# Instalación de componentes

## Instalación de componentes.

Para generar de manera practica una solución web con java utilizaremos el ambiente de desarrollo NetBeans en su versión 12.0 ya que es una de las ultimas actualizaciones más estables a la fecha; esta versión nos permite incorporar y desplegar soluciones web básicas y con componentes complejos.

Este ambiente de desarrollo nos permite desplegar nuestras soluciones en algunos servidores conocidos tal es el caso de Apache Tomcat en su versión 10.0.7, que es uno de los servidores de despliegue más nombrados entre desarrolladores.

Del mismo modo utilizaremos a MySQL como sistema gestor de bases de datos ya con el almacenaremos de forma sistemática la información utilizada por el sistema que se estará definiendo en ejercicios posteriores y junto con este utilizaremos un conector JDBC que nos permitirá la fluidez de la información con código Java que estaremos proporcionando.

Otros elementos importantes son los frameworks como Struts que nos permitirá controlar la información desde las vistas del sistema hasta él envió de la información a la base de datos.

Estos son algunos componentes básicos que un Sistema Web utiliza para ser desarrollado de manera correcta.

## Instalación del ambiente de desarrollo NetBeans 12.0 apache

Para iniciar con la instalación de NetBeans 12.0 requerimos ir a la siguiente dirección: <https://netbeans.apache.org/download/index.html> lo cual nos brindara las opciones generales de descarga, para el cual utilizaremos la última versión estable del IDE denominado **Apache NetBeans 12 LTS (NB 12.0).**

Para continuar con la instalación utilizaremos el instalador que nos arroja la dirección web antes mencionada.

Al abrir el archivo mostrará la imagen 1.0.1 que nos especificará que es lo que se instalará en nuestro equipo de cómputo, para continuar dará clic en el botón Next/Siguiente.

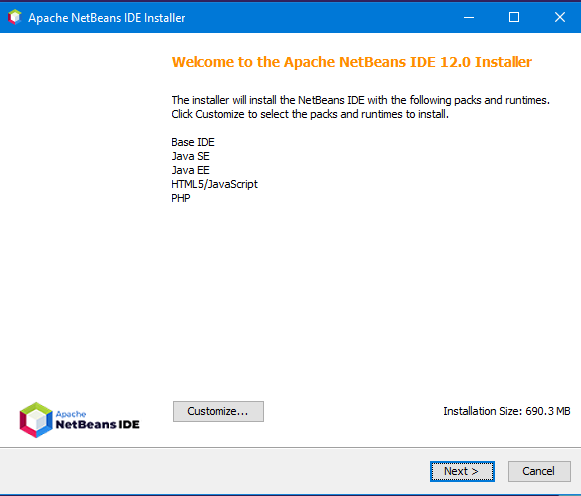


Imagen 1.0.1 Instalación de NetBeans 12.0

Para continuar con la instalación se mostrará la imagen 1.0.2 donde se debe considerar los términos y condiciones aceptando y dando clic en Next/Siguiente.

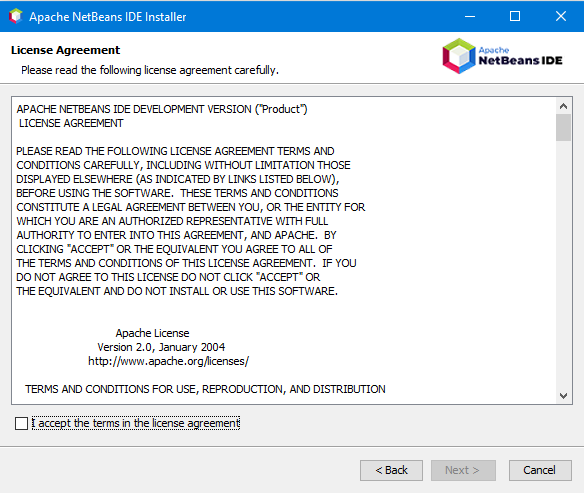


Imagen 1.0.2 Aceptación de términos y condiciones de Instalación de NetBeans 12.0.

Para finalizar con la instalación básica del ambiente de desarrollo de NetBeans 12.0 se debe hacer clic en Install/Instalar, en este punto se instalarán los componentes básicos como se muestra en la imagen 1.0.3.

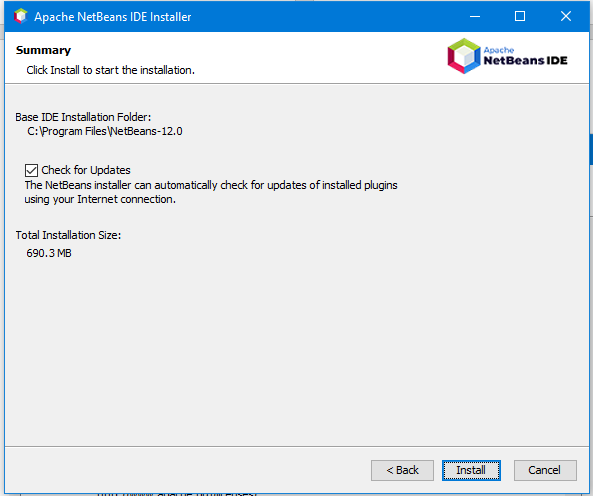


Imagen 1.0.3 Instalación de NetBeans 12.0.

Al terminar la instalación si no hay ningún problema mostrará la imagen 1.0.4, esto nos confirmará que ya podemos iniciar nuestro IDE NetBeans 12.0 de manera correcta.

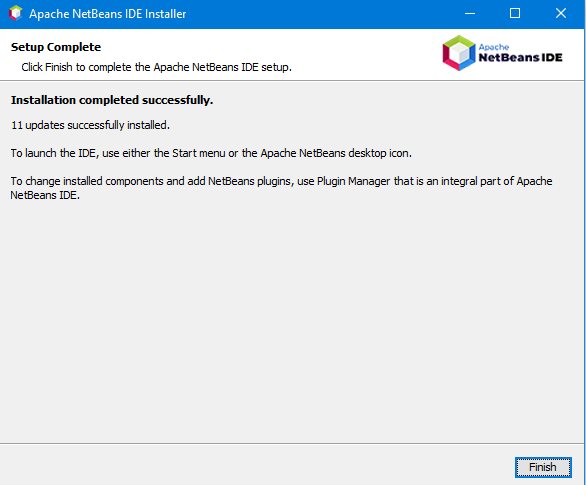


Imagen 1.0.4 Finalización de la Instalación de NetBeans 12.0.

## Instalación del MySQL server

Para realizar una correcta instalación de MySQL quien será nuestro motor de base de datos, primeramente, se requiere ir a la siguiente dirección web: <https://dev.mysql.com/downloads/mysql/> y seleccionar nuestra plataforma en base a nuestro sistema operativo, por ejemplo, utilizaremos la versión del sistema operativo Windows para ello tendremos que seleccionar Microsoft Windows como se muestra en la imagen 1.1.0, posteriormente requerimos dar clic en *Go to Download Page*, utilizaremos la versión que nos muestra la imagen 1.1.0.1 donde nos descargará un archivo ejecutable; para iniciar con la instalación de los componentes básicos que estaremos utilizando.

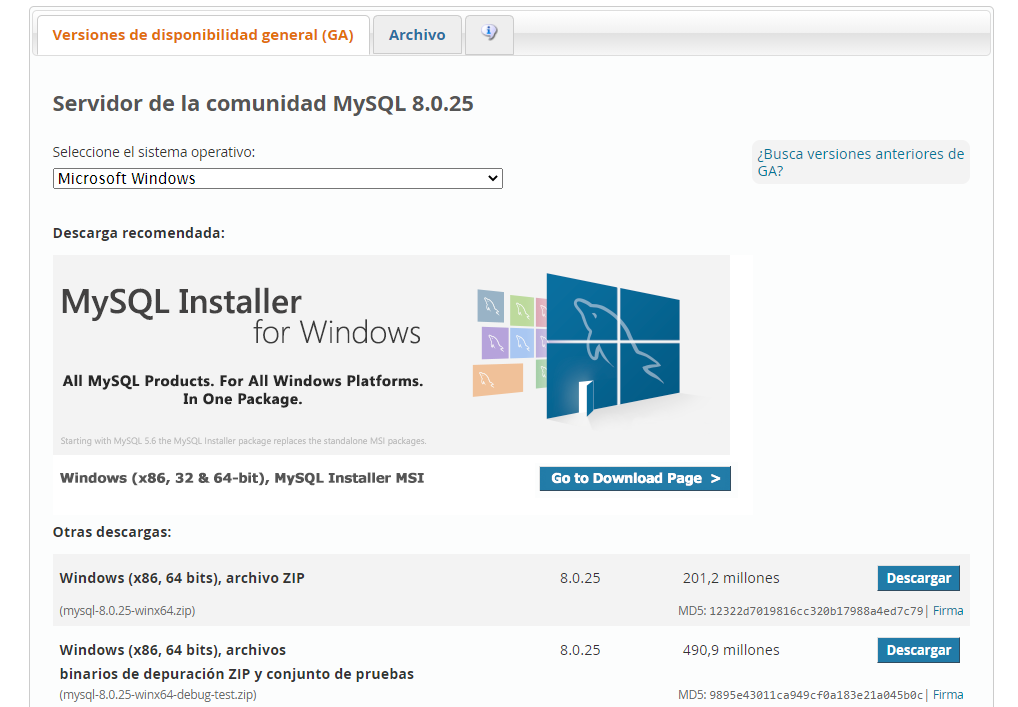


Imagen 1.1.0 Pagina de descarga de componentes MySQL.



Imagen 1.1.0.1 Versión de descarga de MySQL.

