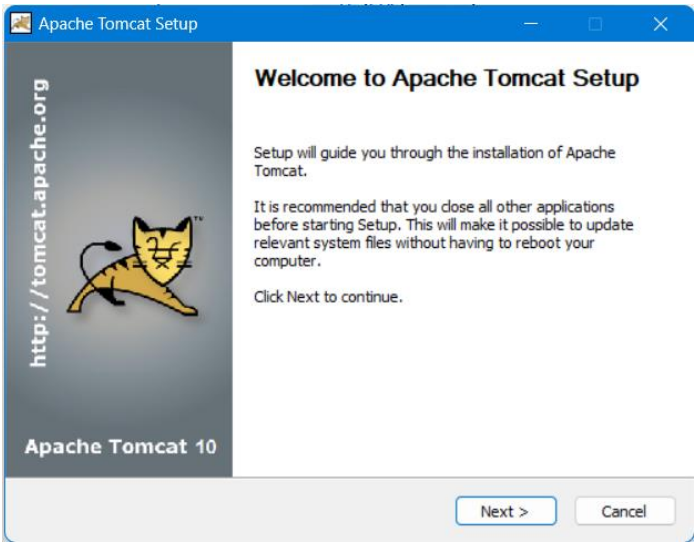
- Si se te pide, selecciona una ubicación para guardar el archivo de instalación.

Descarga el instalador de TOMCAT 64bit Windows.zip

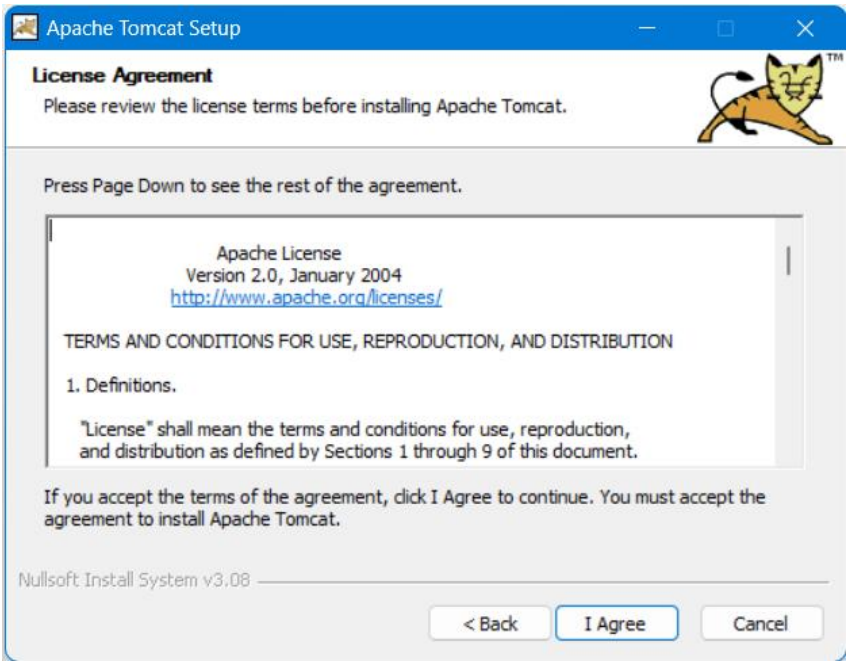
1. Descargar . **APACHE TOMCAT 64bit Windows.zip** descomprimir en un directorio del computador
2. Inicializar **APACHE TOMCAT con start.bat**

