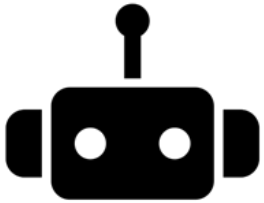
SESIÓN DE LABORATORIO
Teoria de Aplicaciones Web

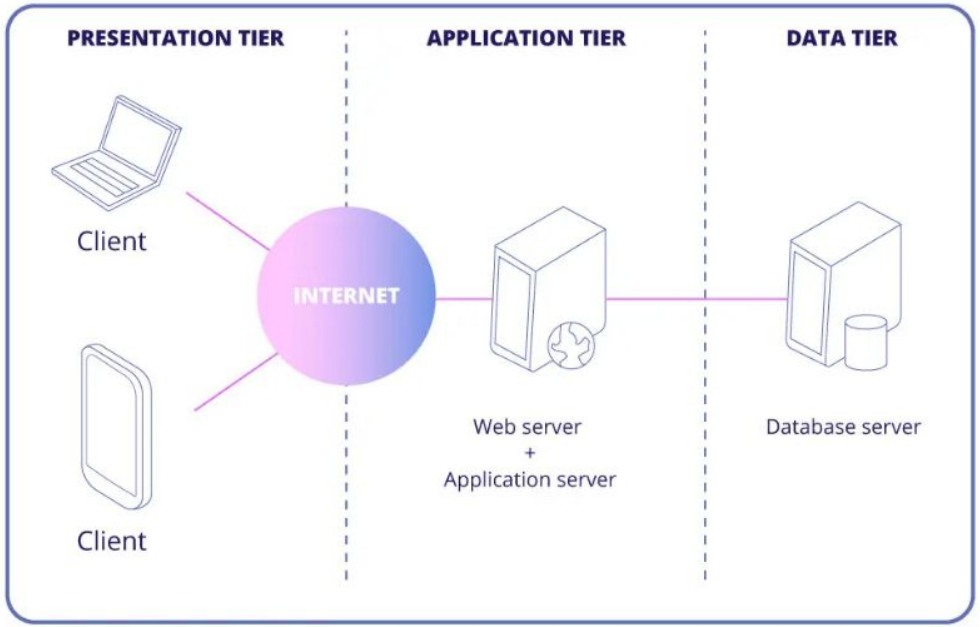


HORARIO 10M1

Bienvenidos !

Empezaremos a las 6:10 p.m
Gracias! c:

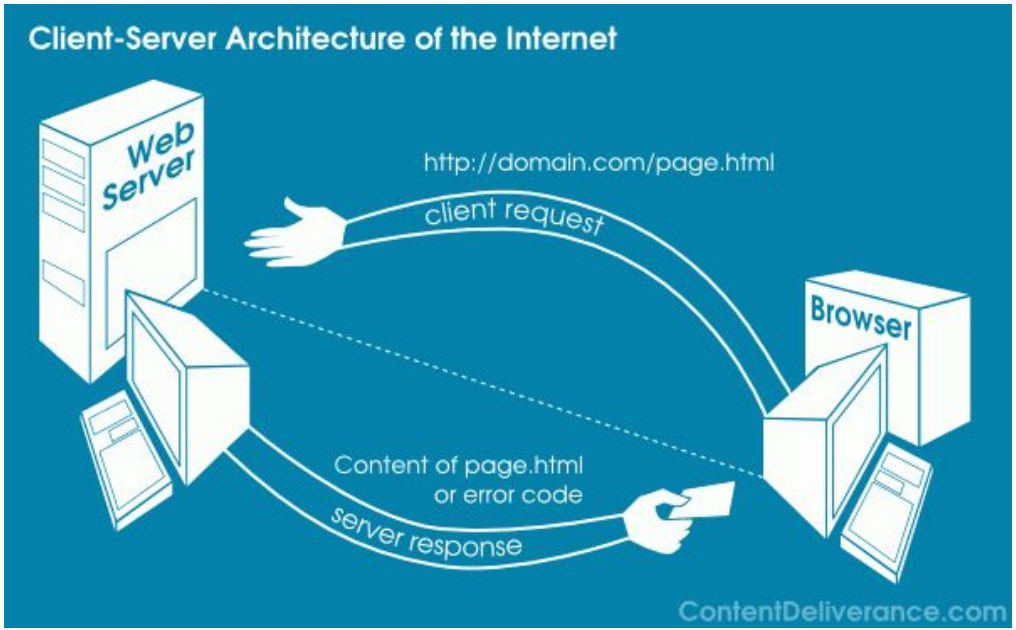
Teoría de aplicaciones web



Una aplicación web , es un software que correr en un servidor web, donde la data necesaria es procesada antes de ser mostrada

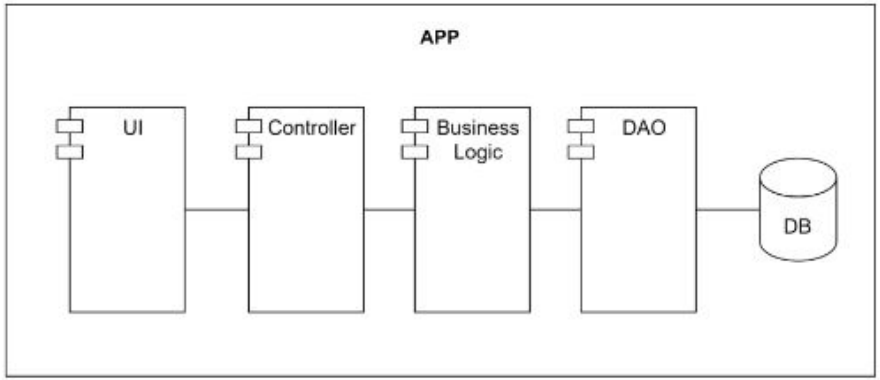
Ejemplos
- Google
- Correo electrónico
- Intranet Pucp
- Redes sociales
- Wikipedia
- etc...

Teoría de aplicaciones web
Arquitectura Cliente - Servidor

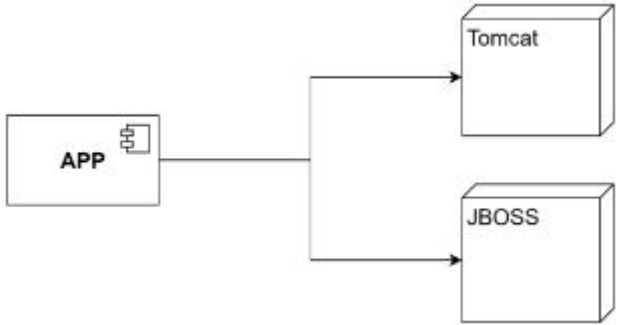


Teoría de aplicaciones web
Arquitectura monolítica

Aplicación :



Desarrollo :



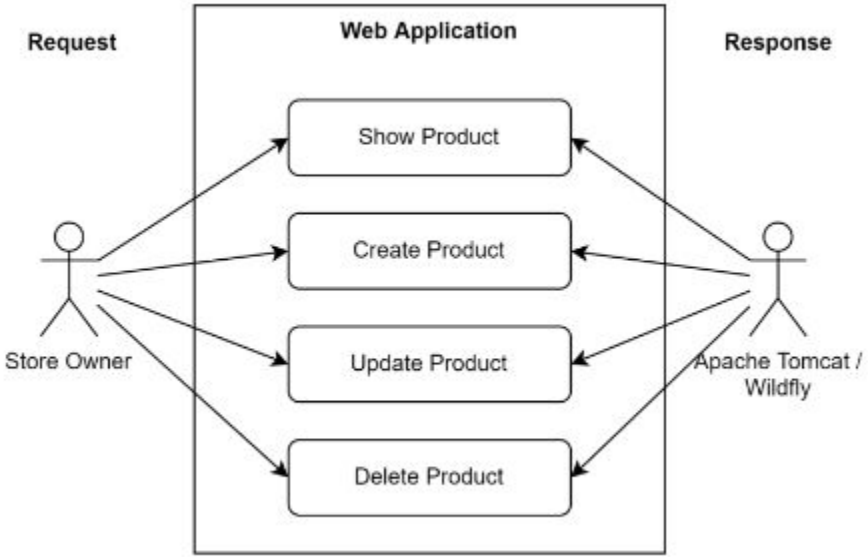
Desarrollo de nuestra aplicación

Sistema de ingesta de productos

Funcionalidades :
Listado de productos
Creación de productos
Modificación de productos
Eliminación de productos

ID	Código	Descripción	Precio de compra	Precio de venta	Existencia	Editar	Eliminar
1	1	Galletas chokis	10.00	15.00	2.00		
2	2	Mermelada de fresa	65.00	80.00	97.00		
3	3	Aceite	18.00	20.00	100.00		
4	4	Palomitas de maíz	12.00	15.00	98.00		
5	5	Doritos	5.00	8.00	99.00		

Desarrollo de nuestra aplicación CRUD



Desarrollo de nuestra aplicación
Requisitos

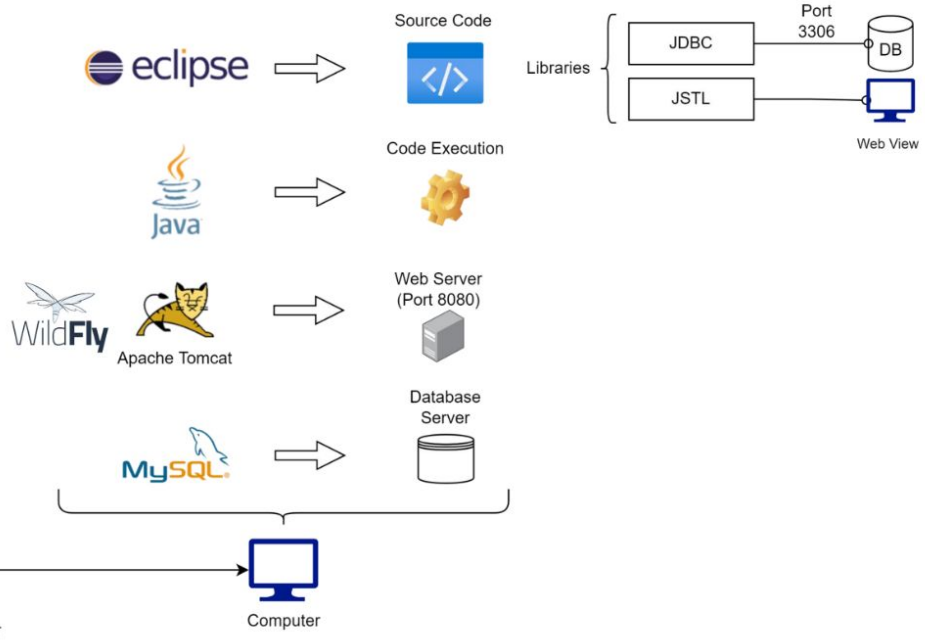
- Java JDK 8
- Eclipse IDE
- Apache Tomcat 9
- MySQL Server
- MySQL Connector for Java 8

Apache Tomcat

eclipse

Desarrollo de nuestra aplicación
Requisitos

- Java JDK 8
- Eclipse IDE
- Apache Tomcat 9
- WildFly (JBOSS Community Version)
- MySQL Server
- MySQL Connector for Java 8



- Todo lo necesario puede ser descargado desde este link :

https://drive.google.com/drive/folders/18stJmTYo3SKm4TszNuNUS4kvEioZvHd6

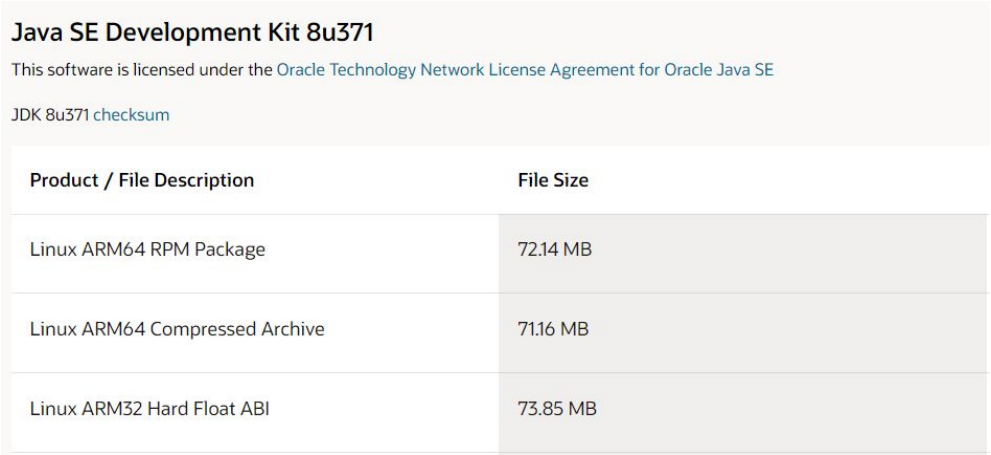
Instalaciones

Java JDK 8 Instalación (1/4)

• (Nota: Para poder descargar desde Oracle es necesario tener una cuenta)

• Deben ir al siguiente url: https://www.oracle.com/java/technologies/downloads/archive/

• Luego deben bajar hasta la sección de java SE 8 y seleccionar el instalador para windows que debe ser del tipo "jdk-8u371..." en el apartado de **Java SE Development Kit 8u371**



Java SE Development Kit 8u371

This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

JDK 8u371 checksum

Product / File Description	File Size
Linux ARM64 RPM Package	72.14 MB
Linux ARM64 Compressed Archive	71.16 MB
Linux ARM32 Hard Float ABI	73.85 MB

Instalaciones

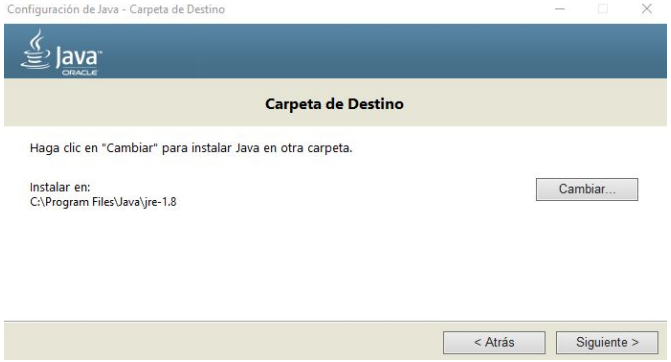
Java JDK 8 Instalación (2/4)

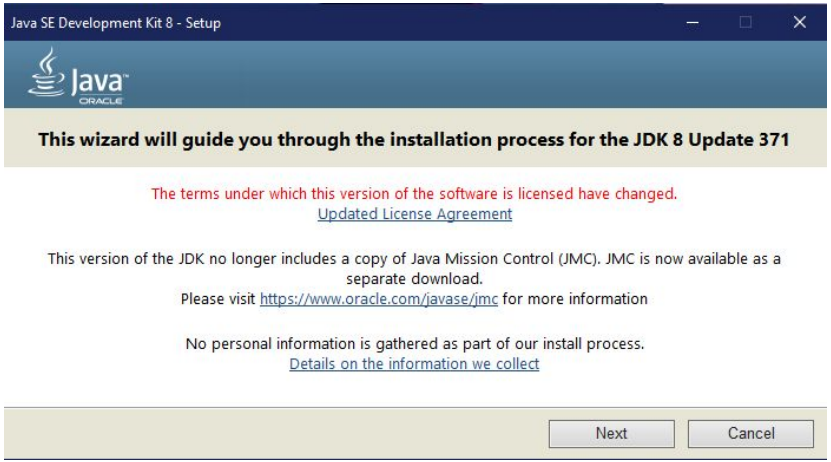
- Deben aceptar los términos de licencia
- Una vez se logeen , la descarga empezará