Cuando iniciemos el instalador generado por la descripción anterior iniciara el programa como la imagen 1.1.1 en donde nos permitirá agregar a nuestro equipo de computo el servidor de base de datos, el espacio de trabajo grafico denominado en el instalador como MySQL Workbench el cual será de gran apoyo para generar gráficamente nuestro modelo de base de datos, asi como llevar de manera mas sencilla la administración de las múltiples bases de datos que sean requeridas en nuestro sistema que será propuesto en bloques posteriores a este.

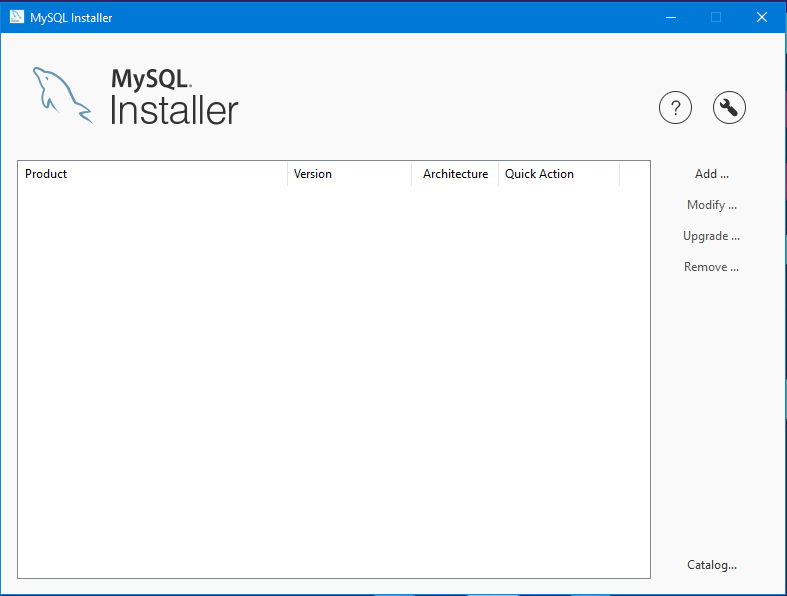


Imagen 1.1.1 Instalación de componentes MySQL.

Para agregar componentes en la imagen 1.1.1 tendremos que dar clic en Add/Agregar para que nos solicite el tipo de instalación, para una instalación básica de componentes seleccionaremos la opción **Developer Default** y dar Clic en Next/Siguiente para continuar con la instalación.

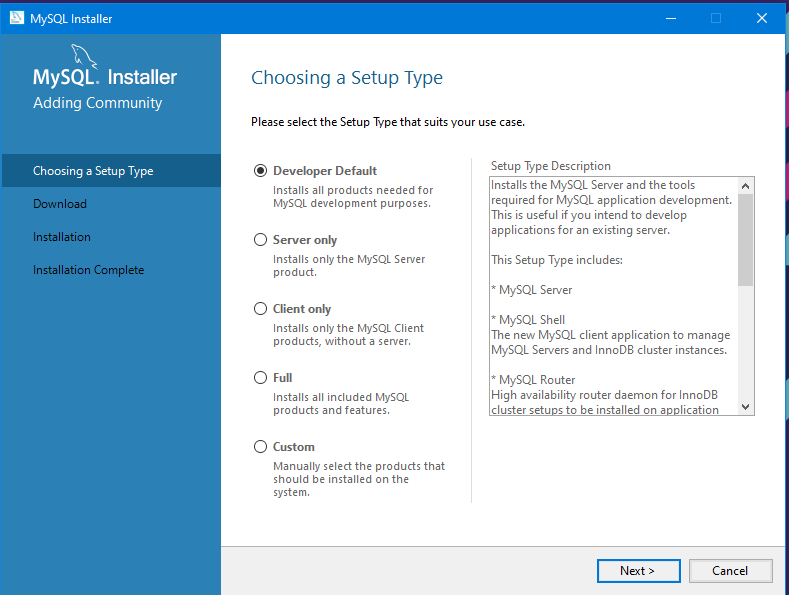


Imagen 1.1.2 Selección del tipo de instalación de MySQL.

En este apartado seleccionaremos la ruta de instalación de nuestro equipo del servidor MySQL y MySQL Workbench, para continuar daremos clic en Next/Siguiente como se muestra en la imagen 1.1.3.

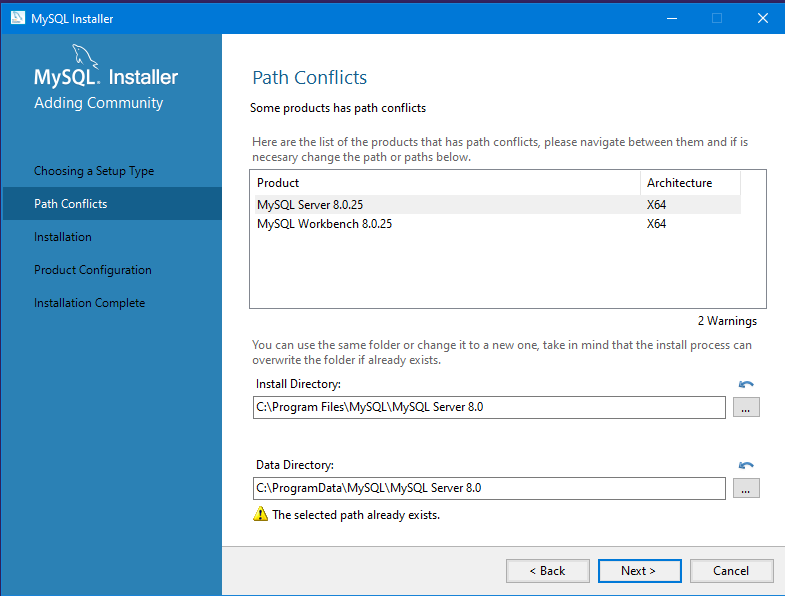


Imagen 1.1.3 Selección de ruta de instalación.

En esta parte daremos clic en Execute/Ejecutar para instalar los componentes, además en esta parte nos brindara conectores para el desarrollo como .Net u ODBC para la unión entre sistemas y bases de datos como se muestra en la imagen 1.1.4.

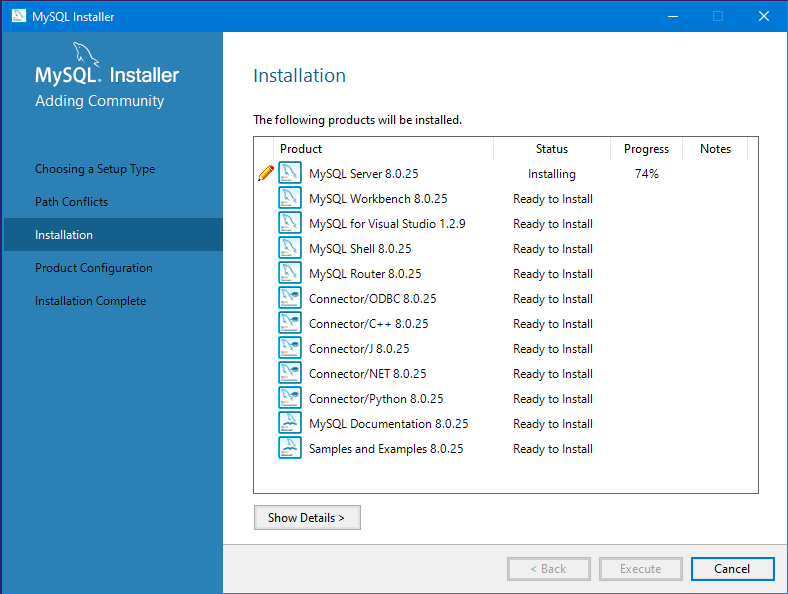


Imagen 1.1.4 Instalación final de componentes.

En la imagen 1.1.5 nos muestra que la instalación se ha completado con normalidad para iniciar la configuración de nuestro servidor.

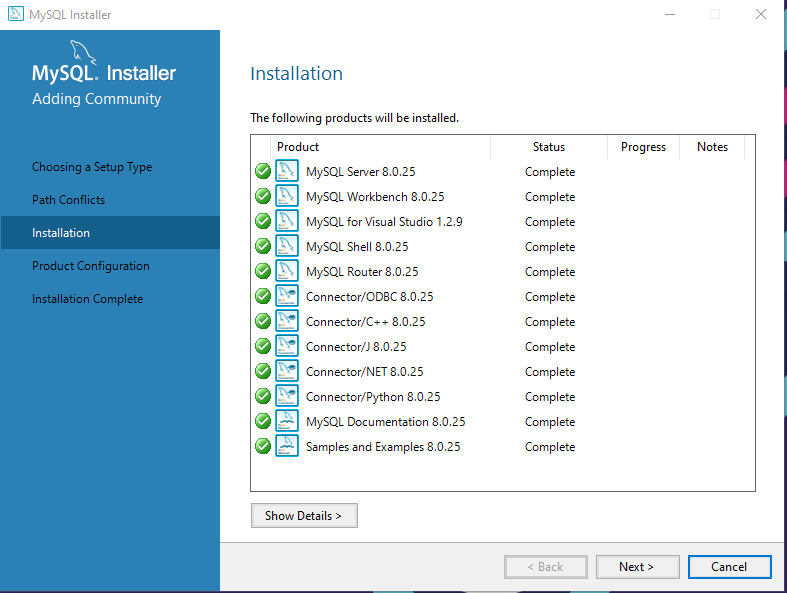


Imagen 1.1.5 Instalación de componentes finalizado.

En este apardado podemos configurar el puerto por la cual el servidor nos conectara con las distintas bases de datos, tambien nos permitira nombrar esa conexión, la recomendación si no hay inconvenientes es dejar los valores por default brindados por el instalador como se muesta en la imagen 1.1.6.