Ejecutar el instalador de TOMCAT SERVICE INSTALLER

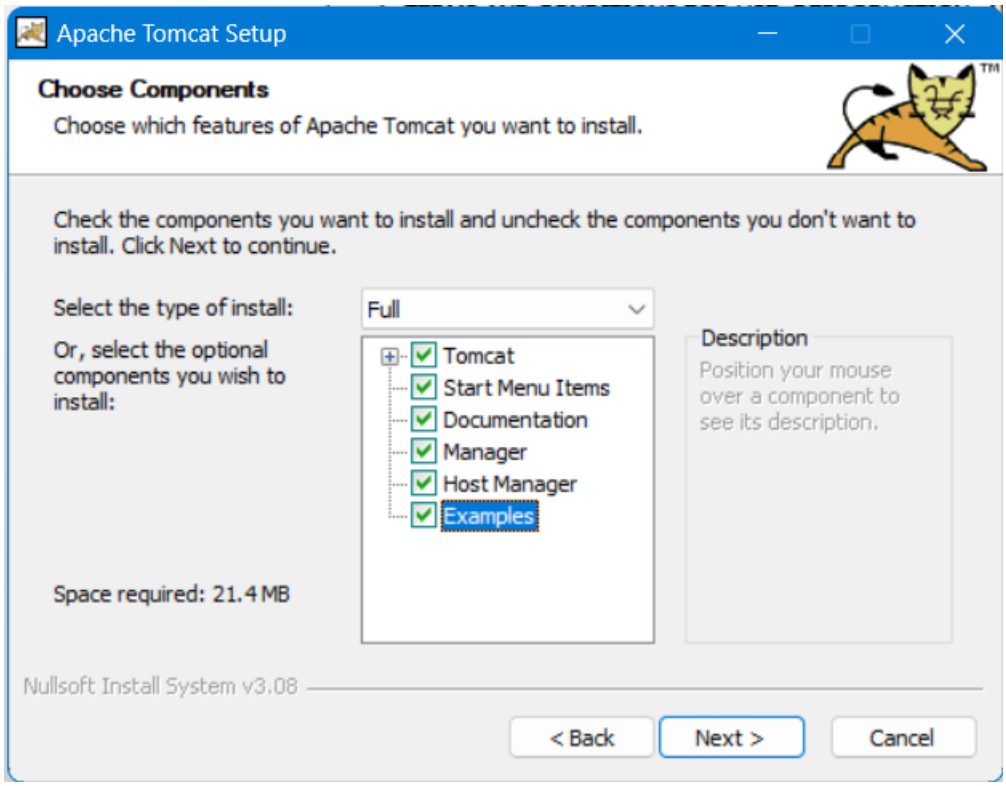
3. Inicia el archivo de instalación en la computadora. Busca el instalador en la carpeta "Descargas", y hazle doble clic para iniciar el asistente de instalación. Y Click en Siguiente.



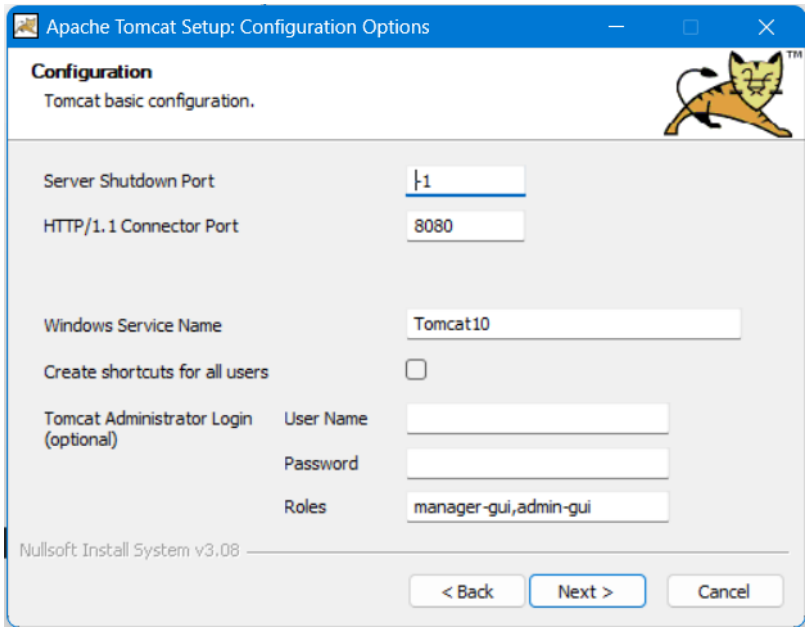
4. Abre el Acuerdo de licencia (License Agreement) en una nueva página. Click en I Agree acepto los acuerdos de licencia.