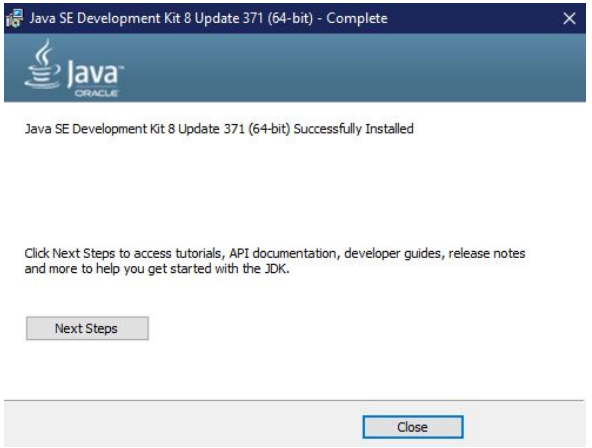
Instalaciones

Java JDK 8 Instalación (3/4)

- Luego deben de ejecutar el instalador
- Deberán hacer click en siguiente en cada pantalla



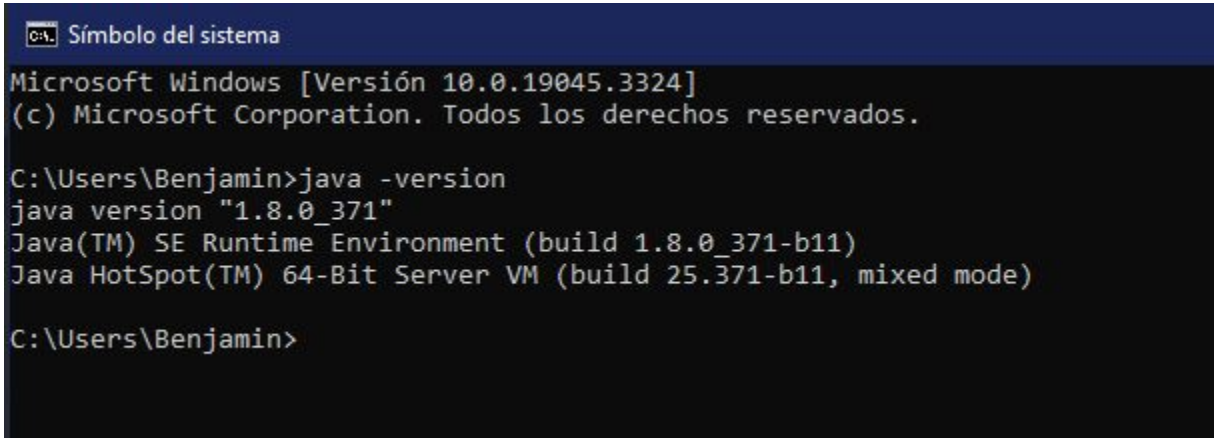




Instalaciones

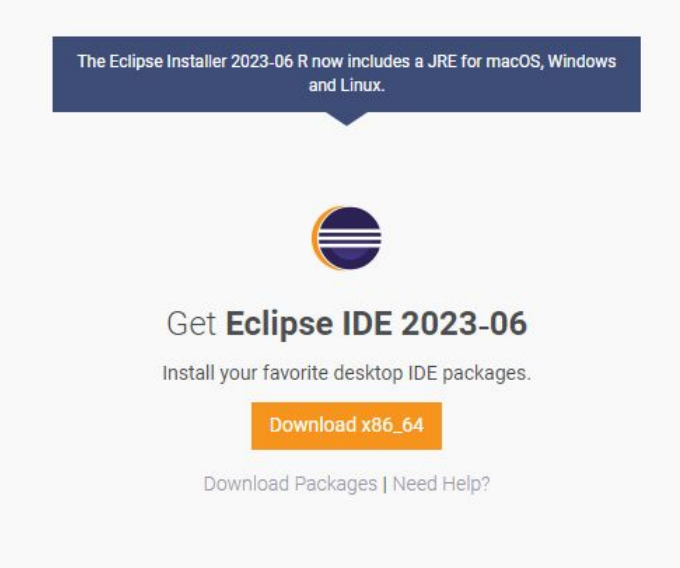
Java JDK 8 Instalación (4/4)

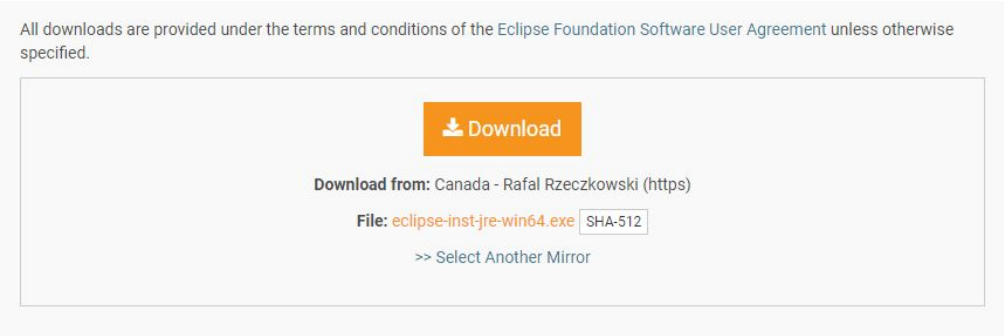
- Para poder verificar que todo se ha instalado correctamente
- Deben hacer Win + R → CMD → java -version



Instalaciones

Eclipse IDE Instalación (1/2)
Deben ir al siguiente url :
https://www.eclipse.org/downloads/
• Click “Download x86_64” -> Download

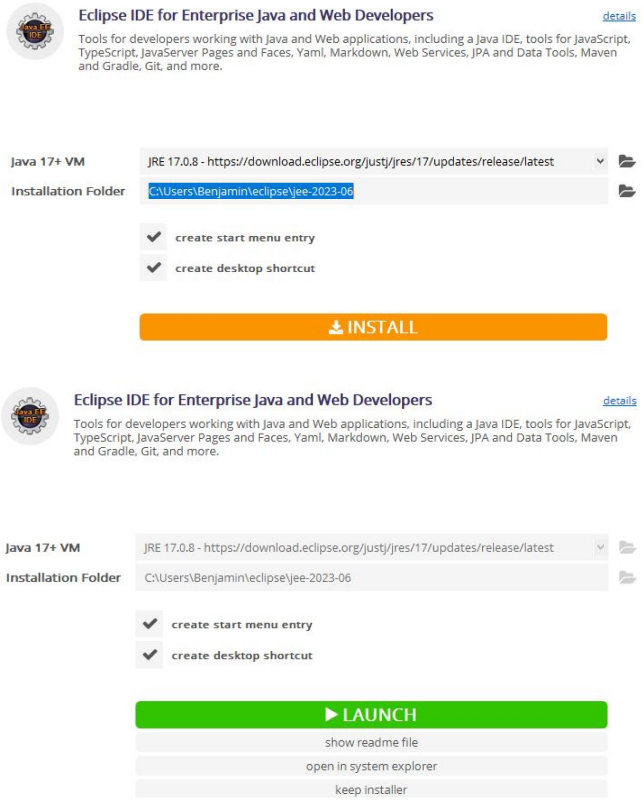




Instalaciones

Eclipse IDE Instalación (2/2)
Al abrir el instalador , deberán seleccionar Eclipse IDE for Enterprise Java and Web Developers y seleccionar en instalar

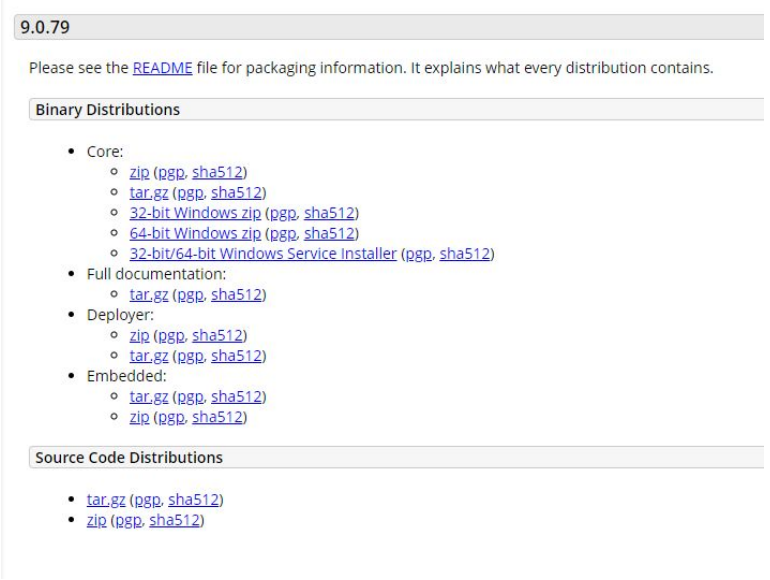
Es importante aclarar que el uso de Java 17 será cambiado luego por Java 8



Instalaciones

Apache Tomcat Instalación (1/1)
Descargarlo desde :
https://tomcat.apache.org/download-90.cgi
utilizando la opción Zip

Extraerlo en un folder



9.0.79

Please see the README file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
  - zip (pgp, sha512)
  - tar.gz (pgp, sha512)
  - 32-bit Windows zip (pgp, sha512)
  - 64-bit Windows zip (pgp, sha512)
  - 32-bit/64-bit Windows Service Installer (pgp, sha512)
- Full documentation:
  - tar.gz (pgp, sha512)
- Deployer:
  - zip (pgp, sha512)
  - tar.gz (pgp, sha512)
- Embedded:
  - tar.gz (pgp, sha512)
  - zip (pgp, sha512)

Source Code Distributions

- tar.gz (pgp, sha512)
- zip (pgp, sha512)

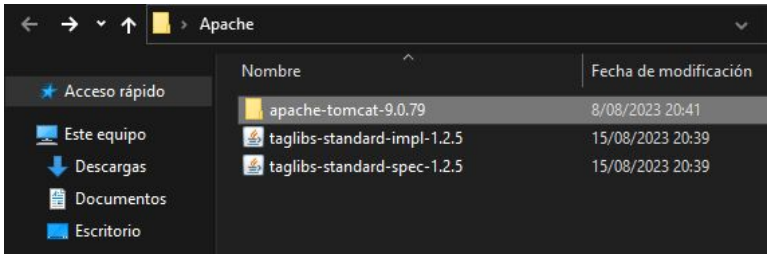
Instalaciones

JSTL Instalación (1/1)
Es una librería para manejar tags en programas java

Descargarlo desde :
https://tomcat.apache.org/download-taglibs.cgi

Donde se deberán escoger las opciones que contienen Impl y Spec

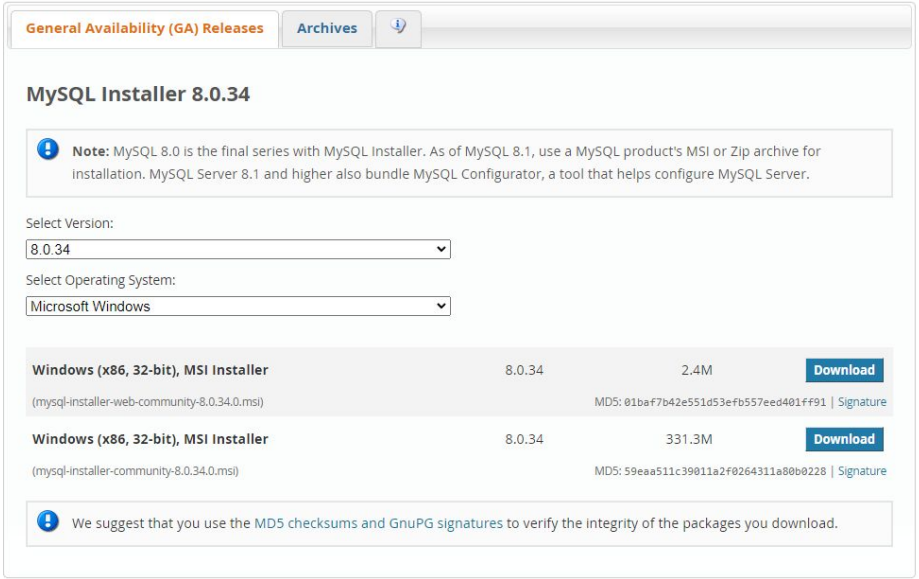
Finalmente se deberán poner en el mismo folder que Apache



Instalaciones

MySQL Server Instalación (1/12)
Descargarlo desde :
https://dev.mysql.com/downloads/windows/installer/8.0.html

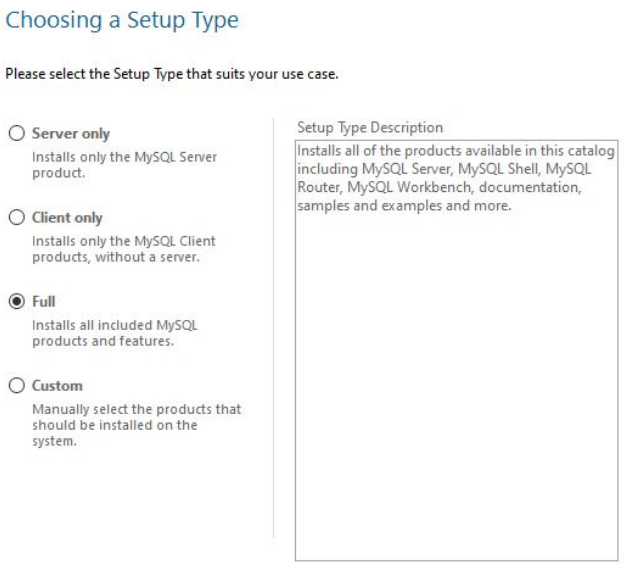
Donde se usará la versión 8.0.34 y la versión community

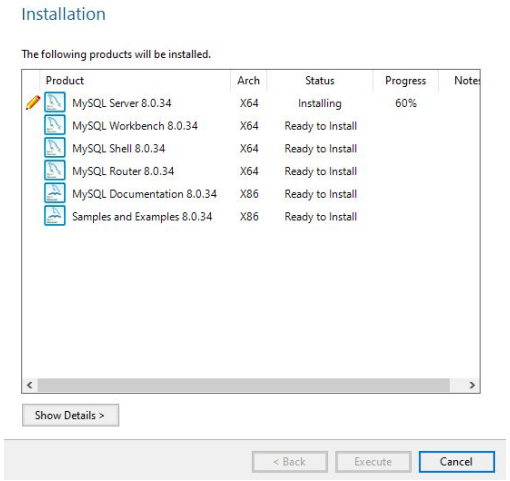


Instalaciones

MySQL Server Instalación (2/12)
Una vez se inicie la instalación haremos click en Full