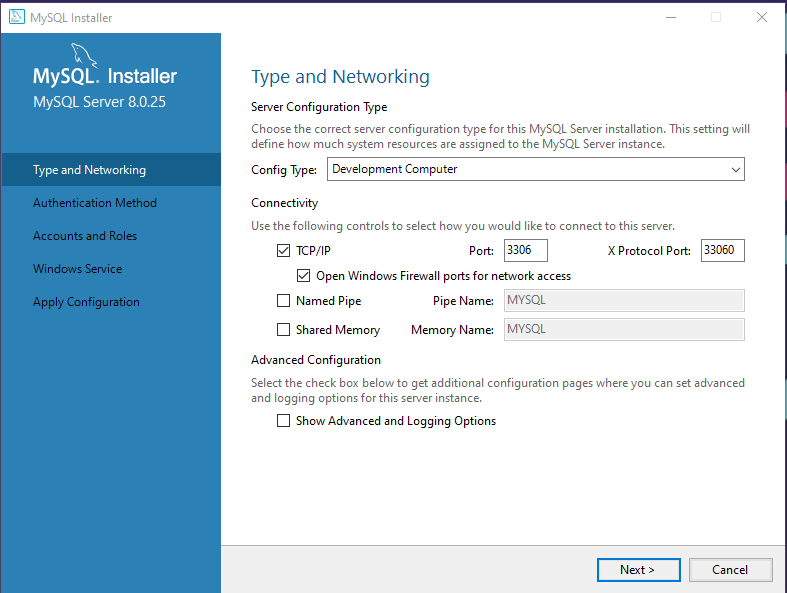


Imagen 1.1.6 Configuración de puertos de red.

En el siguiente aparado podemos configurar los métodos de autenticación, por default el instalador nos permite tener una encriptación fuerte, la cual es recomendada para la seguridad de las bases de datos como se muestra en la imagen 1.1.7, al terminar daremos clic en **Next/Siguiente** para ingresar la contraseña del usuario ROOT que configura el instalador, para ello en la imagen 1.1.8 ingresaremos la contraseña ya que no viene una predeterminada y al dar clic en el botón de **Check** nos verificara si nuestra contraseña cumple con los requisitos mínimos.

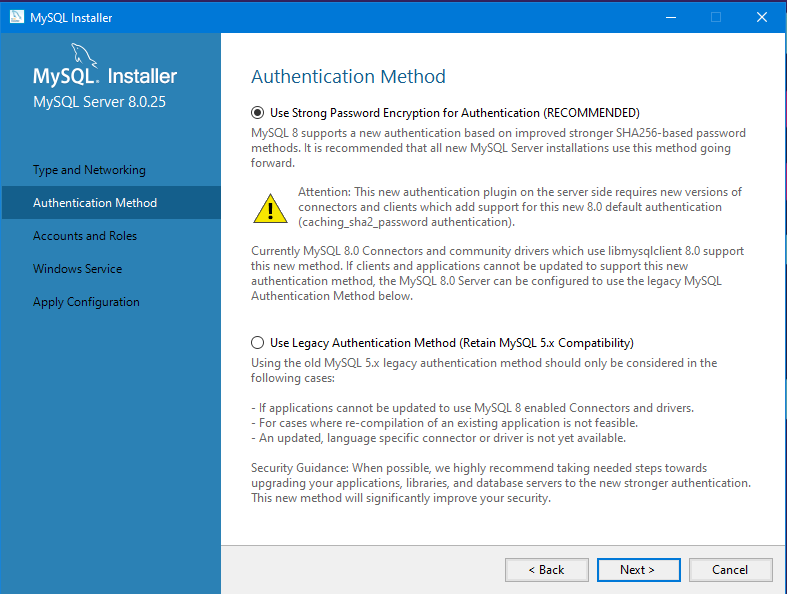


Imagen 1.1.7 Selección de tipo de autenticación.

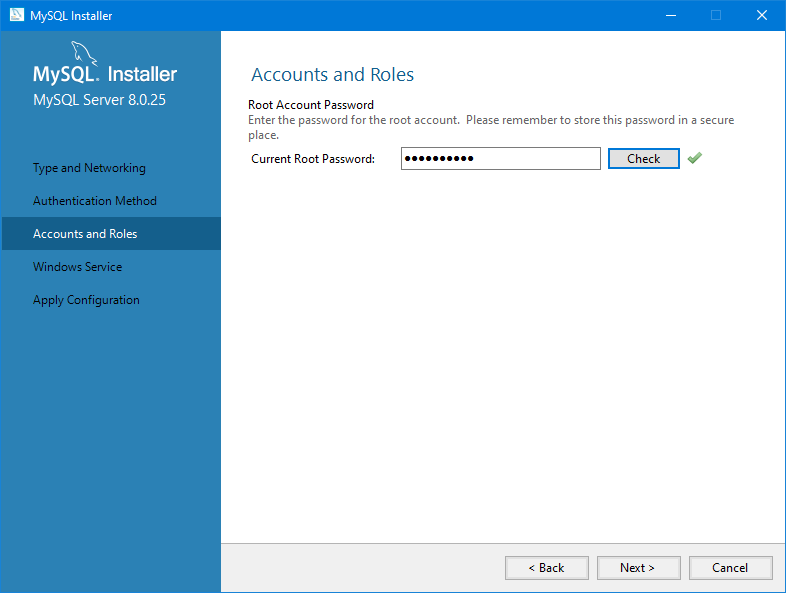


Imagen 1.1.8 Ingreso y verificación de contraseña del usuario ROOT.

En la siguiente imagen 1.1.9 nos permitirá especificar el nombre de nuestro servicio y levantar el servidor, para continuar con la verificación y escritura de la configuración que hemos estado realizando en anteriores explicaciones, para terminar en la imagen 1.1.10

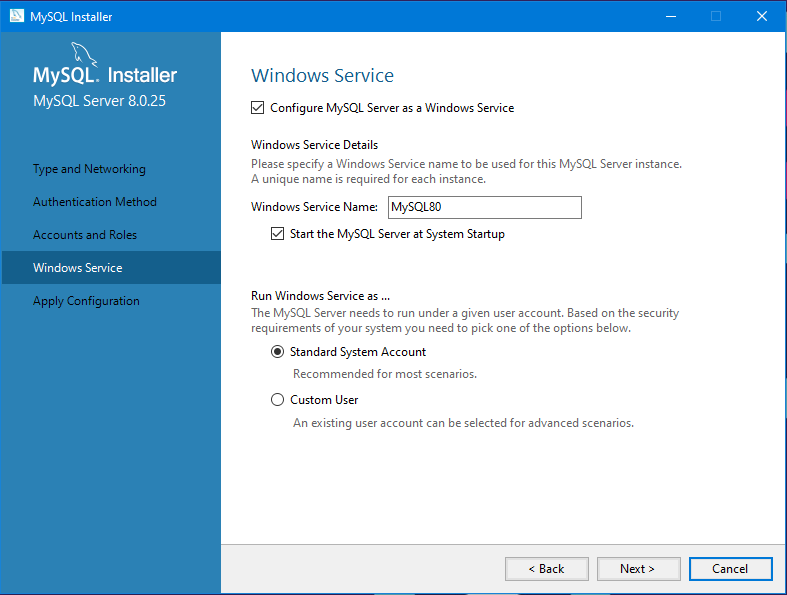


Imagen 1.1.9 Nombramiento del servicio.

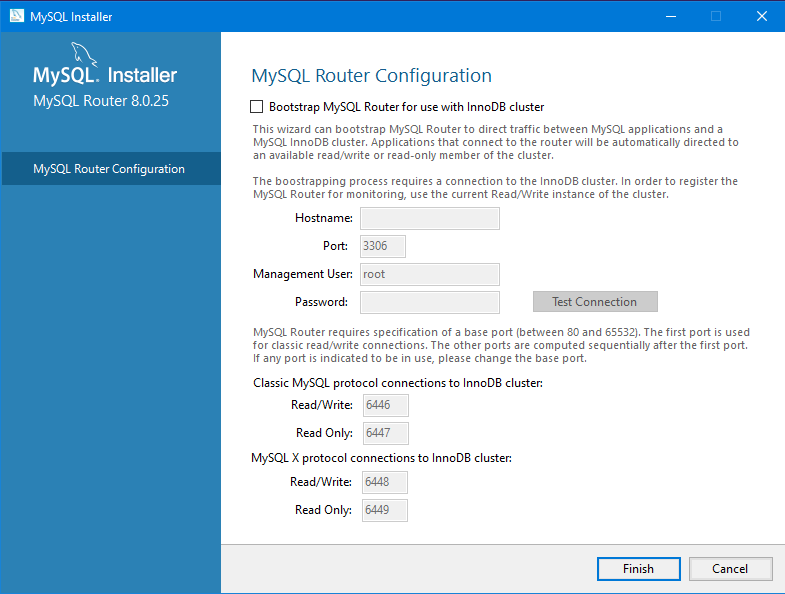


Imagen 1.1.10 Resumen de la configuración

Para finalizar con la instalación levantaremos el servicio ingresando las credenciales que anteriormente configuramos y dando clic en Check conectaremos el servidor, tal como se muestra en la imagen 1.1.11; para finalizar damos clic en botón de Finalizado para terminar y cerrar el instalador en las imágenes 1.1.12 y la imagen 1.1.13.