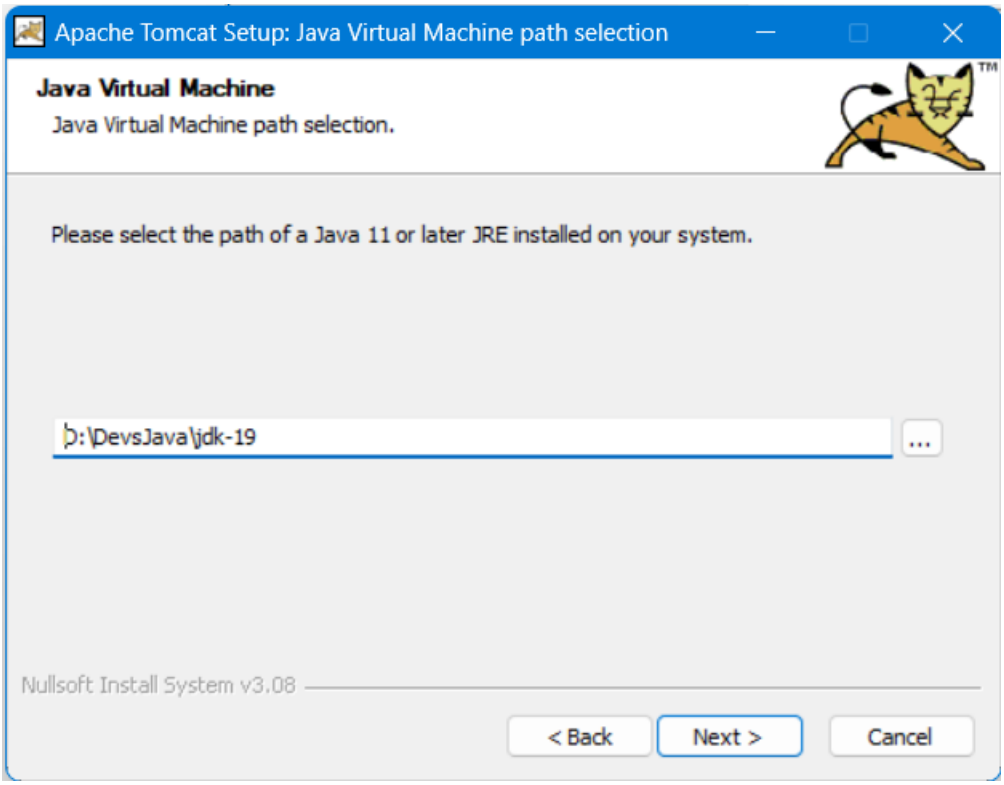
5. En la siguiente ventana. Selecciona el tipo de instalación" (Select the type of install) y selecciona Completo (**Full**) aquí para instalar todos los componentes de Tomcat, incluyendo la documentación y los accesos directos de la aplicación



6. Haz clic en Siguiente en configuración. A menos que vayas a personalizar los puertos, haz clic en Siguiente para continuar.



7. Especificar la ubicación que Java SE tendrá en la computadora en la siguiente página, elegir el botón … si desea examinar.



EJERCICIO 02. CREACIÓN DE PROYECTO WEB JSP

1. Crear una Aplicación Web JSP que muestre la fecha actual

Código

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <title>Ejemplo JSP</title>
</head>
<body>
    <h1>¡Hola desde JSP!</h1>
    <p>La fecha y hora actual es: <%= new java.util.Date() %></p>
</body>
</html>
```

2. Crear una Aplicación Web JSP que utilice los métodos de formulario get y post

```
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Cálculo del factorial</title>
</head>
<body>
    <h1>Cálculo del factorial</h1>

    <!-- Formulario que solicita al usuario el número a usar en el cálculo -->
    <form action="factorial.jsp" method="get">
        <p>Número: <input type="text" name="numero">
        <input type="submit" value="Calcular"></p>
    </form>
```