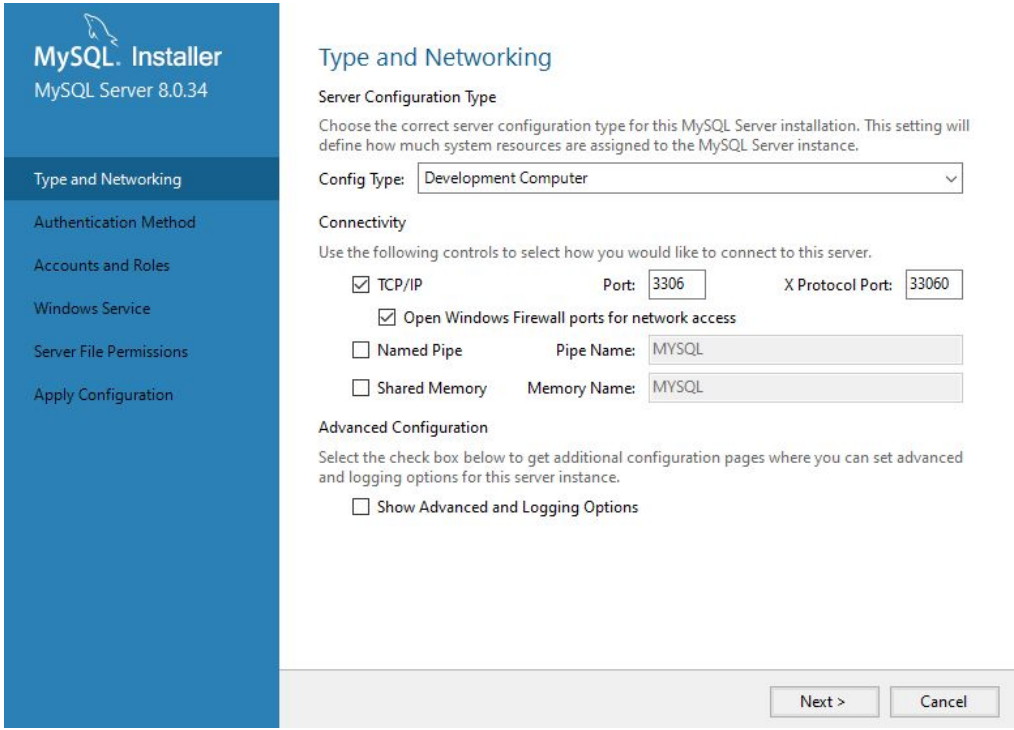
Luego en siguiente y en ejecutar y esperaremos a que culmine la instalación





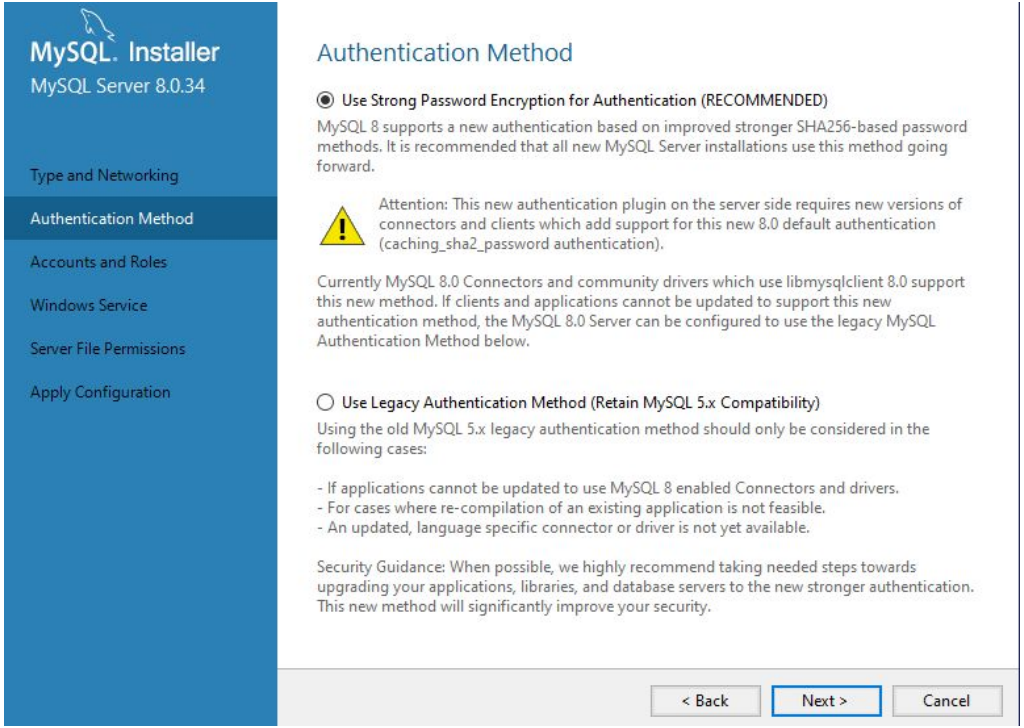
Instalaciones

MySQL Server Instalación (3/12)
Una vez terminada la instalación en el apartado de Type and Networking debemos asegurarnos que el config Type está en Development Computer y apuntar los valores de Port : 3306 y X Protocol Port 33060



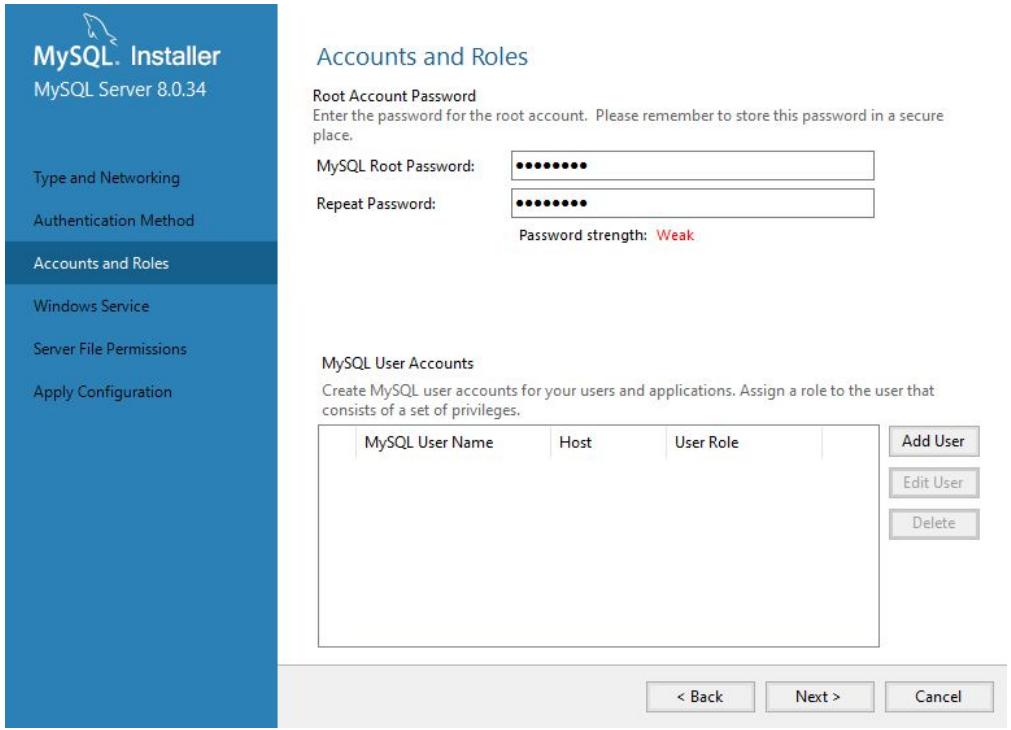
Instalaciones

MySQL Server Instalación (4/12)
En el siguiente apartado se usará la opción recomendada para encriptación.



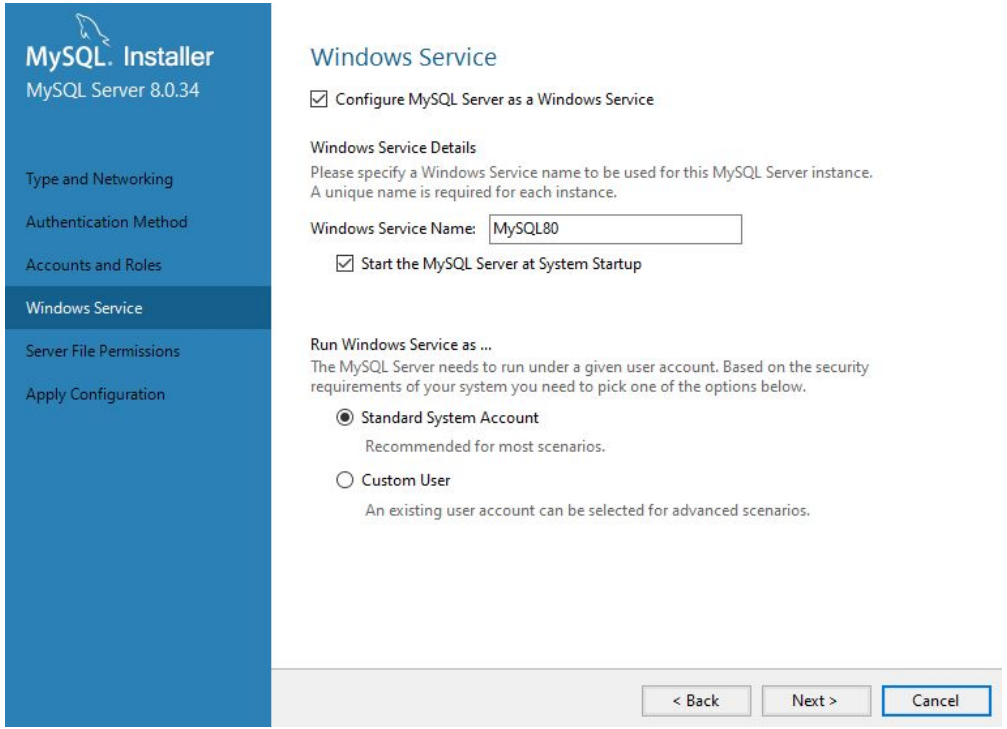
Instalaciones

MySQL Server Instalación (5/12)
En el siguiente apartado debemos escoger una contraseña la cual debemos recordar dado que la usaremos para acceder a la base de datos



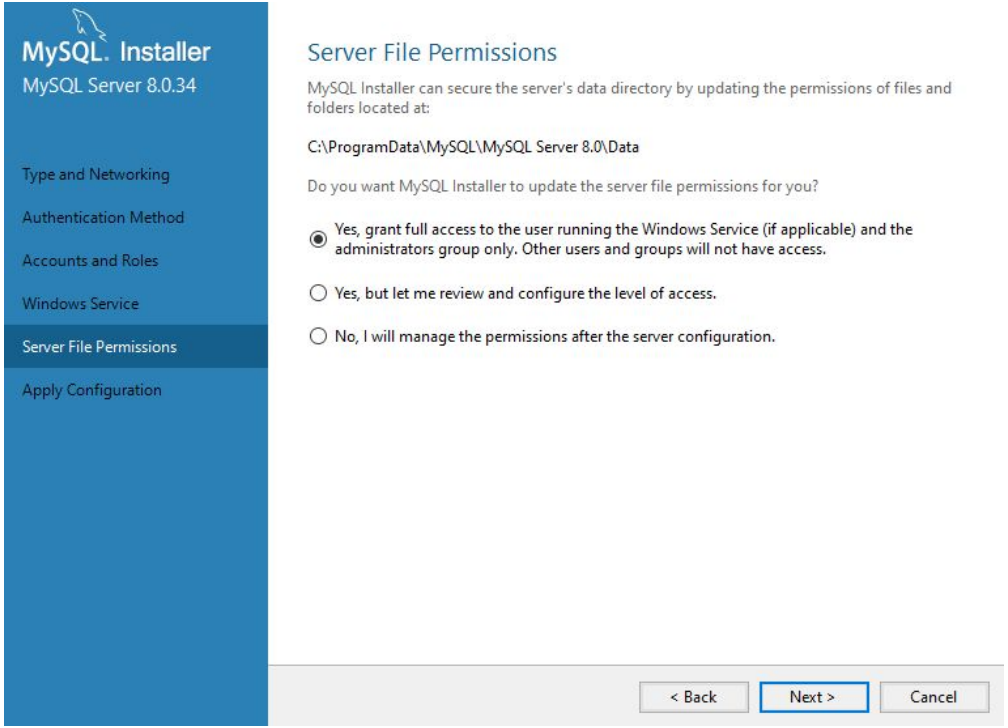
Instalaciones

MySQL Server Instalación (6/12)
Luego deberemos aceptar la configuración por default de MySQL como servicio de windows y aplicarlas



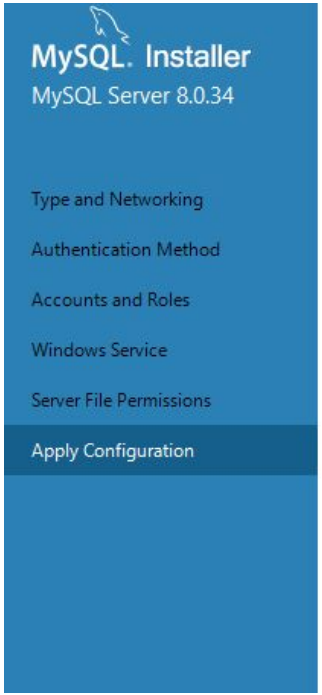
Instalaciones

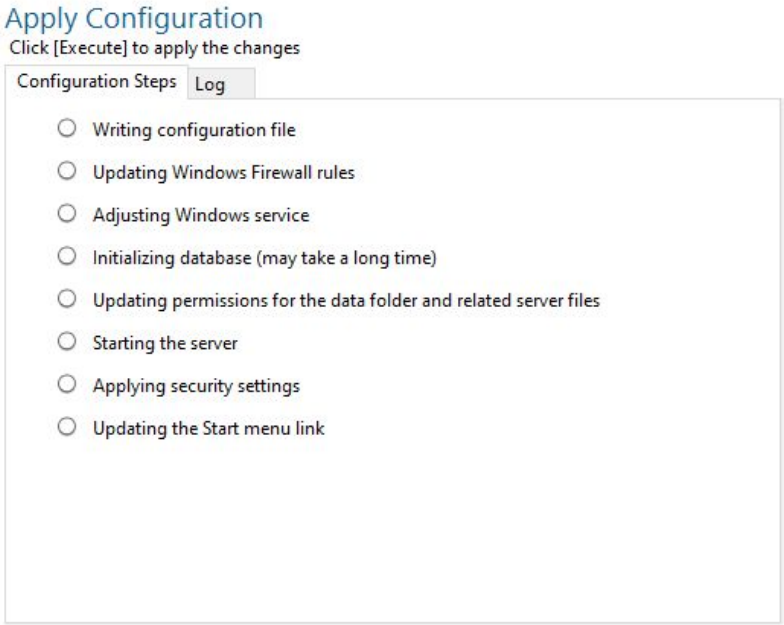
MySQL Server Instalación (7/12)
Luego debemos asignar permisos totales