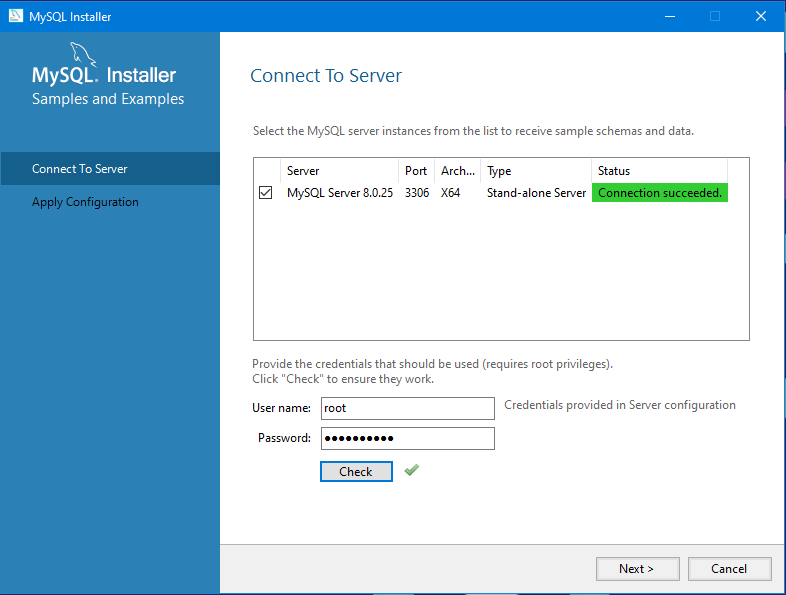


Imagen 1.1.11 Levantamiento del servidor.

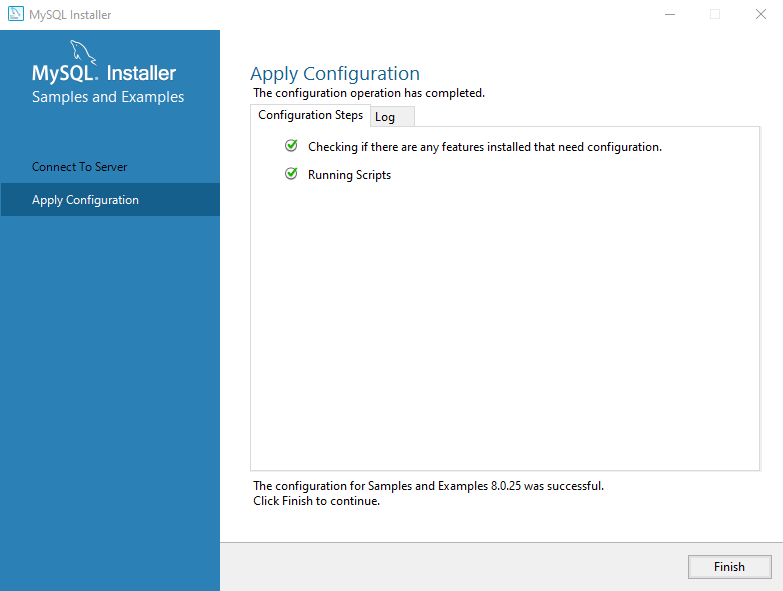


Imagen 1.1.12 Levantamiento del servidor.

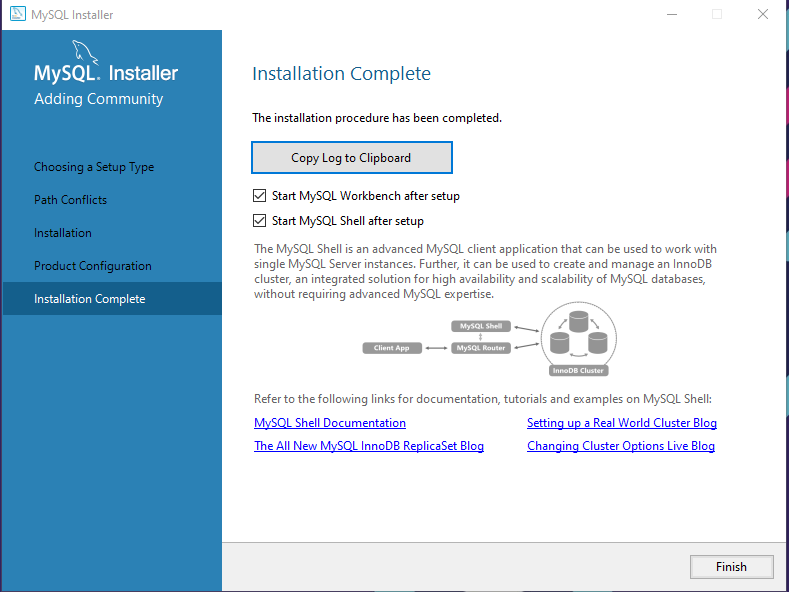


Imagen 1.1.13 Levantamiento del servidor.

## Instalación del servidor de páginas Apache Tomcat 10.

Para desplegar proyectos web regularmente se utilizan servidores instalados o dedicados a administrar los diversos proyectos web que nos permiten ver los sitios en formato HTML dentro de cualquier navegador web como Google Chrome, Internet Explorer, Mozilla Firefox, entre otros más.

Dentro del desarrollo que estaremos dedicando en bloques posteriores requeriremos desplegar nuestro proyecto web en algún navegador y asi verificar el correcto funcionamiento de este, utilizaremos Apache Tomcat en su versión mas estable 10.0, para ello nos a la siguiente dirección web: <https://tomcat.apache.org/download-10.cgi> como nos muestra la imagen 1.2.0.

Descargaremos la opción ZIP y lo colocaremos en nuestro equipo de computo preferentemente en la raíz de nuestro equipo como se muestra en la imagen 1.2.1 para localizarlo fácilmente a la hora de incorporarlo con nuestro IDE o entorno de desarrollo.

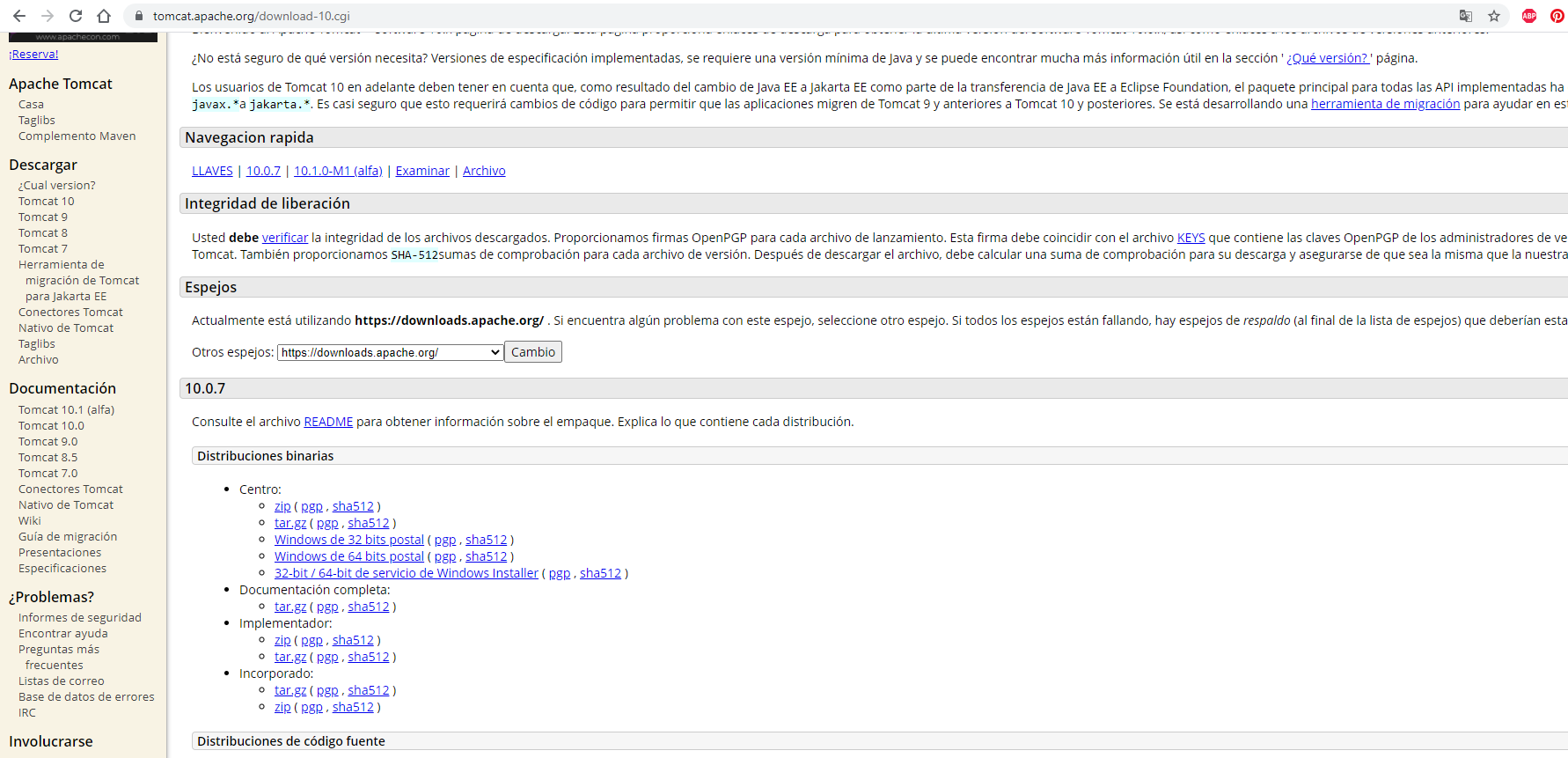


Imagen 1.2.0 Descarga de los archivos ZIP.