```
    <%
    // el objeto request se obtiene el valor pasado por el formulario
        String numeroGet = request.getParameter("numero");
        if(numeroGet!=null) {
            int numero=0;
            double factorial=1;
            boolean error = false;
            try {
                numero = Integer.valueOf(numeroGet);
                if(numero<1) {
                    error = true;
                } else {
                    for(int i=numero; i>1; i--) {
                        factorial *= i;
                    }
                }
            } catch(NumberFormatException e) {
                error = true;
            }
            if(error) {
                out.println("<p>Debe indicar un número entero mayor que
0</p>");
            } else {
                // Mostrar el resultado en la página usando el objeto out
                out.println("<p>Resultado: "+numero + "! = " +
factorial+"</p>");
            }
        }

        // Uso del objeto session para contar las veces que se ejecuta la
aplicación
        Integer contador = (Integer)session.getAttribute("contadorVisitas");
        if(contador!=null) {
            contador = Integer.valueOf(contador);
        } else {
            contador = 0;
        }
        if(contador!=0) {
            out.println("<p>Ejecuciones de la aplicación en esta sesión:
"+contador+ "</p>");
        }
        contador++;
        session.setAttribute("contadorVisitas", contador);
    %>
</body>
</html>
```

3. Desarrollar un API con Java y Spring

Segunda Unidad: Desarrollo API JAVA SPRING
GUÍA DE LABORATORIO N° 10:
CREACIÓN DEL MODELO MVC - JAVA

Docente: Jaime Suasnabar Terrel

Fecha: …….. **Duración:** 90 minutos

Instrucciones: Contar con JDK Jakarta, Intelli J Idea y Tomcat.

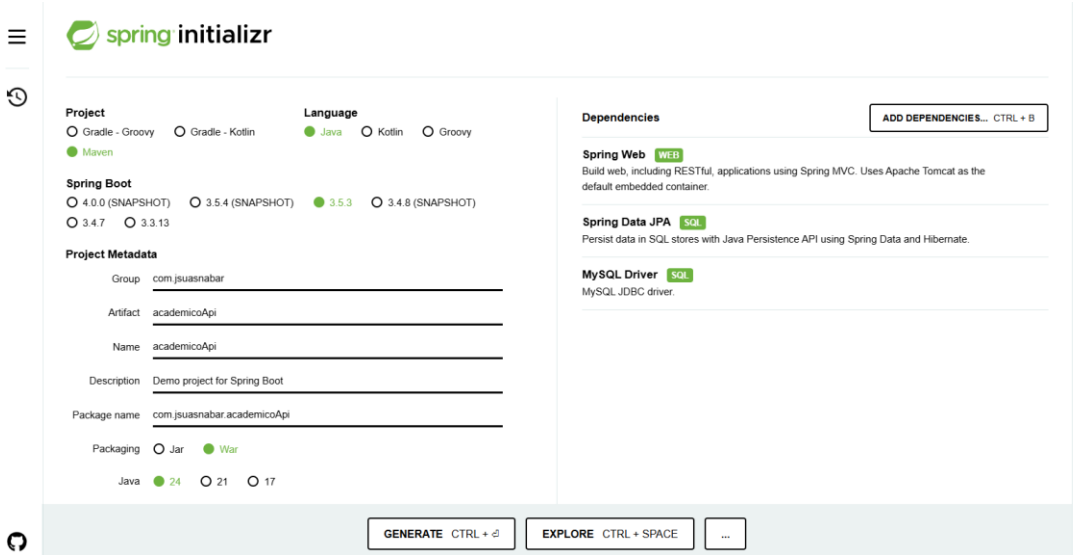
METAS

- Creación del Proyecto SPRING con SpringBoot
- Abrir el Proyecto con IntelliJ Idea y Configurar Ejecución de SpringBoot
- Configurar properties para MySQL y Crear los paquetes controllers, models, repositories y services