Instalaciones

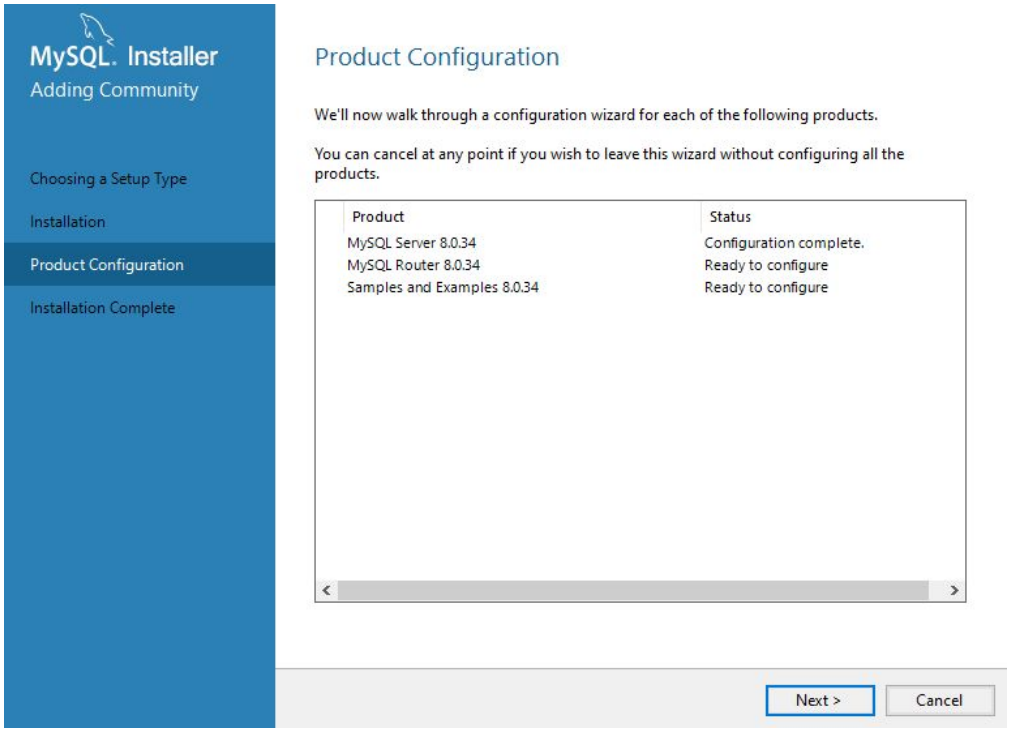
MySQL Server Instalación (8/12)
Finalmente se aplican las configuraciones dandole click en execute





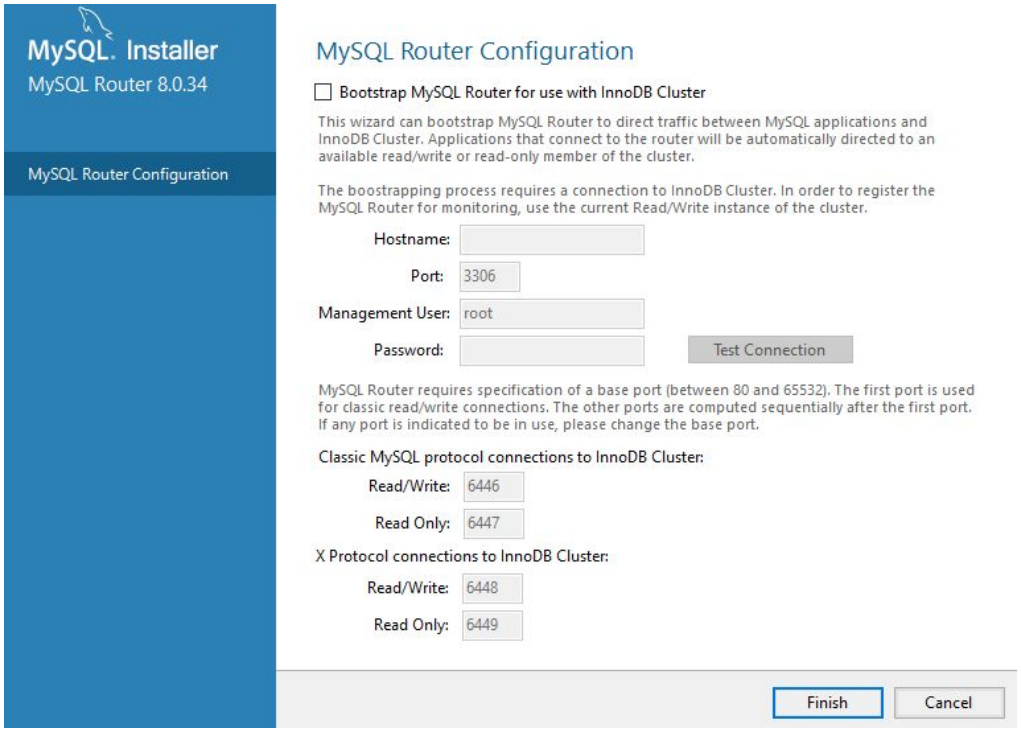
Instalaciones

MySQL Server Instalación (9/12)
Continuando se deberá hacer la configuración de los otros productos



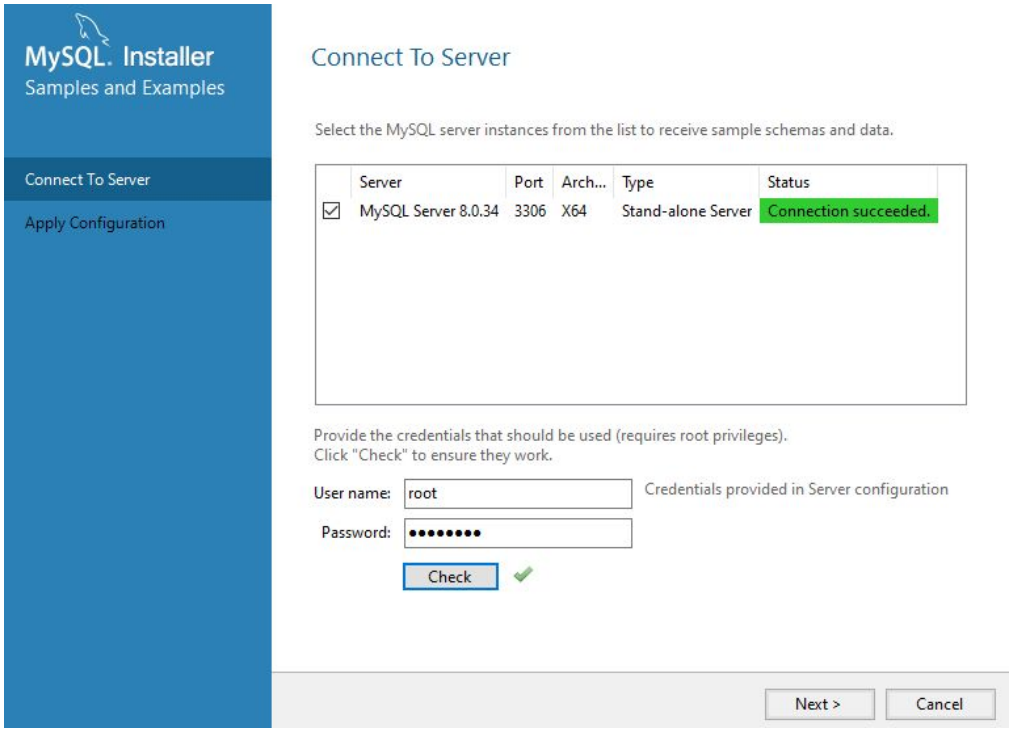
Instalaciones

MySQL Server Instalación (10/12)
Respecto a la configuración de Router , se debe mantener la configuración



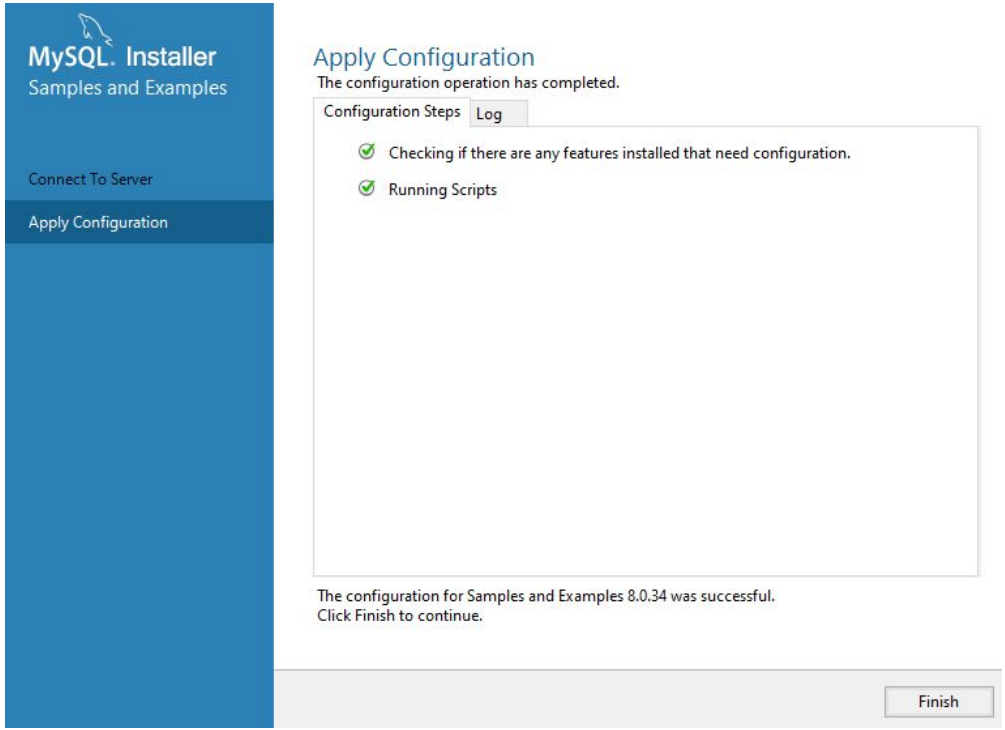
Instalaciones

MySQL Server Instalación (11/12)
Finalmente se debe conectar al servidor y aplicar la configuración



Instalaciones

MySQL Server Instalación (12/12)
Finalmente se debe completar la instalación

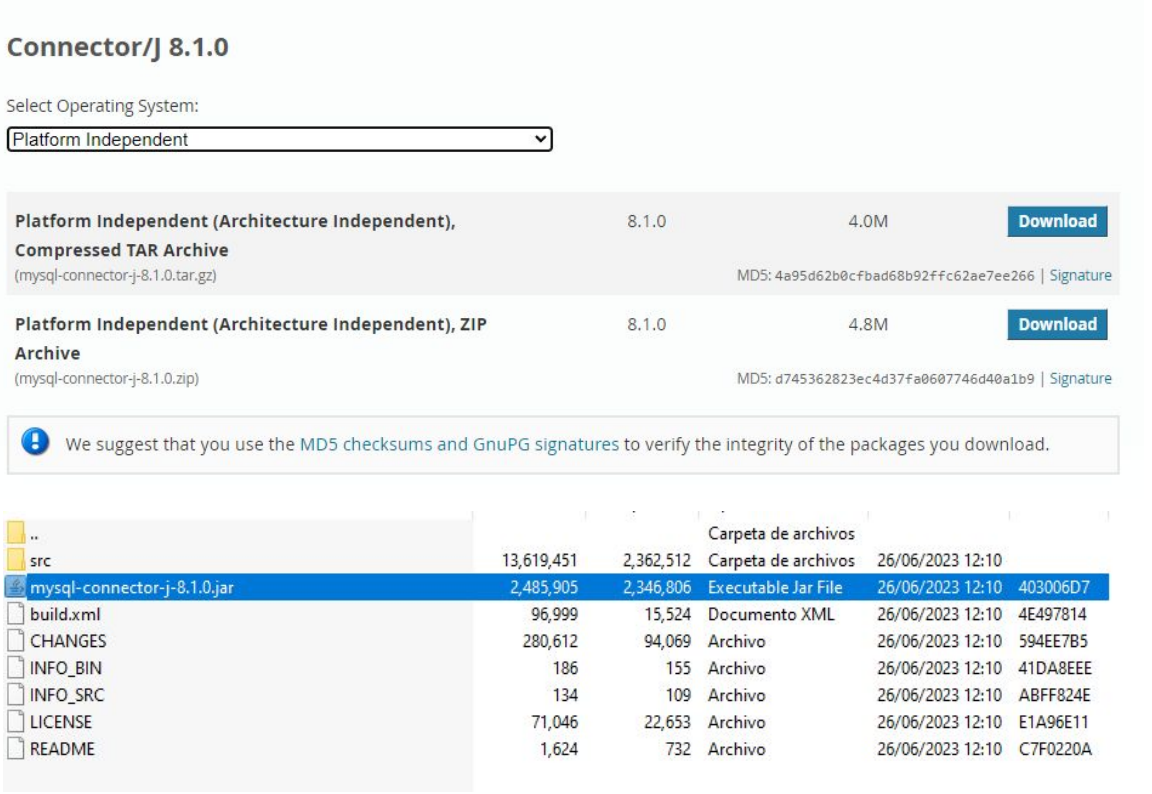


Instalaciones

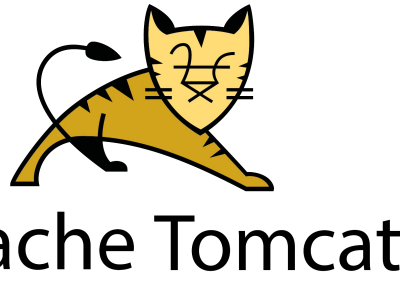
MySQL Connector Download
Se debe descargar desde :
https://dev.mysql.com/downloads/connector/j/

Aquí se debe seleccionar Plataforma Independiente y descargar.