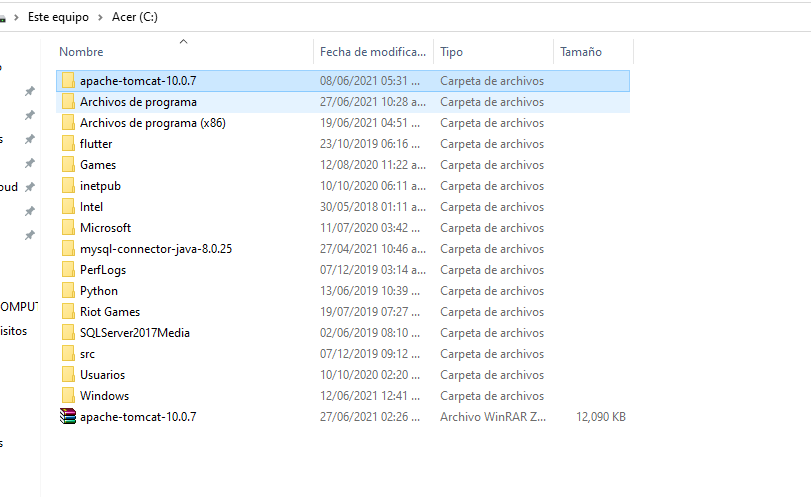


Imagen 1.2.1 Descarga y descompresión de archivos ZIP de Apache Tomcat.

## 2.0 Diseño de la base de datos relacional básico de un sistema web de ventas.

Cuando se desarrolla un sistema web de tipo ventas es de manera prioritario tener un reposito de datos para el cual mantener y genera un flujo de información que va desde generar una venta, hasta consultar productos, lo cual hace interactiva y dinámico cualquier sistema web de este tipo, en este caso consideraremos 3 entidades principales para generar una venta dentro del sistema web las cuales son: Usuarios, Producto y Ventas. Estas entidades nos permitirán expandir las necesidades de los usuarios desde lo más básico hasta algo más complejo como la administración de direcciones o ventas; categorizando cada una de ellas para que la información fluya de manera más comprensiva para el desarrollador que está llevando a cabo la parte dinámica a la hora de generar flujos completos de información bajo un patrón de diseño como el MVC que en bloques posteriores mencionaremos de manera básica la utilización, desarrollo y despliegue de cada elemento que conforma este patrón.

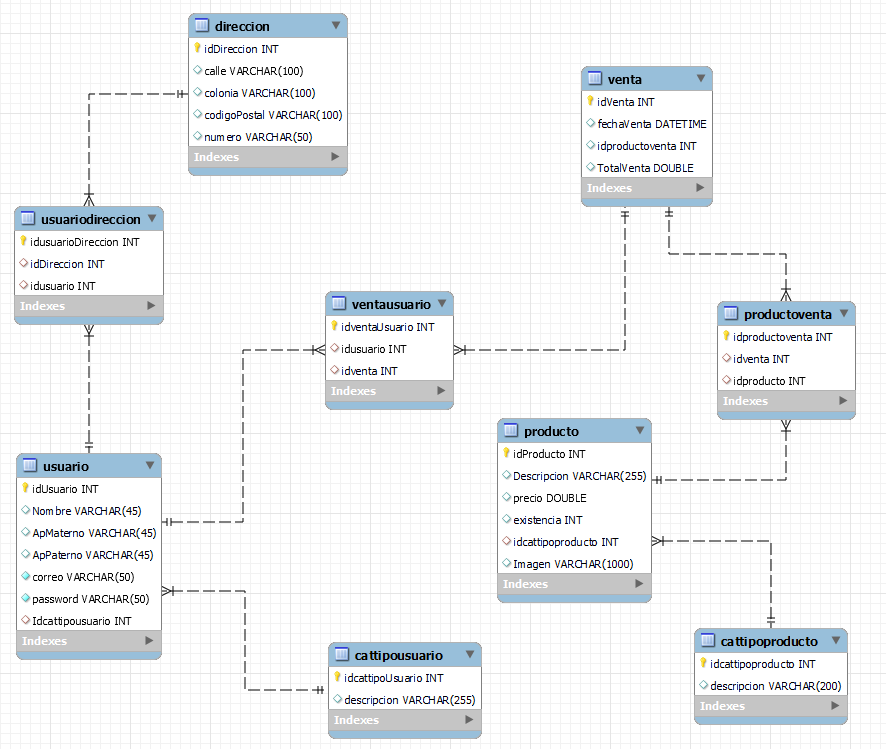


Imagen 1.2.2 Modelo Relacional

Podemos generar el modelo con la siguiente información de un Diccionario de datos desglosado en cada tabla que utilizaremos en nuestro modelo Relacional que observamos en la imagen 1.2.2.

En el sistema web se utilizarán catálogos como el catálogo de productos (Tabla 1.0.1) que nos permitiría categorizar distintos tipos de productos por su descripción, además del catálogo de usuarios (Tabla 1.0.2) que nos permitirá gestionar distintos tipos de usuarios y roles en el sistema web como gestionar productos en nuestro sistema de ventas.

Tabla 1.0.1 Cattipoproducto.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nombre** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idcattipoproducto | int | X |  | identificador |
| descripcion | Varchar (255) |  |  | Descripcion corta |

Tabla 1.0.2 Cattipousuario.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idcattipoUsuario | int | X |  | identificador |
| descripcion | Varchar (255) |  |  | Descripcion corta |

Para casi cualquier sistema web del tipo de ventas se requiere gestionar el acceso al sistema para lo cual tenemos nuestra entidad Usuario que modelado en tabla obtendremos las credenciales del usuario que utilizará el sistema web, como se muestra en la tabla 1.0.3.

Tabla 1.0.3 Usuario

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| id Usuario | int | X |  | identificador |
| Nombre | Varchar (50) |  |  | Nombre del usuario. |
| Materno | Varchar (50) |  |  | Apellido materno del usuario. |
| Paterno | Varchar (50) |  |  | Apellido Paterno del usuario. |
| correo | Varchar (50) |  |  | Correo del usuario. |
| password | Varchar (50) |  |  | Password del del usuario. |
| IdcattipoUsuario | int |  | X | Identificador tabla cattipousuario |

Dentro de nuestra definición en el modelo relacional utilizaremos distintas direcciones asi que para ello requerimos la table de Dirección establecida en la tabla 1.0.4 y por definición una tabla de UsuarioDireccion en la tabla 1.0.5.

Tabla 1.0.4 Direccion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idDireccion | int | X |  | identificador |
| calle | Varchar (100) |  |  | Nombre de la calle |
| colonia | Varchar (100) |  |  | Nombre de la colonia |
| codigoPostal | Varchar (100) |  |  | Identificador postal |
| numero | Varchar (100) |  |  | Numero de domicilio |

Tabla 1.0.5 UsuarioDireccion.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idusuarioDireccion | int | X |  | identificador |
| idDireccion | int |  | X | Identificador tabla dirección |
| idusuario | int |  | X | Identificador tabla usuario |

Para que un usuario pueda generar múltiples ventas se utilizaron las tablas ventausuario y venta para gestionar lo que cada usuario pueda generar a partir de la selección de productos; la tabla ventausuario establecida en la tabla 1.0.6 y la tabla venta generada en la tabla 1.0.7.

Nos permitirá establecer un flujo de venta entre usuario y las ventas que este pueda generar.

Tabla 1.0.6 VentaUsuario.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idventausuario | int | X |  | identificador |
| idusuario | int |  | X | Identificador tabla usuario |
| idventa | int |  | X | Identificador tabla venta |

Tabla 1.0.7 Venta.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idVenta | int | X |  | identificador |
| fechaVenta | datetime |  |  | Fecha de creación de la venta |
| idproductoventa | int |  | X | Identificador tabla productoventa |
| TotalVenta | double |  |  | Precio acumulado de los productos |

Las ventas en los sistemas web vienen acompañados de la obtención de productos que son entidades con variaciones por definición ya que esta entidad cuenta con actualizaciones seguidas muy comunes en el desarrollo web a continuación mostramos las tablas de productos, al mismo tiempo como se muestra en la tabla 1.0.8. También por definición en las ventas presentemos múltiples productos en las ventas, para ello utilizamos la relación en una tabla denominada productoventa establecida en la tabla 1.0.9.

Tabla 1.0.8 Producto.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idProducto | int | X |  | identificador |
| Descripcion | Varchar (100) |  |  | Descripcion corta |
| precio | double |  |  | Precio del Producto |
| existencia | int |  |  | Numero de Productos |
| idcattipoproducto | int |  | X | Identificador tabla cattipoproducto |
| Imagen | Varchar (1000) |  |  | Imagen del producto |

Tabla 1.0.9 ProductoVenta.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripción** |
| idproductoventa | int | X |  | identificador |
| idventa | int |  | X | Identificador tabla venta |
| idproducto | int |  | X | Identificador tabla producto |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Columna** | **Tipo de dato** | **PK** | **FK** | **Descripcion** |
| idwishlist | int | X |  | identificador |
| Descripcion | Varchar (255) |  |  | Descripción de la lista |
| idProducto | int |  | X | Identificador tabla producto |
| idusuario | int |  | X | Identificador tabla producto |

## 3.0 Integración del servidor Tomcat 10 con NetBeans 12.0

Para instalar ciertos componentes en un IDE completo como NetBeans es necesario seguir ciertas características, ya que muchas veces en tipo de desarrollo la integración con servidores locales para desplegar un proyecto web, es necesario seguir ciertos pasos con características que son propias del IDE NetBeans, en este apartado integraremos estos dos componentes para hacer un primer despliegue de un proyecto llamado *TestWeb* creado desde NetBeans y se mostrara en nuestro navegador local sobre un servidor web Tomcat 10.