- Creación del modelo Estudiante
- Creación del EstudianteRepository
- Creación del EstudianteController
- Comprobar CRUD API con Posman

PROCEDIMIENTO

EJERCICIO 01. CREACIÓN DEL PROYECTO CON SPRING-BOOT

1. En https://start.spring.io/ configurar un proyecto Maven con la siguiente estructura



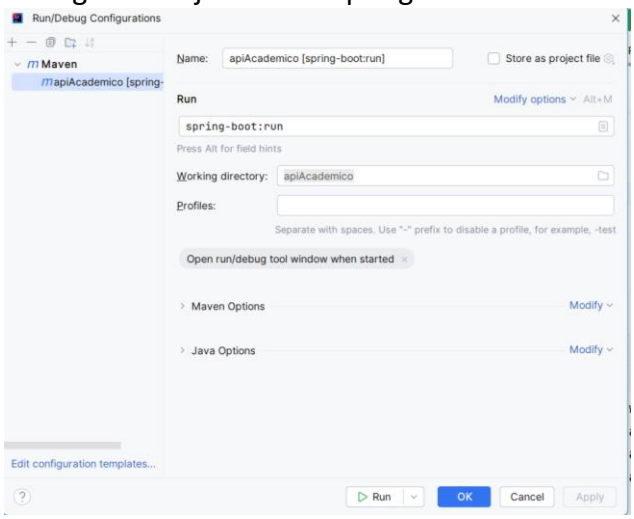
Descargar el proyecto con el botón GENERATE en un archivo *.zip

Descargar

Se descomprime el archivo zip descargado.

EJERCICIO 02. ABRIR EL PROYECTO CON INTELLIJ IDEA Y CONFIGURAR EJECUCIÓN DE SPRINGBOOT

1. Abrir en IntelliJ Idea el Proyecto Maven.
2. Verificar la instalación de librerías
3. Configurar la ejecicón de spring-boot

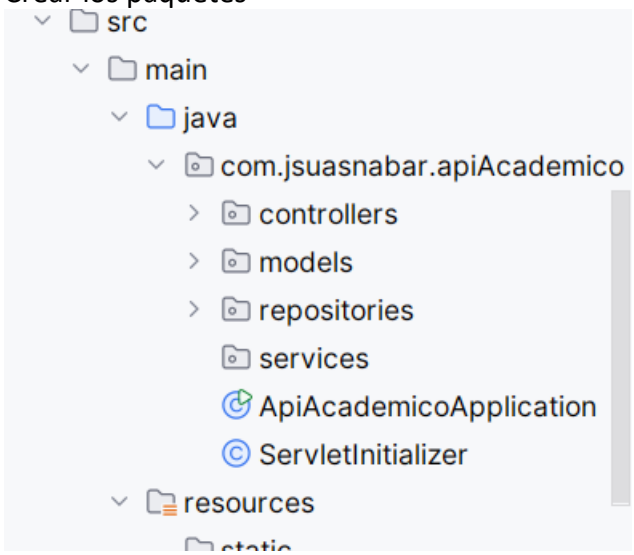


EJERCICIO 03. CONFIGURAR PROPERTIES PARA MYSQL Y CREAR LOS PAQUETES CONTROLLERS, MODELS, REPOSITORIES Y SERVICES

1. Confiigurar Properties del Proyecto.

```
spring.datasource.url=jdbc:mysql://localhost:3306/academico
spring.datasource.username=root
spring.datasource.password=123456
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

2. Crear los paquetes



EJERCICIO 04. CREACIÓN DEL MODELO ESTUDIANTE

```
package com.jsuasnabar.apiAcademico.models;

import jakarta.persistence.*;

@Entity
@Table(name = "estudiante")
public class Estudiante {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long idEstudiante;

   private String nomEstudiante;
   private String dirEstudiante;
   private String ciuEstudiante;

   // Getters y setters

   public String getNomEstudiante() {
      return nomEstudiante;
   }

   public void setNomEstudiante(String nomEstudiante) {
      this.nomEstudiante = nomEstudiante;
   }

   public String getDirEstudiante() {
      return dirEstudiante;
   }

   public void setDirEstudiante(String dirEstudiante) {
      this.dirEstudiante = dirEstudiante;
   }

   public String getCiuEstudiante() {
      return ciuEstudiante;
   }

   public void setCiuEstudiante(String ciuEstudiante) {
      this.ciuEstudiante = ciuEstudiante;
   }
}
```