Luego se debe extraer el archivo ZIP y mantener solo el archivo jar

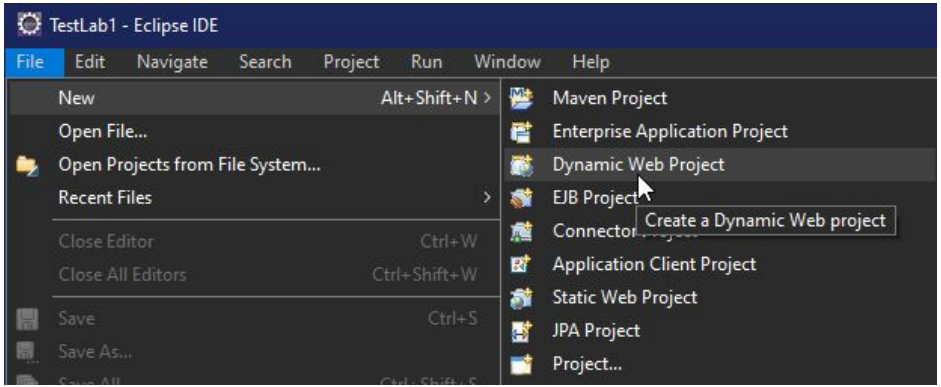


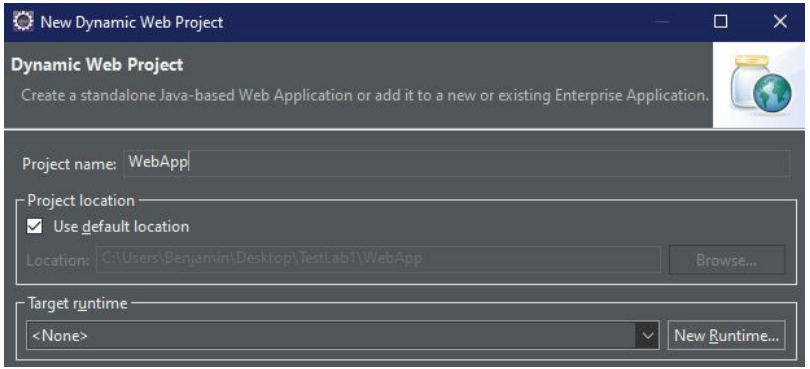
Configuración y creación del proyecto



Configurando Eclipse IDE con Tomcat y Java 8 (1/5)

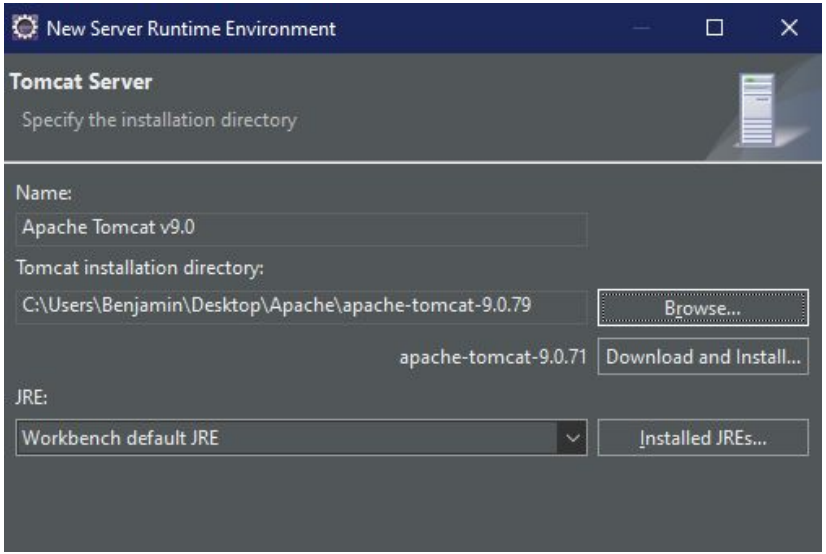
- • File → New → Dynamic Web Project
- Se debe dar un nombre al projecto (WebApp)
- Se da click a New Runtime

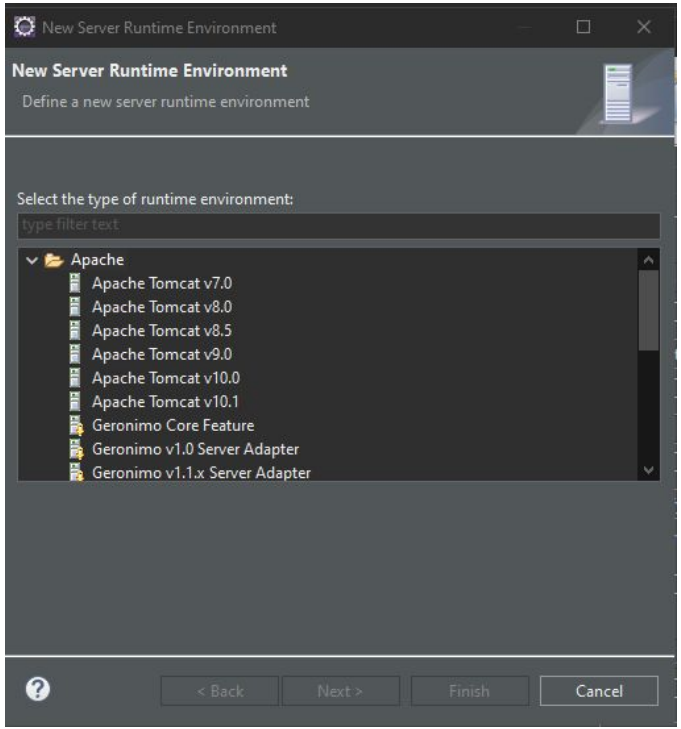




Configurando Eclipse IDE con Tomcat y Java 8 (2/5)

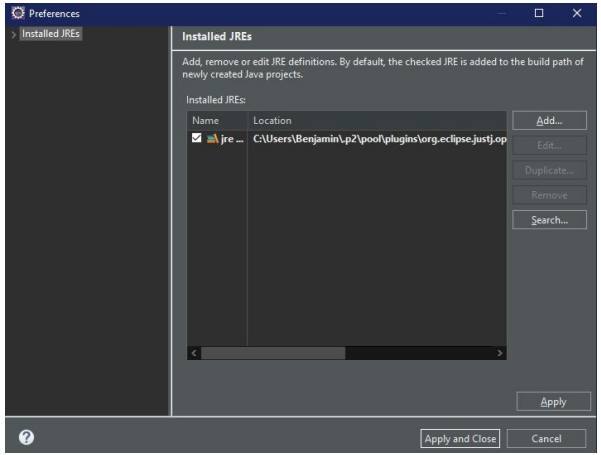
Aquí vamos a seleccionar Apache TomCat 9.0 y seleccionamos la carpeta donde lo descomprimimos y luego le damos click en Finish

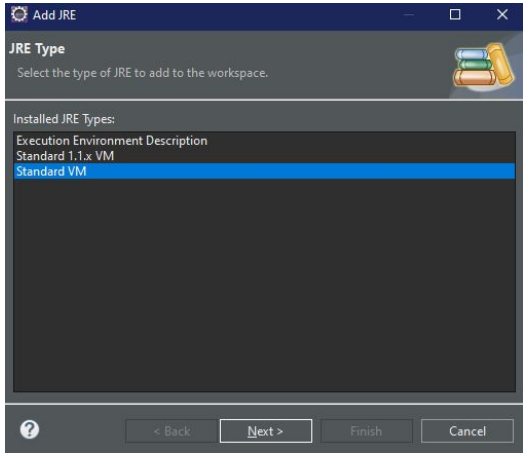


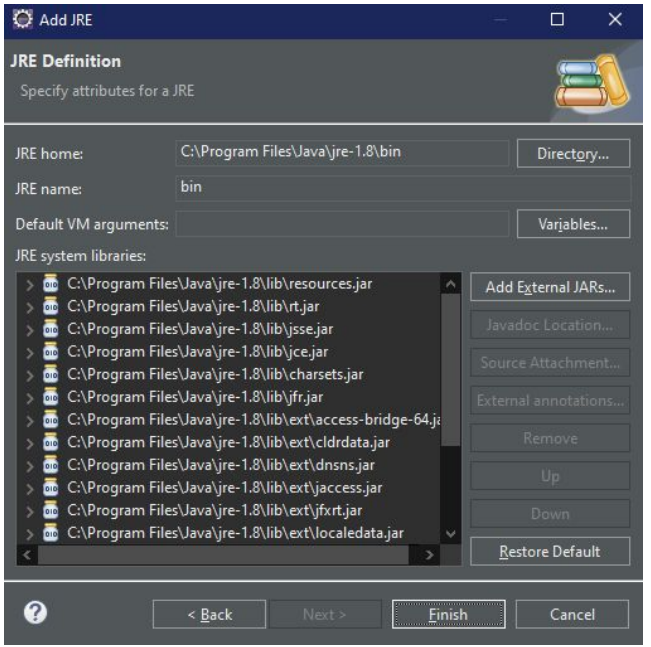


Configurando Eclipse IDE con Tomcat y Java 8 (3/5)

Luego debemos hacer click en Installed JREs y le damos en Add. Aquí haremos click en Standard VM y para JRE Home seleccionaremos “C:\Program Files\Java\jre-1.8\bin” y le damos a finish

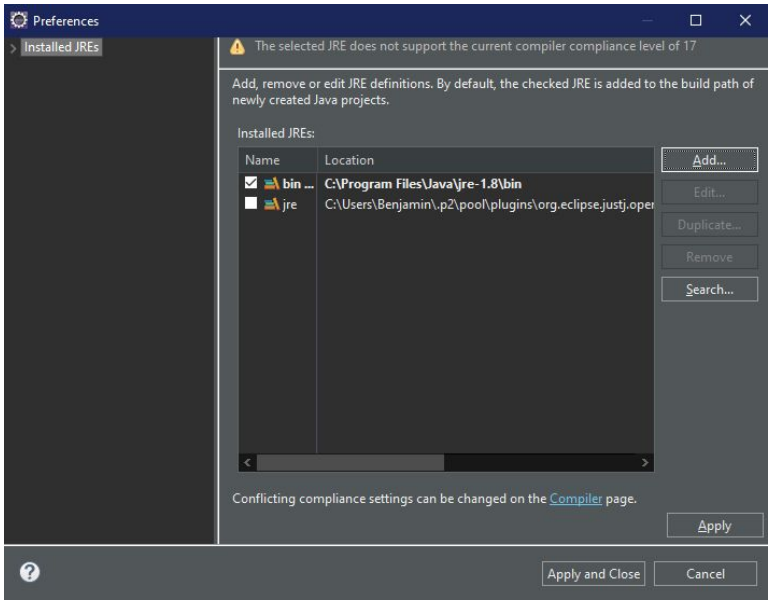


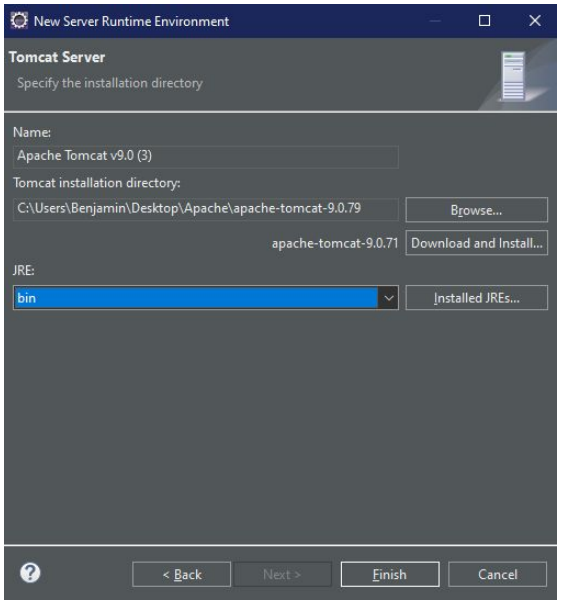




Configurando Eclipse IDE con Tomcat y Java 8 (4/5)

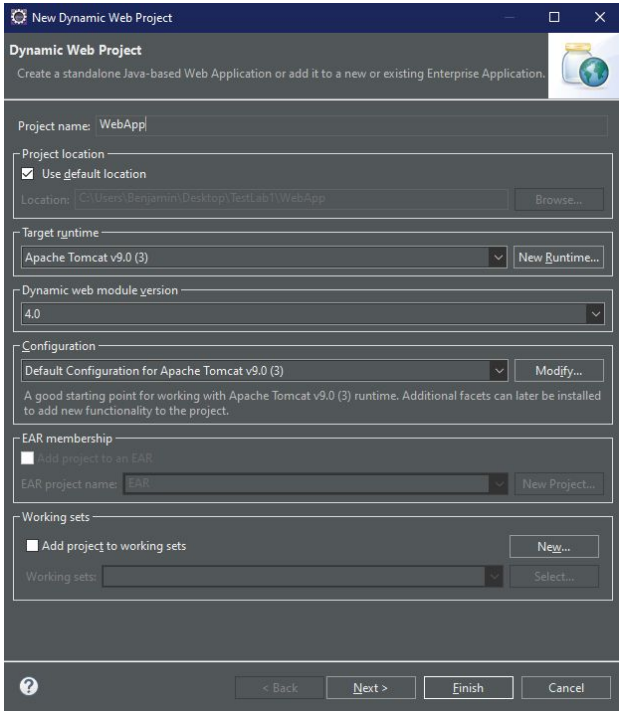
Una vez tengamos el nuevo bin, lo seleccionamos y le damos click en Apply and Close, Finalmente en JRE , Seleccionamos bin y le damos en Finish.





Configurando Eclipse IDE con Tomcat y Java 8 (5/5)

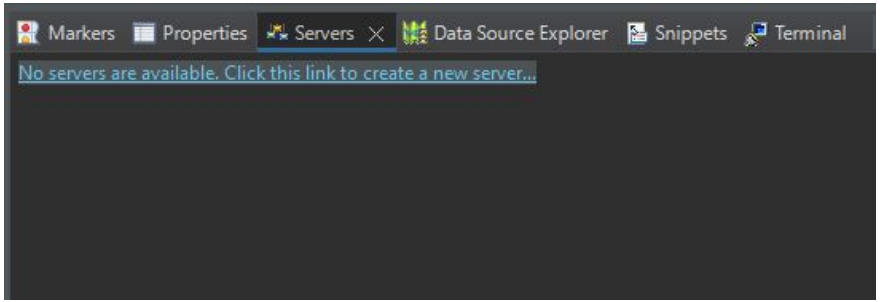
Para finalizar con la configuración le damos click en Finish



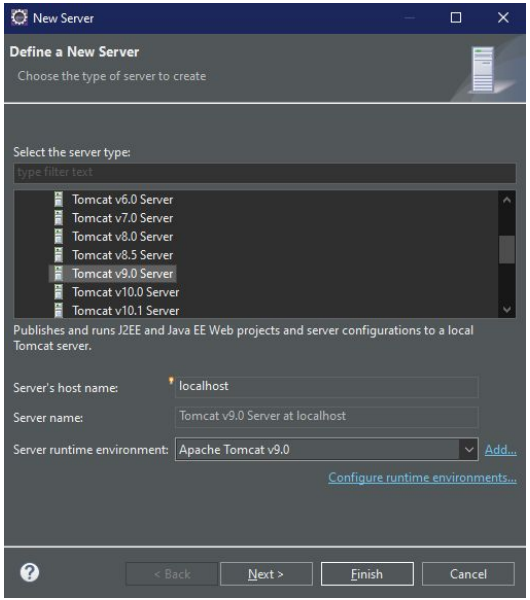
Configuración del servidor (1/2)

Para activar la vista de servidores, debemos ir a window, show view, servers

Una vez hecho esto, la pestaña servidores será visible en la parte inferior del programa