Para ello requerimos abrir nuestro IDE NetBeans, para levantar un servidor Tomcat requerimos ir al apartado Tools/Herramientas dando clic en Servers/Servidores como se muestra en nuestra Imagen 1.2.3.

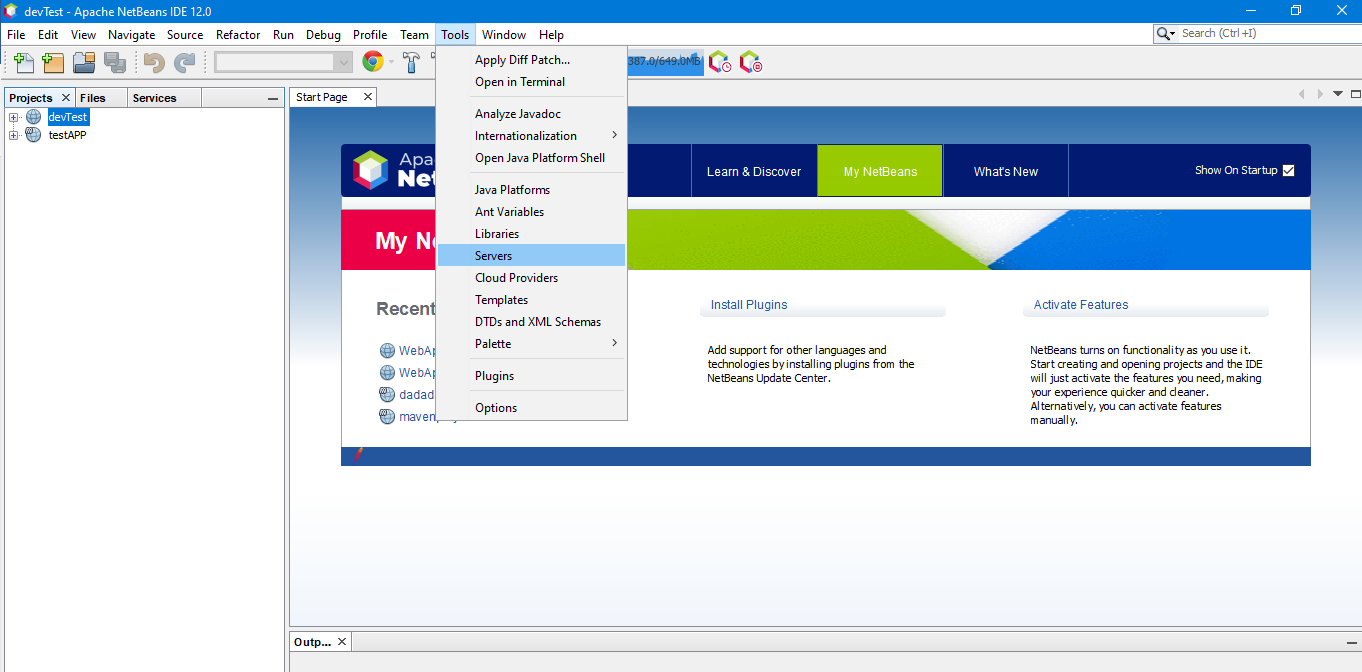


Imagen 1.2.3 Levantamiento de Servidor web Tomcat.

A continuación, nos muestra la imagen 1.2.4 donde se encuentran la configuración necesaria para la integración y levantamiento del servidor. Para agregar un servidor damos clic en Add Server/Agregar servidor.

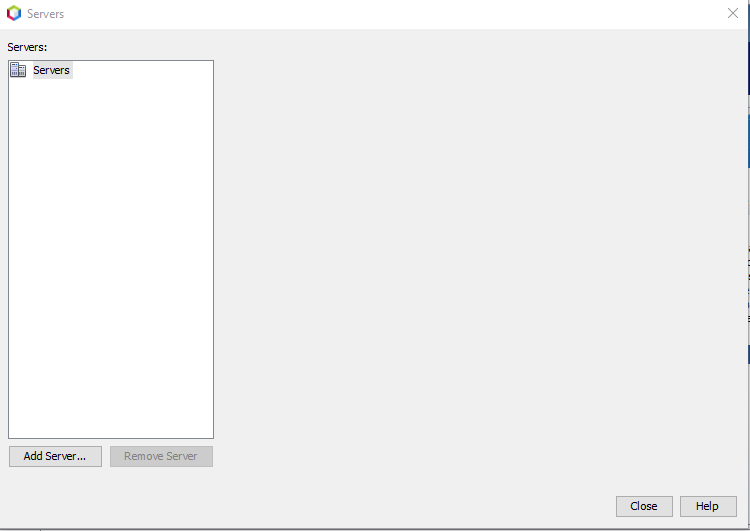


Imagen 1.2.4 Agregar servidor.

Este paso nos mostrará los servidores web que el IDE NetBeans soporta para esta versión; para continuar con la integración seleccionamos el servidor *Apache Tomcat or TomEE* y damos clic en siguiente tal como se muestra en la imagen 1.2.5.

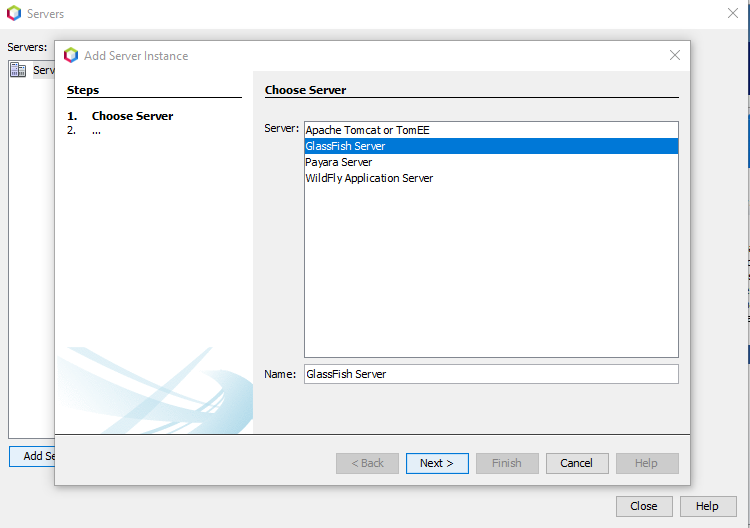


Imagen 1.2.5 Selección de servidor nuevo.

Como anteriormente mostramos en el apartado 1.2 utilizaremos el archivo ZIP que descargamos anteriormente y extraemos en nuestro equipo de cómputo, para lo cual en la opción *Server Location* damos clic en el botón *Browse* y seleccionamos la ruta de nuestro archivo extraído como se muestra en la imagen 1.2.6.

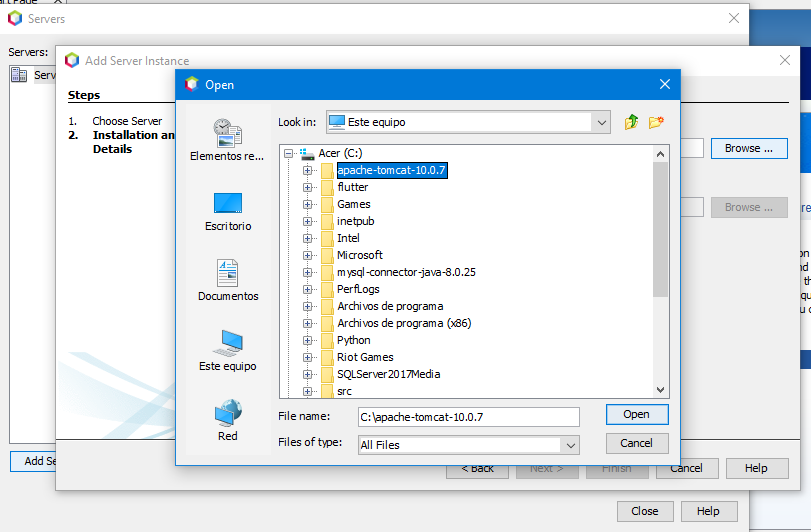


Imagen 1.2.6 Selección de archivos propios del servidor.

A continuación, ingresamos las credenciales iniciales para nuestro servidor web, tenga presente que estas se utilizaran al levantar el servicio en apartados posteriores; al ingresar las credenciales damos clic al botón *Finish* como se muestra en la imagen 1.2.7*.*

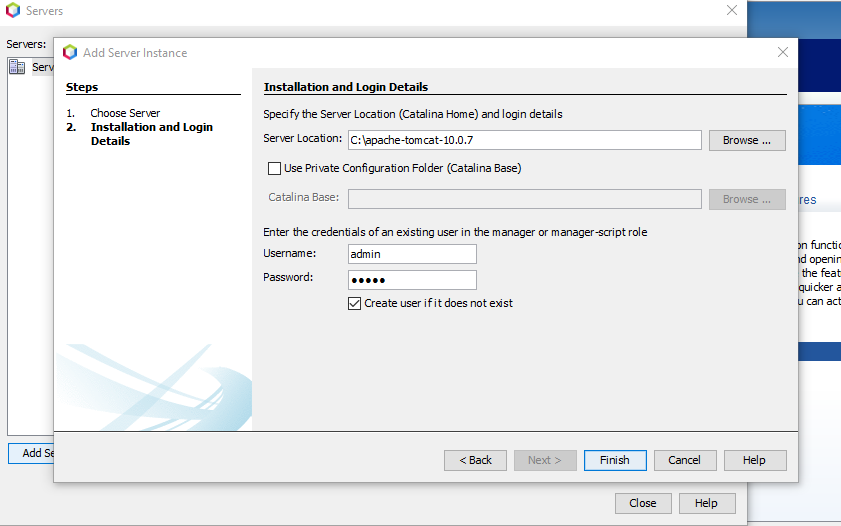


Imagen 1.2.7 Configuración de credenciales del servidor web.