EJERCICIO 05. CREACIÓN DEL ESTUDIANTEREPOSITORY

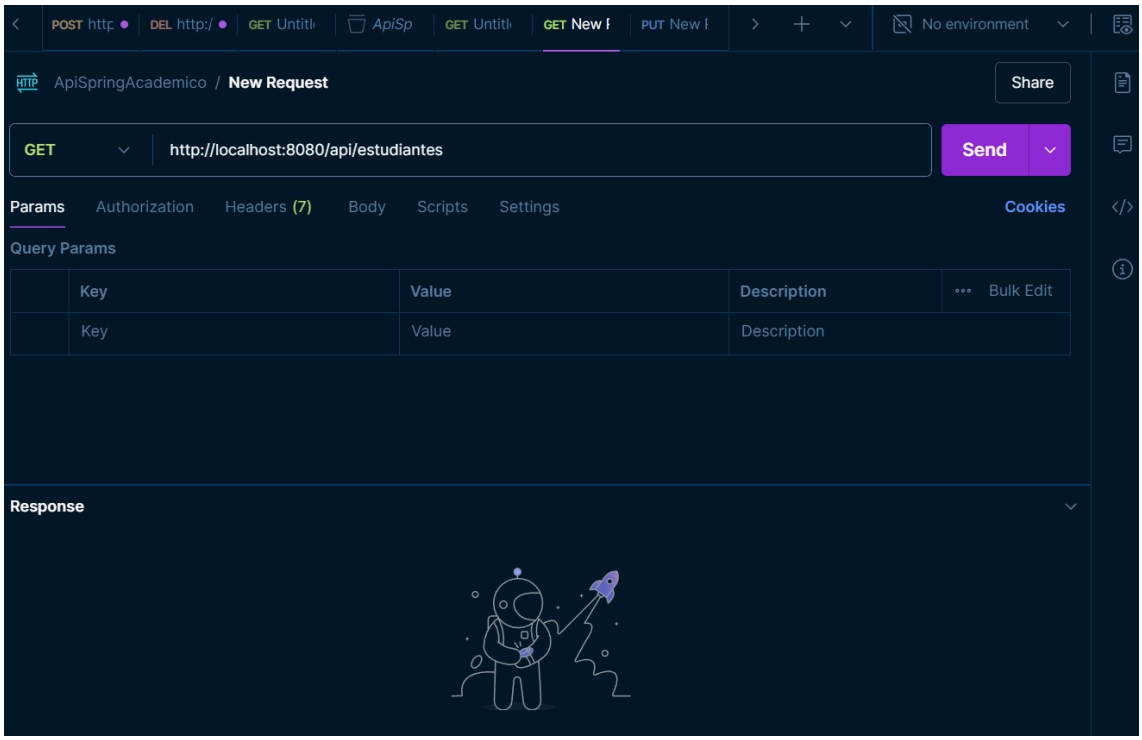
```
package com.jsuasnabar.apiAcademico.repositories;

import com.jsuasnabar.apiAcademico.models.Estudiante;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EstudianteRepository extends JpaRepository<Estudiante, Long>{
}
```