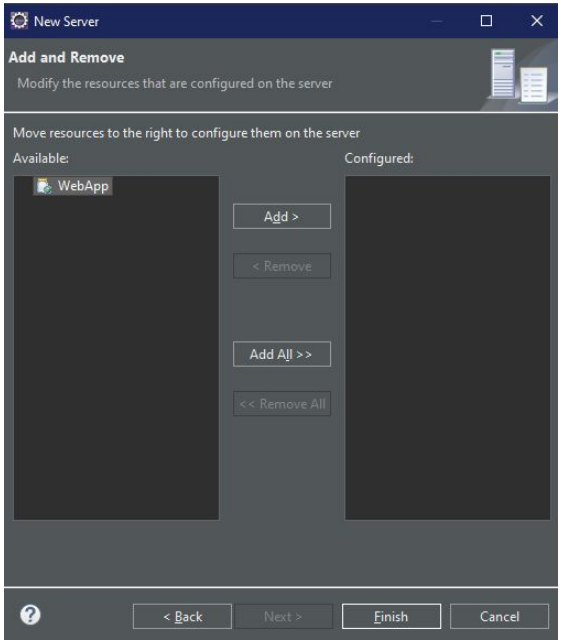


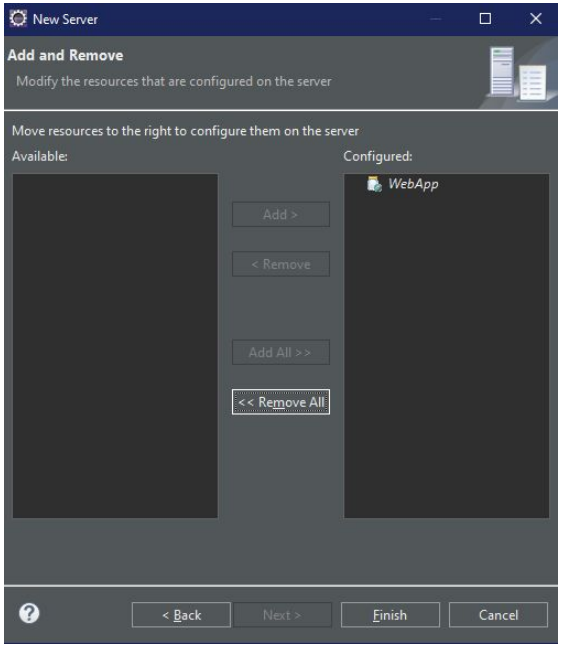
Una vez hecho esto daremos clic en crear nuevo servidor, Apache, Tomcat v9.0 y daremos en next



Configuración del servidor (2/2)

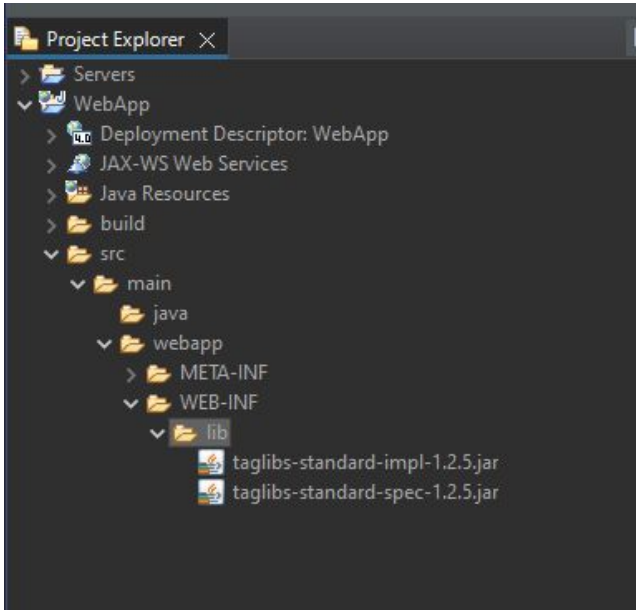
Aquí vamos a mover el proyecto a Configured y le daremos click en Finish





Configuración JSTL

En el folder del proyecto vamos a ubicar la carpeta SRC
Aquí vamos a expandir de la siguiente forma : “src/main/webapp/WEB-INF/lib”.
Aquí vamos a copiar y a pegar los archivos “taglibs-standard-impl-1.2.5.jar” y “taglibs-standard-spec-1.2.5.jar”



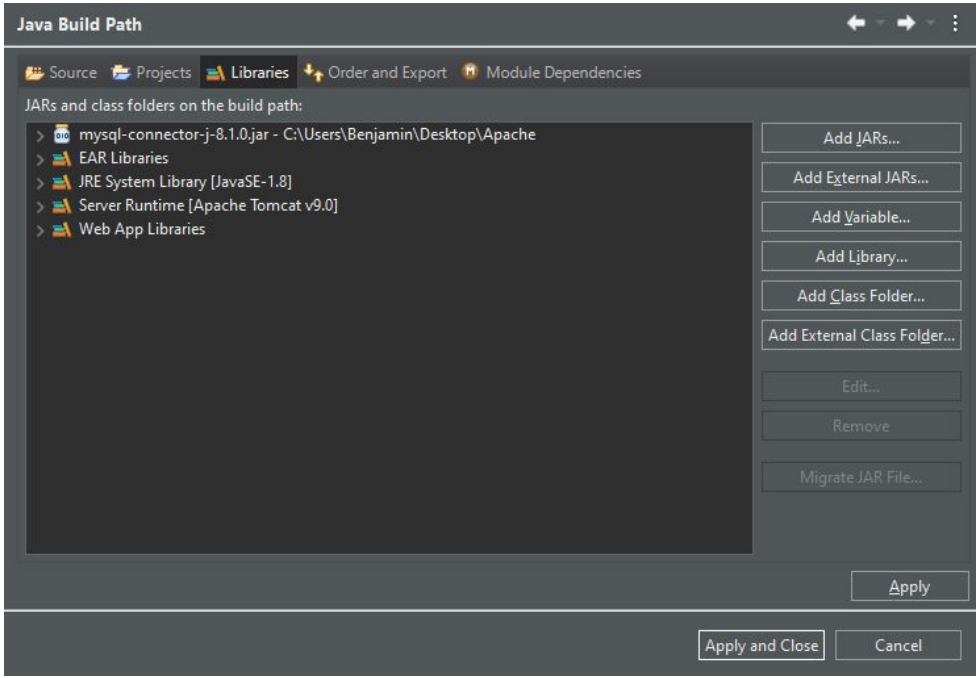
Configuración de la base de datos con MySQL Connector

Para esto vamos darle click derecho al proyecto y darle a properties.

Aquí iremos a Java Build Path y seleccionaremos la pestaña libraries.

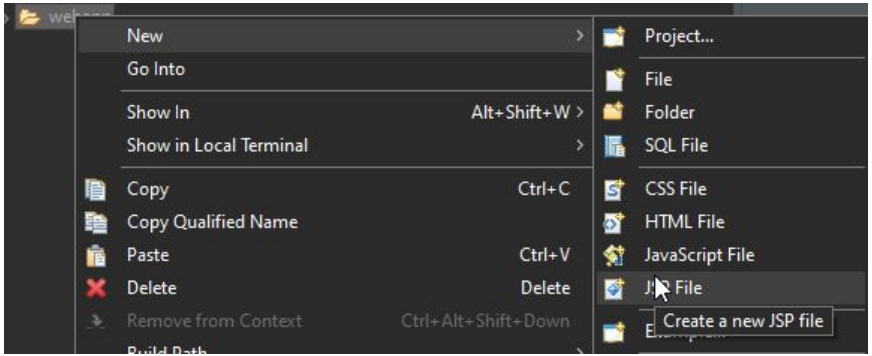
Finalmente le daremos click en Add External JARs y seleccionaremos el archivo previamente descargado.

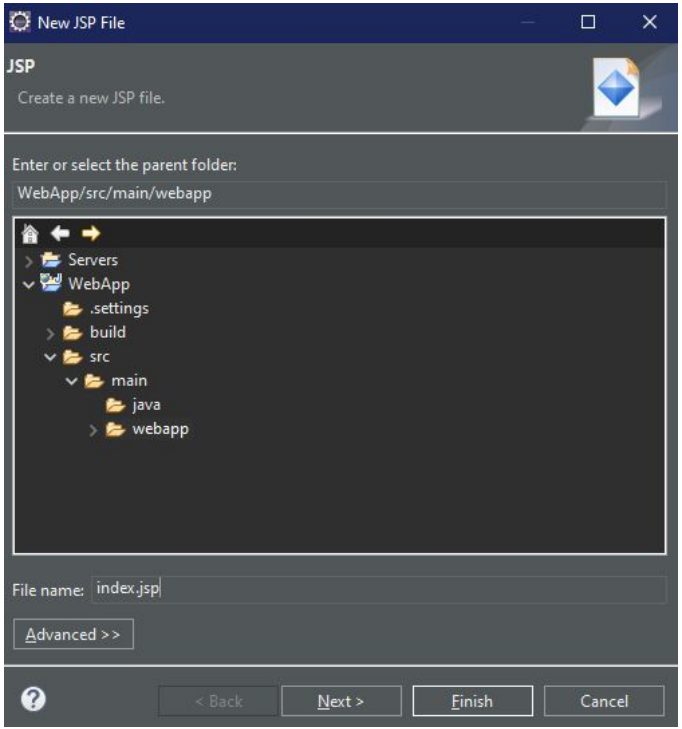
Finalmente le daremos en Apply and Close



Test de nuestro proyecto (1/2)

Para poder hacer el test , lo que vamos a hacer es crear un nuevo archivo JSP en src/main/webapp/ y lo nombraremos como index.jsp

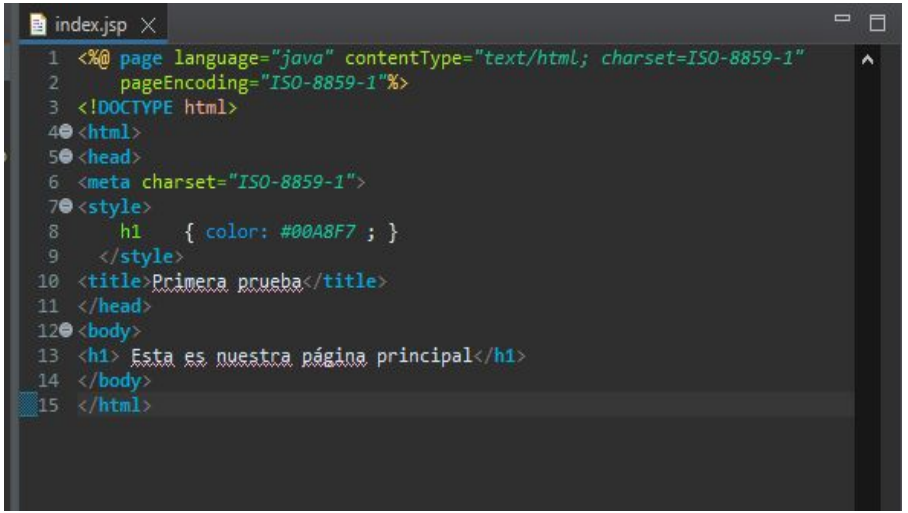




Test de nuestro proyecto (1/2)

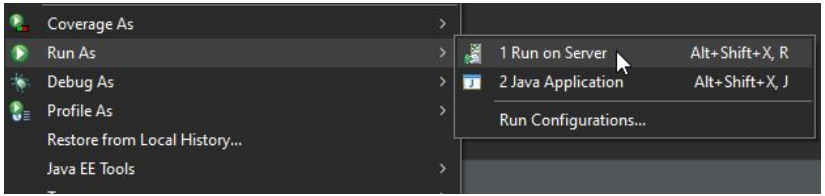
Esto nos abrirá un nuevo documento donde podremos hacer los cambios respectivos para que sea visible en nuestro test, en este caso:
<h1>Esta es nuestra página principal</h1>
Además de un poco de color con :
<style>
h1 { color: #00A8F7 ; }
</style>
Y reemplazaremos el titulo por “Primera Prueba”

Finalmente grabaremos con Ctrl + S



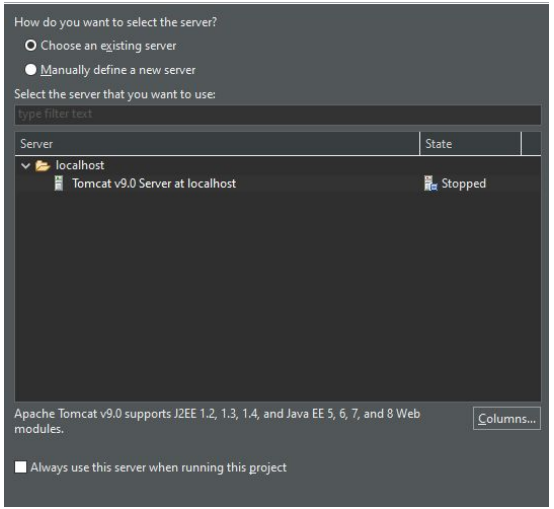
Test de nuestro proyecto (2/2)

Para poder visualizar nuestro test le daremos click derecho a nuestro aplicativo y le daremos a Run As , Run on Server



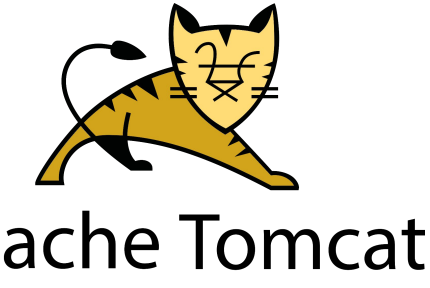
En caso de que nos salga una alerta de firewall , debemos dar click en permitir acceso.