Automáticamente colocará la configuración básica de nuestro servidor web Tomcat 10, en donde se muestra el servidor de puerto predeterminado que es configurable si es que otro servicio no esta utilizándolo, la recomendación para estos casos es dejar el puerto predeterminado, también mostrará las credenciales antes ingresadas de nuestro servidor web, para terminar, damos clic en *Close* para terminar la instalación del componente, como se muestra en la imagen 1.2.8.

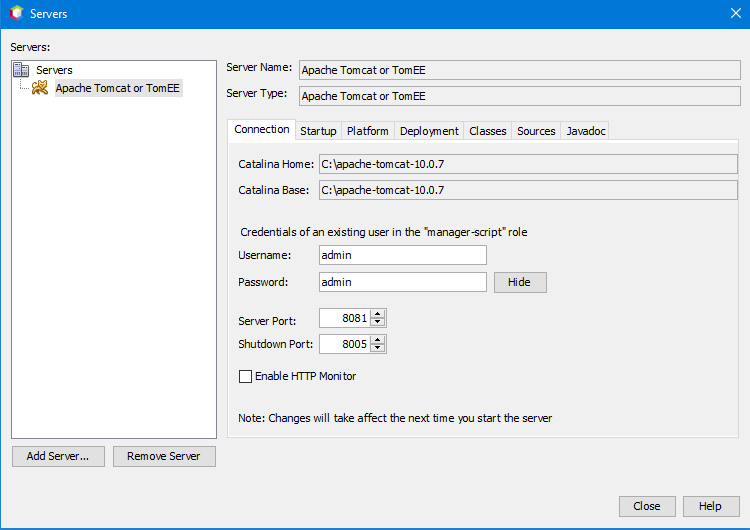


Imagen 1.2.8 Finalización de instalación del servidor web con NetBeans 12.0

## 3.1 Creación de un proyecto web TestWeb y levantamiento del servidor Tomcat 10.

Para hacer uso de nuestro servidor web antes instalado en el apartado 3.0, debemos regresar a nuestro IDE NetBeans donde crearemos un proyecto nuevo tal como se muestra en la imagen 1.2.9 donde el tipo será un *Web Application*, en este apartado nombraremos el proyecto como *TestWeb* y daremos clic en el botón de *Next*.

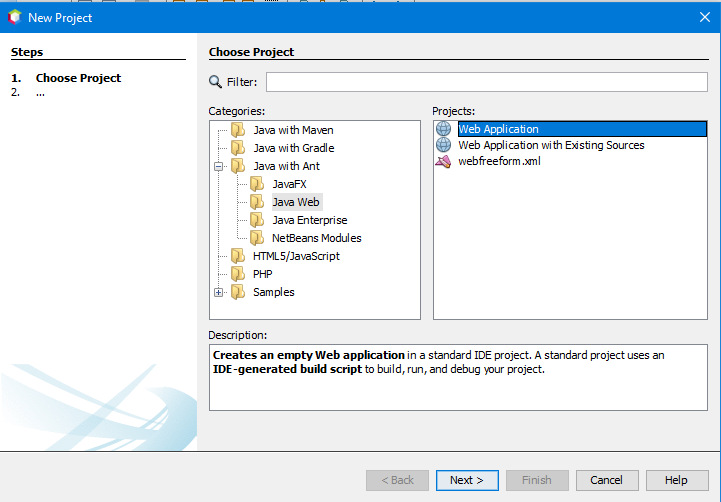


Imagen 1.2.9 Creación del proyecto web TestWeb.

Para continuar nombraremos el proyecto y elegiremos el lugar donde estará alojado nuestro proyecto, terminado esto daremos clic en el botón Nextcomo se muestra en la imagen 1.3.0.

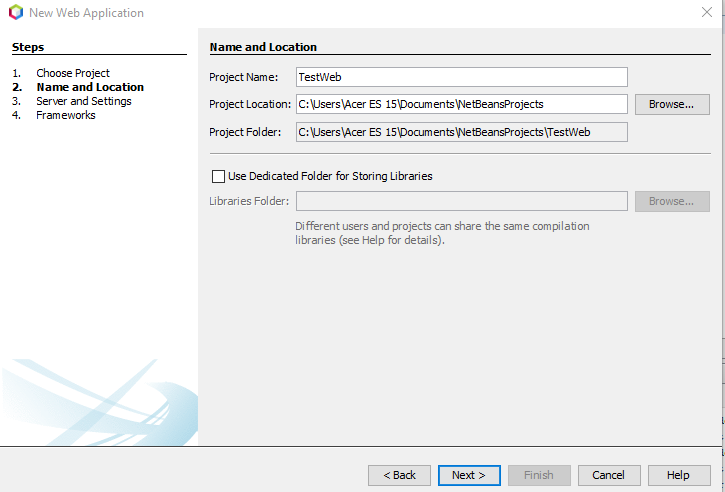


Imagen 1.3.0 Nombramiento del proyecto web.

En el siguiente apartado nos mostrará el servidor mediante el cual se estará desplegando dentro de nuestro navegador web, en este apartado se pueden tener más de un servidor, pero solo se puede seleccionar uno principalmente; en este caso elegimos el servidor web *Apache Tomcat or TomEE* tal como se muestra en la imagen 1.3.1, al finalizar damos clic en el botón *Next*.

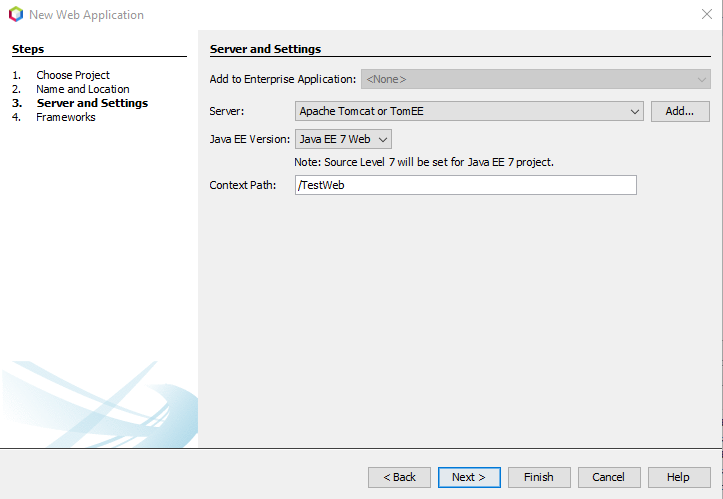


Imagen 1.3.1 Elección de servidores.

Para concluir con la generación del proyecto damos clic en el botón *Finish* como se muestra en la imagen 1.3.2; en este apartado esperaremos un momento para que el IDE genere el proyecto con nuestra configuración antes generada, al terminar nos mostrará nuestro proyecto como la imagen 1.3.3 con las librerías básicas.

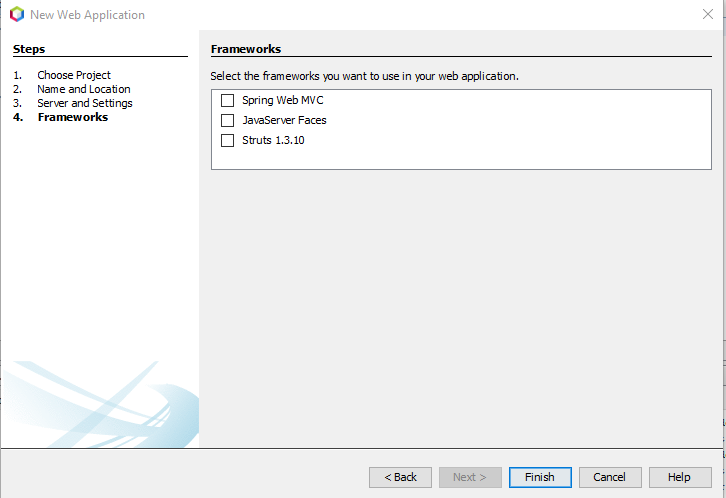


Imagen 1.3.2 Finalización de instalación del proyecto.