EJERCICIO 06. CREACIÓN DEL ESTUDIANTECONTROLLER

```
package com.jsuasnabar.apiAcademico.controllers;

import com.jsuasnabar.apiAcademico.models.Estudiante;
import com.jsuasnabar.apiAcademico.repositories.EstudianteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/estudiantes")
public class EstudianteController {

   @Autowired
   private EstudianteRepository repo;

   @PostMapping
   public Estudiante agregar(@RequestBody Estudiante estudiante) {
      return repo.save(estudiante);
   }

   @PutMapping("/{id}")
   public ResponseEntity<Estudiante> modificar(@PathVariable Long id, @RequestBody
Estudiante datos) {
      return repo.findById(id)
            .map(est -> {
               est.setNomEstudiante(datos.getNomEstudiante());
               est.setDirEstudiante(datos.getDirEstudiante());
               est.setCiuEstudiante(datos.getCiuEstudiante());
               return ResponseEntity.ok(repo.save(est));
            })
            .orElse(ResponseEntity.notFound().build());
   }

   @DeleteMapping("/{id}")
   public ResponseEntity<Void> eliminar(@PathVariable Long id) {
      if (repo.existsById(id)) {
```

```
            repo.deleteById(id);
            return ResponseEntity.noContent().build();
        }
        return ResponseEntity.notFound().build();
    }

    @GetMapping
    public List<Estudiante> listarTodos() {
        return repo.findAll();
    }

    @GetMapping("/{id}")
    public ResponseEntity<Estudiante> buscarPorId(@PathVariable Long id) {
        return repo.findById(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
```

EJERCICIO 07. COMPROBAR EN POSMAN



Practica Calificada 10

Desarrollar una API RESTful para la gestión de docentes utilizando Jakarta EE *y* Spring. La aplicación permitirá realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar y Otras) sobre una entidad llamada **Docente**, que estará asociada a una base de datos MySQL.

REQUISITOS DE INFORMACIÓN

1. **Entidad Docente:**
   - idDocente
   - nomDocente
   - dirDocente
   - ciuDocente
   - emailDocente
   - fecNacimiento
   - tiempoServicio

REQUISITOS FUNCIONALES

2. **Operaciones a implementar en la API:**
   - Lista todos los docentes
   - Obtiene un docente por ID
   - Crea un nuevo docente
   - Actualiza los datos de un docente
   - Elimina un docente
   - Listar todos los docentes que residen en una ciudad específica (EndPoint: /api/docentes/ciudad/Cusco) (2 PUNTOS)
   - Listar los docentes con al menos cierta cantidad de años de servicio (EndPoint: /api/docentes/experiencia/10. (2 PUNTOS)
   - Calcula y devuelve la edad promedio de todos los docentes registrados (EndPoint: /api/docentes/edad-promedio. (2 PUNTOS)
   - Agregar documentación Swagger. (2 PUNTOS)
   - Manejo de excepciones globales. (2 PUNTOS)
   - Listar los docentes para paginación en el listado de docentes (EndPoint: GET /api/docentes?page=0&size=10) page=0 es la primera página. size=10 indica que se devolverán 10 docentes por página. (8 PUNTOS)
3. **Validaciones mínimas:**
   - El emailDocente debe tener el formato de correo (2 PUNTOS)
   - El tiempoServicio no puede ser negativo. (2 PUNTOS)
   - La fecNacimiento debe ser anterior a la fecha actual. (2 PUNTOS)
4. **Framework:**
   Utiliza Jakarta versión 24, SPRING y Spring Data y MySQL.

5. **Demostración:**
   POSMAN.

Ejercicios Propuestos

1. Desarrolle un controlador de API que compruebe que el id y nota de la ruta "/student/{id}/{nota}" son números. En caso de certeza mostrar un json que contiene el id, nota, un mensaje de ruta válida y el estado 202.
2. Desarrolle un controlador de API con los id y notas según la siguiente ruta "/student/{id}?nota1=xx&nota2=xx&nota3=xx". Que muestre un json que contiene el id, promedio, un mensaje de ruta válida y el estado 202.
3. Diseñar y desarrolle una API REST para que devuelva datos desde una base de datos de 3 tablas departamento(id, name), provincia(id, name) y distrito(id, name).
4. Diseñar y desarrolle una API REST para realizar las operaciones CRUD desde y hacia una base de datos de 3 tablas departamento(id, name), provincia(id, name) y distrito(id, name).