Aquí seleccionaremos nuestro servidor Tomcat 9.0 , Next y Finish



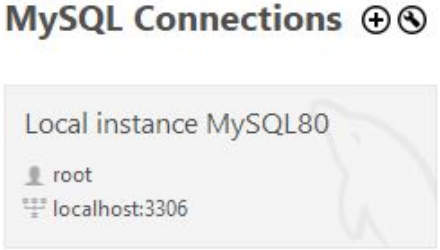
Para poder acceder a nuestro test deberemos abrir una página en nuestro navegador y dirigirnos a “localhost:8080/WebServer/”

Implementación del proyecto

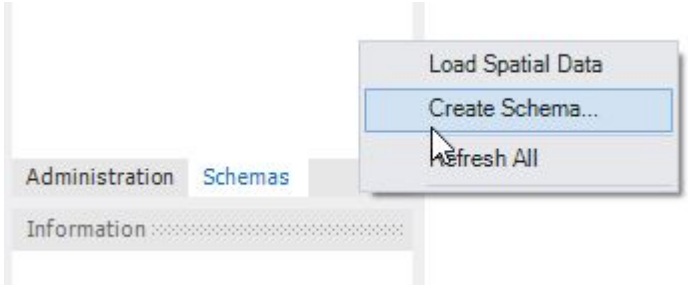


Inicialización de la base de datos (1/3)

Debemos abrir MySQL WorkBench
Luego hacemos click en Local Instance MySQL80 y ingresamos nuestras credenciales

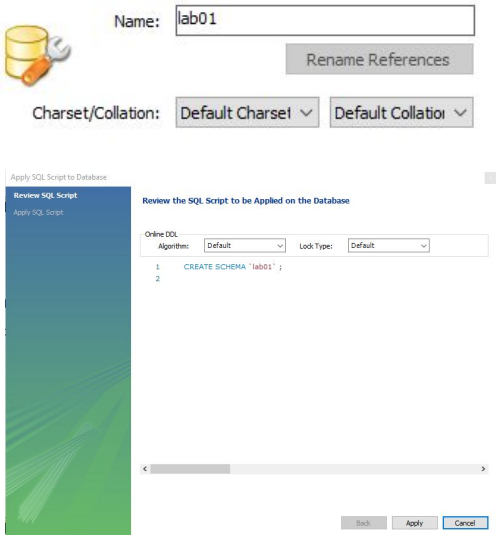


Luego en el apartado Schemas le damos click derecho y seleccionamos create schema

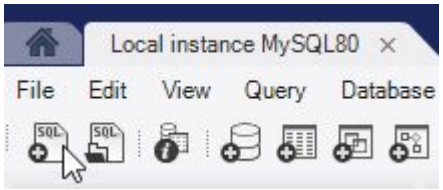


Inicialización de la base de datos (2/3)

Aqui generamos un nuevo schema al cual llamaremos lab 01



Una vez se ha creado el schema generaremos un nuevo query con el boton +SQL y añadiremos el siguiente query




```
CREATE TABLE lab01.product(
id INT NOT NULL AUTO_INCREMENT,
name VARCHAR(45) NOT NULL,
stock INT NOT NULL,
PRIMARY KEY(id)
);
```

```
INSERT INTO lab01.product(name, stock)
VALUES ('Laptop', 1);
```

Inicialización de la base de datos (3/3)

Para poder correr cualquier query , es necesario darle click al rayito

Este query nos generará una nueva tabla llamada productos con un nuevo row

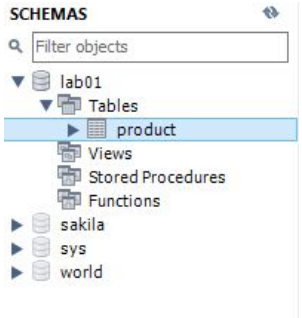
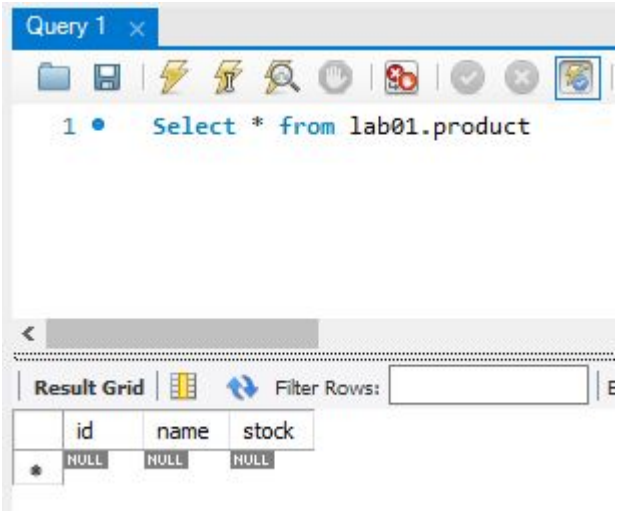


Table: product

Columns:

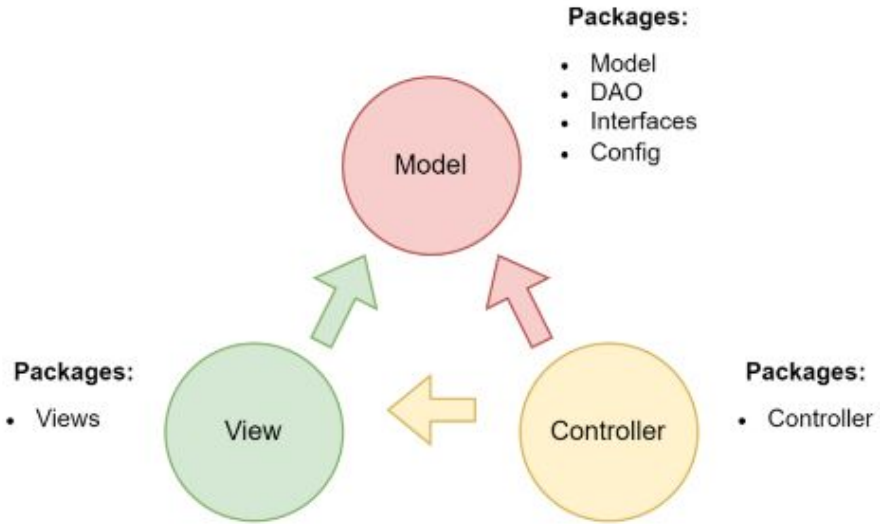
id	int AI PK
name	varchar(45)
stock	int

Para poder ver los valores de nuestra nueva tabla solo debemos hacer Select * from lab01.productos



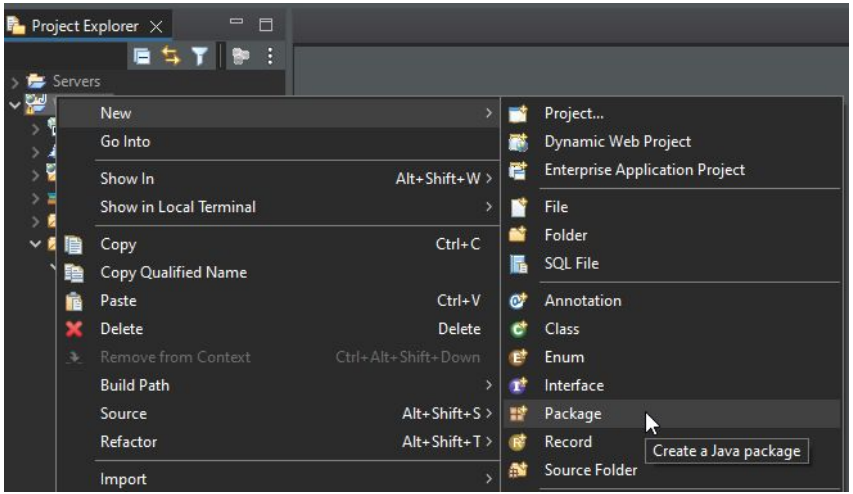
Arquitectura en Capas (1/4)

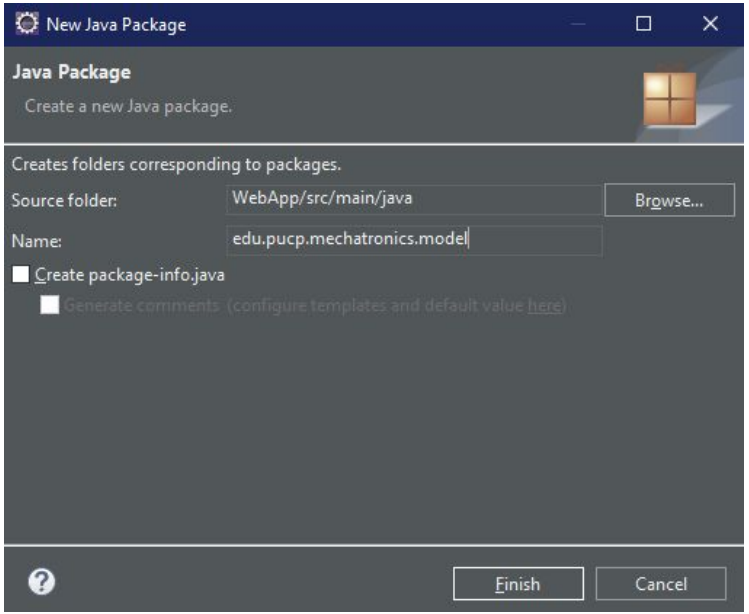
Para seguir un patrón MVC(Modelo-Vista-Controlador), utilizaremos paquetes para aislar las capas de nuestra aplicación.



Arquitectura en Capas (2/4)

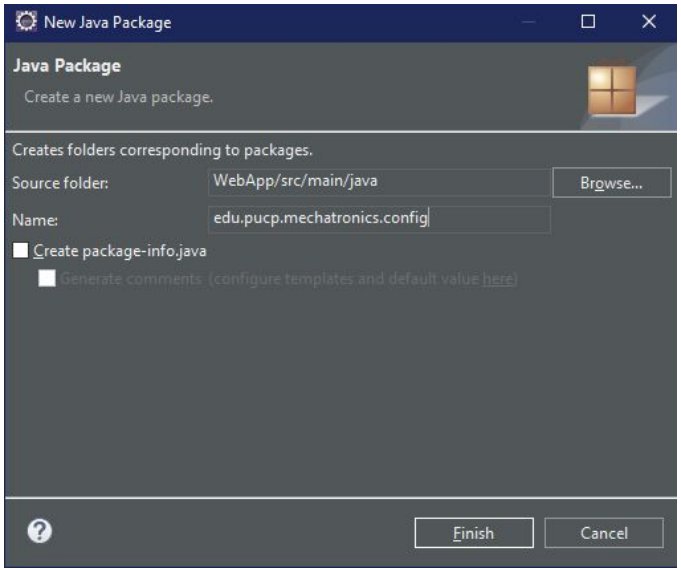
Para poder aplicar esto, en nuestro proyecto ya creado le daremos click derecho → new → paquete
El cual se llamara : edu.pucp.mechatronics.model

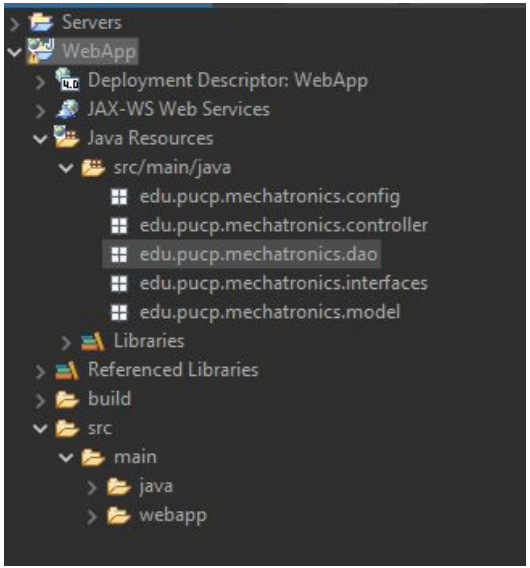




Arquitectura en Capas (3/4)

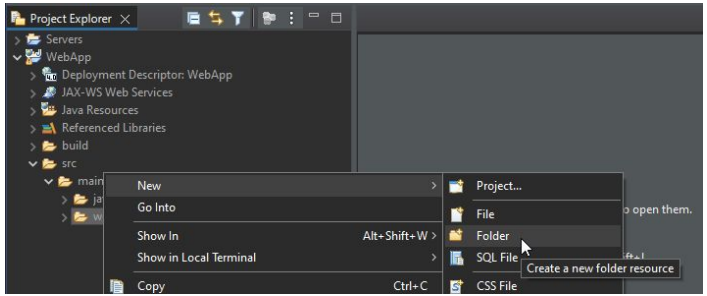
Una vez creado el nuevo paquete, crearemos otros nuevos paquetes añadiendo lo que buscamos luego del nombre como : **edu.pucp.mechatronics.model**
Los paquetes a crear seran: **controller,dao,interfaces y config**

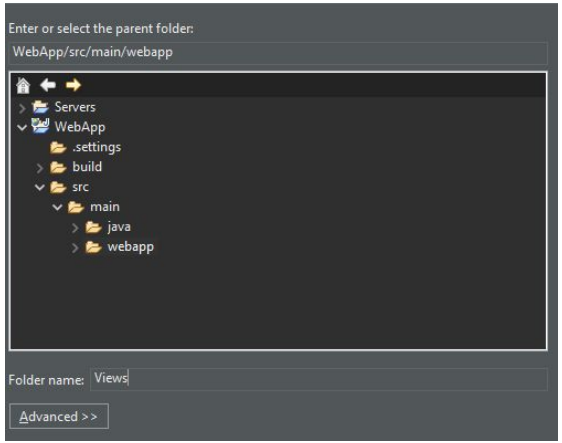


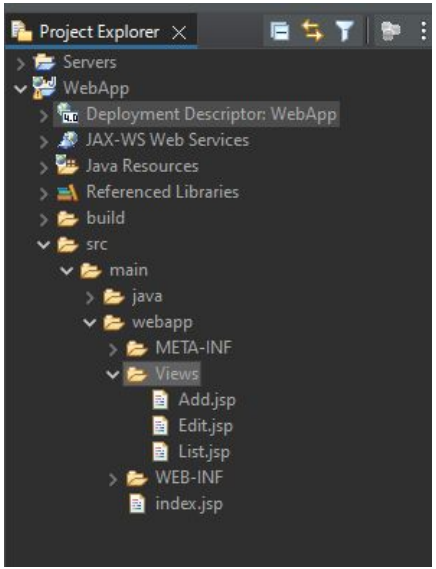


Arquitectura en Capas (4/4)

Nuestra Capa Vista , estará compuesta de archivos .jsp
Por lo cual dentro de nuestro WebApp Folder, crearemos un nuevo folder llamado views.
Dentro de este folder crearemos las vistas, add, edit y list

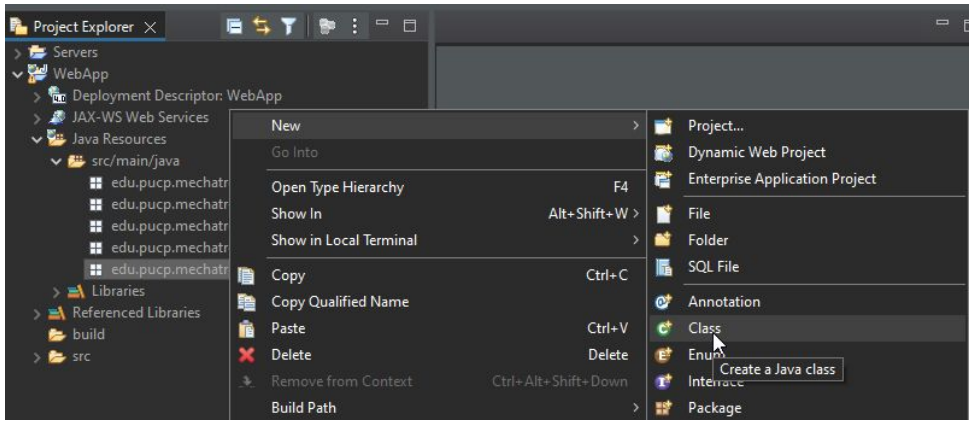


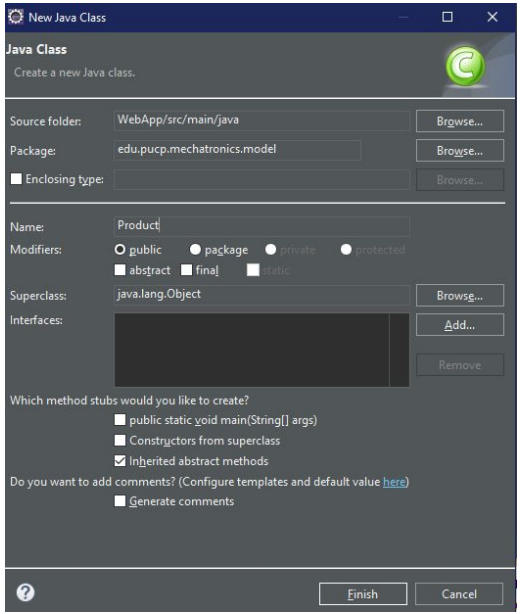




Paquete Model (1/4)

Aqui definiremos nuestra clase Producto, para eso crearemos una nueva clase dentro de nuestro paquete modelo , el cual se llamara Product

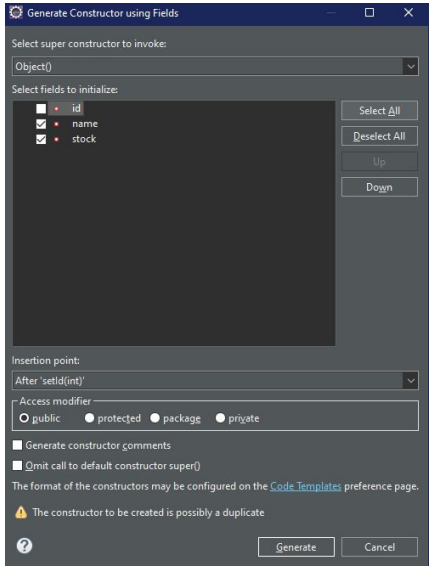




Paquete Model (2/4)

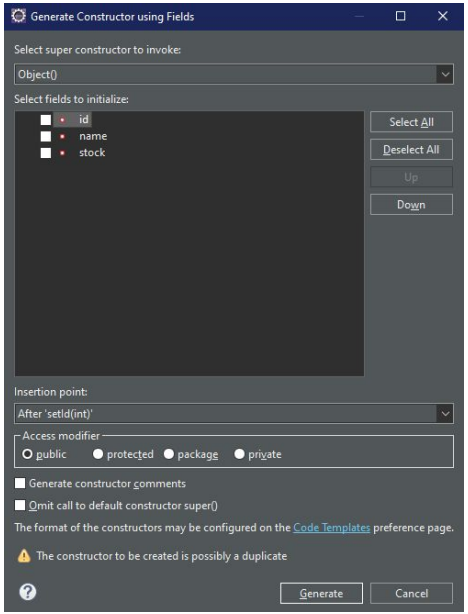
Una vez creado definiremos nuestra clase , la cual contendrá : id, name y stock
Una vez tengamos esto, podemos añadir los constructores dandole click derecho a nuestra clase → Source → Generate Constructor using Fields

```java
package edu.pucp.mechatronics.model;

public class Product {
    private int id;
    private String name;
    private int stock;
```



Paquete Model (3/4)

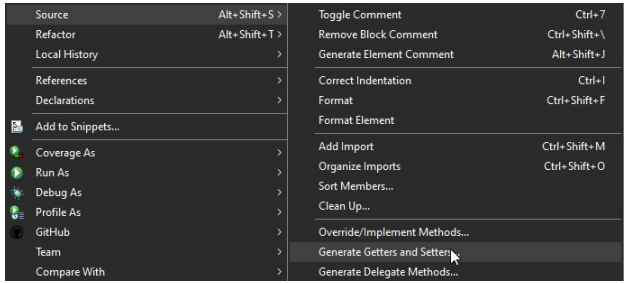
Los constructores que crearemos serán dos , un Constructor Producto que solo contenga Nombre y Stock , y otro que no contenga ninguno




```java
package edu.pucp.mechatronics.model;

public class Product {
    private int id;
    private String name;
    private int stock;

    public Product() {
    }

    public Product(String name, int stock) {
        this.name = name;
        this.stock = stock;
    }
```

Paquete Model (4/4)

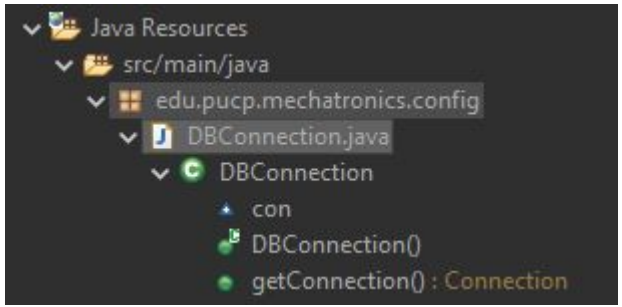
Finalmente Implementaremos los getters and setters de nuestra clase :




```java
package edu.pucp.mechatronics.model;

public class Product {
    private int id;
    private String name;
    private int stock;

    public Product() {
    }

    public Product(String name, int stock) {
        this.name = name;
        this.stock = stock;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getStock() {
        return stock;
    }

    public void setStock(int stock) {
        this.stock = stock;
    }

}
```

Paquete Config

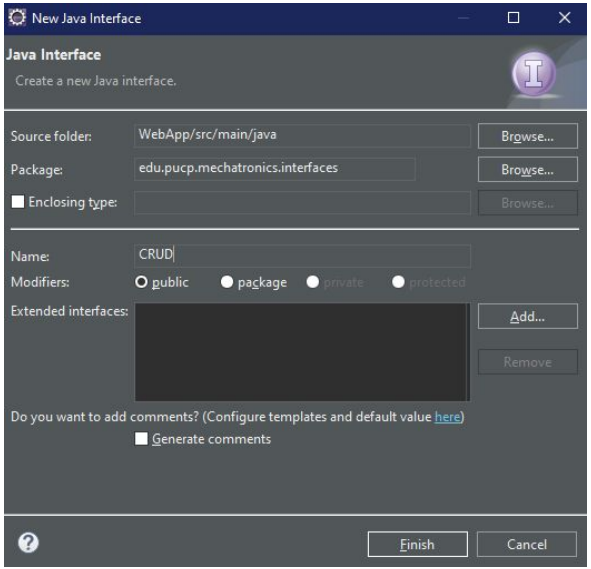
Este nos permite definir las conecciones con la base de datos asi como tambien su interacción con este.
Para ello generamos una clase llamada DBConnection y insertamos el siguiente código




```java
package edu.pucp.mechatronics.config;

import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnection {
    Connection con;
    public DBConnection() {
        try {
            Class.forName("com.mysql.jdbc.Driver");

            con = DriverManager.getConnection("jdbc:mysql://localhost:3306/lab01","root","pucp2023");
        }
        catch (Exception e) {
            System.err.println("Error:" +e);
        }
    }
    public Connection getConnection() {
        return con;
    }

}
```

Paquete Interface

Este paquete define las actividades CRUD, las cuales son funciones que deben ser implementadas para realizar procedimientos como buscar, guardar, updatear o eliminar. Para ello generaremos una Interfaz llamada CRUD e insertamos el siguiente código :




```java
package edu.pucp.mechatronics.interfaces;

import java.util.List;

public interface CRUD<E> {
    public List<E> findAll();
    public Product find(int id);
    public boolean save(Product p);
    public boolean update(Product p);
    public boolean delete(int id);

```

Paquete DAO

Este paquete define como los objetos son leídos y escritos dentro de la base de datos.
Para esto generamos una clase llamada ProductDao la cual implementara la interfaz CRUD y ingresamos el siguiente código :

```java
package edu.pucp.mechatronics.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.ArrayList;
import java.util.List;


import edu.pucp.mechatronics.config.DBConnection;
import edu.pucp.mechatronics.interfaces.CRUD;
import edu.pucp.mechatronics.model.Product;


public class ProductDAO implements CRUD<Product>{
    DBConnection connDB = new DBConnection();
    Product currentProduct = new Product(); //product for details, updates or deletion

    @Override
    public List<Product> findAll() {

        List<Product> products = new ArrayList<>();
        String sql = "select * from product;" ;
        try {
            Connection con = connDB.getConnection();
            PreparedStatement statement = con.prepareStatement(sql);
            ResultSet rs = statement.executeQuery();
            while(rs.next()) {
                Product prod = new Product();
                prod.setId(rs.getInt("id"));
                prod.setName(rs.getString("name"));
                prod.setStock(rs.getInt("stock"));
                products.add(prod);
            }

        } catch (Exception e) {
            System.err.println("Error: "+e.getMessage());
        }

        return products;
    }
```

```java
    @Override
    public Product find(int id) {
        String sql = "select * from product where id=?;" ;
        try {
            Connection con = connDB.getConnection();
            PreparedStatement statement = con.prepareStatement(sql);
            statement.setInt(1, id);
            ResultSet rs = statement.executeQuery();
            while(rs.next()) {
                currentProduct.setId(rs.getInt("id"));
                currentProduct.setName(rs.getString("name"));
                currentProduct.setStock(rs.getInt("stock"));
            }

        } catch (Exception e) {
            System.err.println("Error: "+e.getMessage());
        }

        return currentProduct;
    }

    @Override
    public boolean save(Product p) {
        String sql = "insert into product(name, stock) values(?,?)";
        try {
            Connection con = connDB.getConnection();
            PreparedStatement statement = con.prepareStatement(sql);
            statement.setString(1, p.getName());
            statement.setInt(2, p.getStock());
            statement.executeUpdate();


        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }


        return false;
    }
```

```java
    @Override
    public boolean update(Product p) {
        String sql = "update product set name=?,stock=? where id=?";
        try {
            Connection con = connDB.getConnection();
            PreparedStatement statement = con.prepareStatement(sql);
            statement.setString(1, p.getName());
            statement.setInt(2, p.getStock());
            statement.setInt(3, p.getId());
            statement.executeUpdate();
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
        return false;

    }

    @Override
    public boolean delete(int id) {
        String sql = "delete from product where id=?;";
        try {
            Connection con = connDB.getConnection();
            PreparedStatement statement = con.prepareStatement(sql);
            statement.setInt(1,  id);
            statement.executeUpdate();
        } catch (Exception e) {
            System.err.println("Error: " + e.getMessage());
        }
        return false;
    }

}
```

Paquete Controlador

Este paquete define la clase controlador quien es el encargado de tomar acciones mandadas atraves de las vistas como transiciones, modificaciones , etc .
Para ello crearemos la clase Controller y añadiremos el siguiente código :

```java
package edu.pucp.mechatronics.controller;

import java.io.IOException;
import java.util.List;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


import edu.pucp.mechatronics.model.Product;
import edu.pucp.mechatronics.dao.ProductDAO;

@WebServlet("/Controller")
public class Controller extends HttpServlet {
    private static final long serialVersionUID = 1L;

    String list = "views/list.jsp";
    String edit = "views/edit.jsp";
    String add = "views/add.jsp";
    ProductDAO dao = new ProductDAO();

    public Controller() {
        super();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String currentAcces = "";
        String action = request.getParameter("action"); //parameter from "index.jsp"
        if(action.equalsIgnoreCase("list")) {

            currentAcces = list;
        }
        else if(action.equalsIgnoreCase("goAdd")){ //from index.jsp
            currentAcces  = add;
        }
        else if(action.equalsIgnoreCase("Add")){ //from add.jsp, submit button
            String name = request.getParameter("txtName");
            int stock = Integer.parseInt(request.getParameter("txtStock"));
            Product prod = new Product(name, stock);
            dao.save(prod);
            currentAcces = list;

        }
        else if(action.equalsIgnoreCase("goEdit")) {
            request.setAttribute("idProd", request.getParameter("id"));
            currentAcces = edit;

        }
        else if(action.equalsIgnoreCase("Update")) {
            int id = Integer.parseInt(request.getParameter("txtId"));
            String name = request.getParameter("txtName");
            int stock = Integer.parseInt(request.getParameter("txtStock"));
            Product prod = new Product();
            prod.setId(id);
            prod.setName(name);
            prod.setStock(stock);
            dao.update(prod);
            currentAcces = list;

        }
        else if (action.equalsIgnoreCase("Delete")) {
            int id = Integer.parseInt(request.getParameter("id"));
            dao.delete(id);
            currentAcces= list;

        }
        List<Product> products = dao.findAll();
        request.setAttribute("products", products); //link object for jsp

        RequestDispatcher view = request.getRequestDispatcher(currentAcces);
        view.forward(request, response);
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

Vistas

Este apartado contiene todos los archivos JSP que permitirán ver y hacer update de los distintos objetos, este tipo de archivo permite que código java pueda ser insertado dentro de un código HTML , lo cual permite el uso de objetos variables y métodos.

```jsp
<%@page import="java.util.Enumeration"%>
<%@page import="edu.pucp.mechatronics.model.Product"%>
<%@page import="edu.pucp.mechatronics.dao.ProductDAO"%>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <div>
        <h1>Edit Product Information</h1>
        <%
            ProductDAO dao = new ProductDAO();
            int id = Integer.parseInt((String)request.getAttribute("idProd"));
            Product p = dao.find(id);
        %>
        <form action="Controller">
            Product:<br>
            <input type="text" name="txtName" value="<%= p.getName()%>"><br>
            Stock:<br>
            <input type="text" name="txtStock" value="<%= p.getStock()%>"><br>
            <input type="hidden" name="txtId" value="<%= p.getId()%>">
            <input type="submit" name="action" value="Update"><br>

        </form>
    </div>
</body>
</html>
```

```jsp
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <div>
        <h1>Products</h1>
        <a href="Controller?action=goAdd">Create a new product</a>
        <table border="1">
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Product</th>
                    <th>Stock</th>
                </tr>
            </thead>
            <tbody>
            <c:forEach var="element" items="${products}">
                <tr>
                    <td>${element.id}</td>
                    <td>${element.name}</td>
                    <td>${element.stock}</td>
                    <td>
                        <a href="Controller?action=goEdit&id=${element.id}">Edit</a>
                        <a href="Controller?action=Delete&id=${element.id}">Delete</a>
                    </td>
                </tr>
            </c:forEach>
            </tbody>

        </table>
    </div>
</body>
</html>
```

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <div>
        <h1>New Product</h1>
        <form>
            Product:<br>
            <input type="text" name="txtName"><br>
            Stock:<br>
            <input type="text" name="txtStock"><br>
            <input type="submit" name="action" value="Add"><br>
        </form>
    </div>

</body>
</html>
```

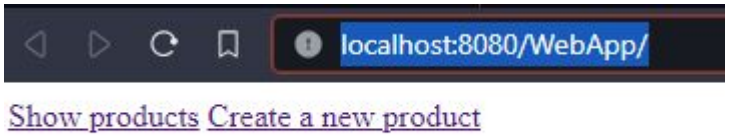
Index

Asi mismo , es necesario modificar el index , para que tengamos un punto de acceso en nuestra página

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

    <div>
        <a href="Controller?action=list">Show products</a>
        <a href="Controller?action=goAdd">Create a new product</a>
    </div>

</body>
</html>
```

Correr nuestro proyecto

Para correr nuestro proyecto , es necesario que le demos click derecho → Run A → Run on Server
Aquí seleccionaremos nuestro servidor Tomcat 9.0 y le daremos en Finish
Nuestro Navegador nos dirigirá al url: “localhost:8080/WebServer/”



En caso de que no se reconozca el driver en la ejecución

En caso de que esto suceda deberemos dar click derecho en nuestro proyecto → Run As→ Run Configurations
Aquí seleccionaremos nuestro servidor TomCat y le daremos click on Classpath→ Bootstrap Entries → Add External Jars
Aquí ingresamos nuestro connector Mysql en formato jar y le daremos en Apply