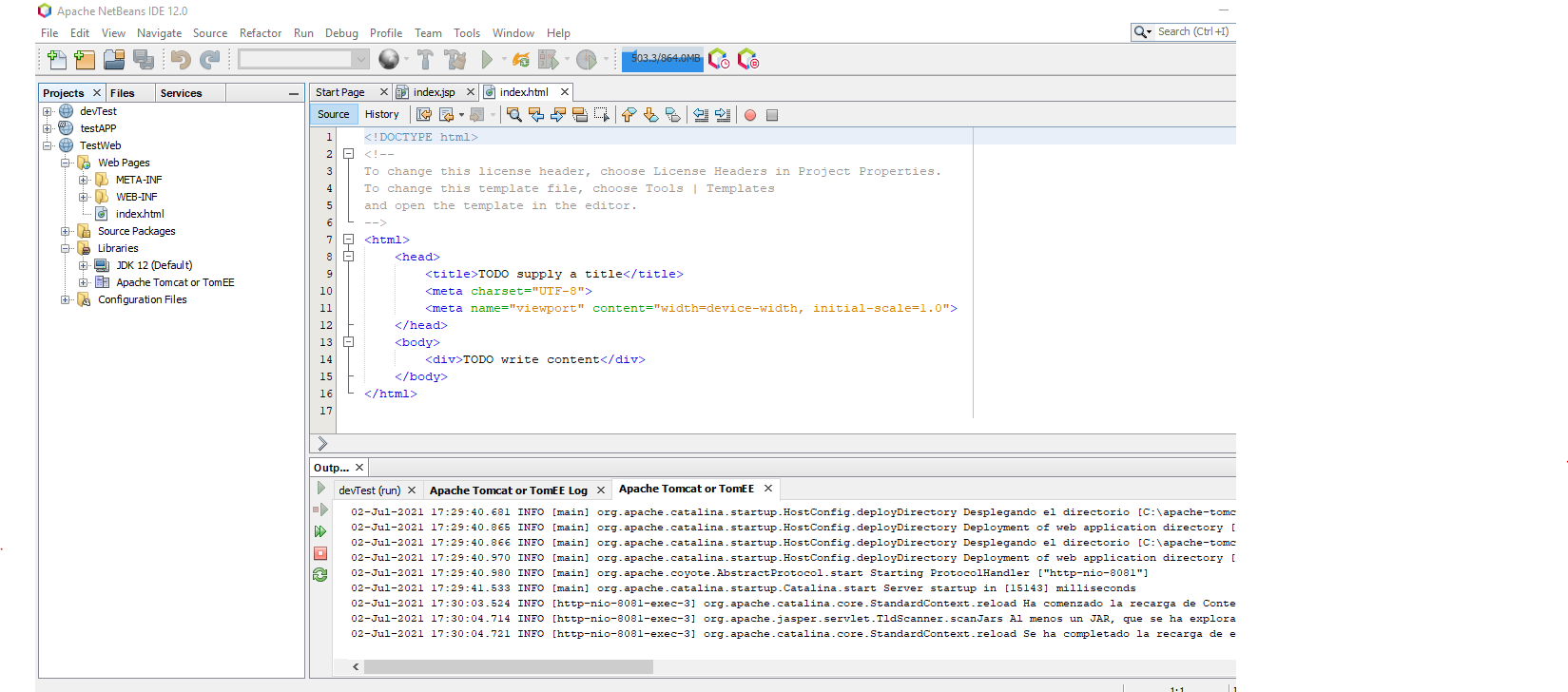


Imagen 1.3.3 Generación un proyecto web.

Si todo ha sido satisfactorio abrimos el archivo index.html que el proyecto contiene por inicio y corremos el proyecto para desplegar el proyecto con el servidor Tomcat 10, como se muestra la imagen 1.3.4, esto nos indicará que el proyecto se generó de manera correcta y que el servidor Tomcat funciona de manera correcta en la integración con NetBeans 12.0 tal como se muestra en la imagen.

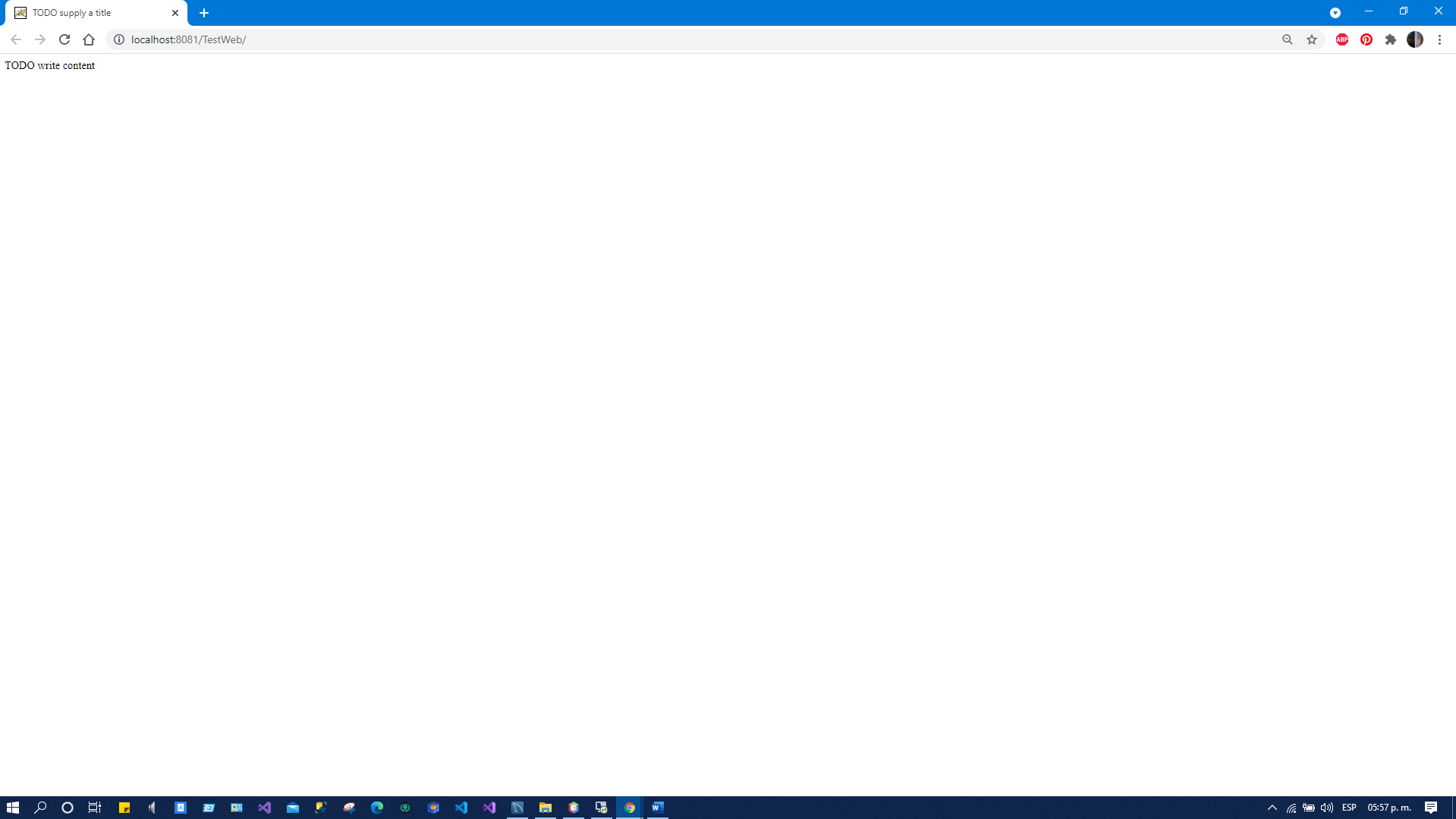
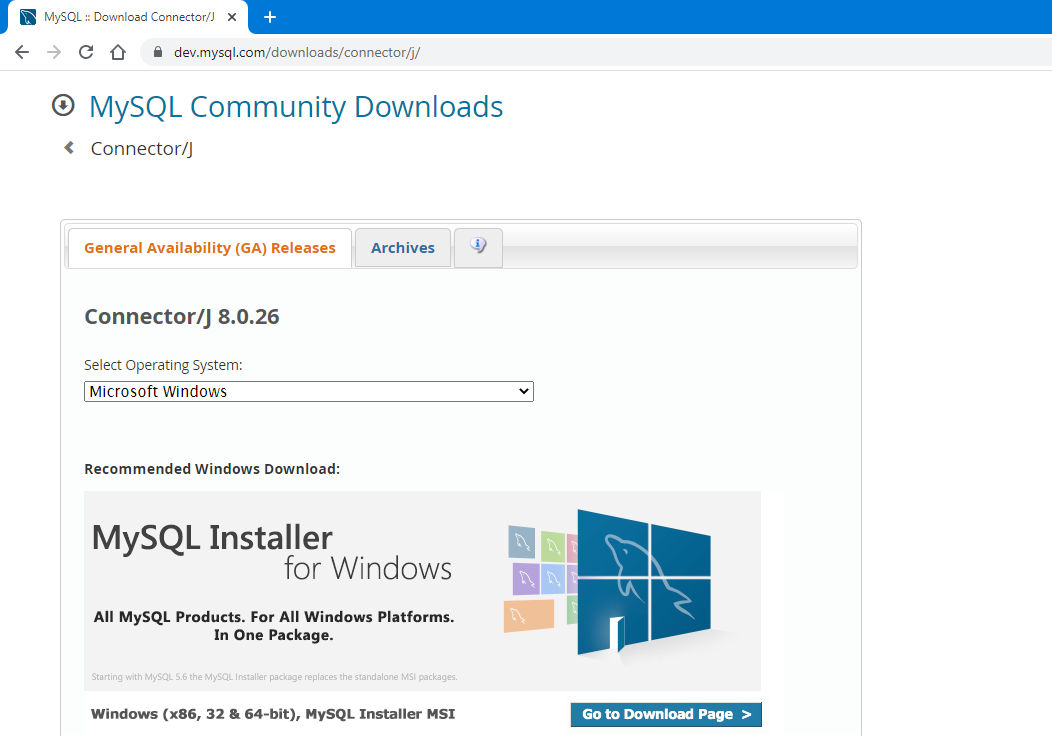


Imagen 1.3.4 Despliegue y levantamiento del servidor

## Instalación del componente JDBC en el proyecto de desarrollo en el IDE NetBeans 12.0.

Como mencionamos en apartados anteriores requerimos utilizar un conector JDBC para realizar consultas directo a la base de datos MySQL con el fin de interactuar con la información desde la vista del cliente o también llamado usuario.

Para descargar el componente necesario es requerido ir a la siguiente direccion web: <https://dev.mysql.com/downloads/connector/j/> dentro de esta nos desplegará una página como se muestra en la imagen 1.4.0:



1.4.0 Sitio de descarga del conector JDBC.

Para obtener directamente el conector elegiremos la opción plataforma independiente en el apartado de elección del sistema operativo y daremos clic en la opción de descarga; esta opción nos descargará un archivo zip nombrado *mysql-connector-java-8.0.26.zip* que contiene el archivo JAR como se muestra en la imagen 1.4.1 para incorporar al proyecto anteriormente generado.

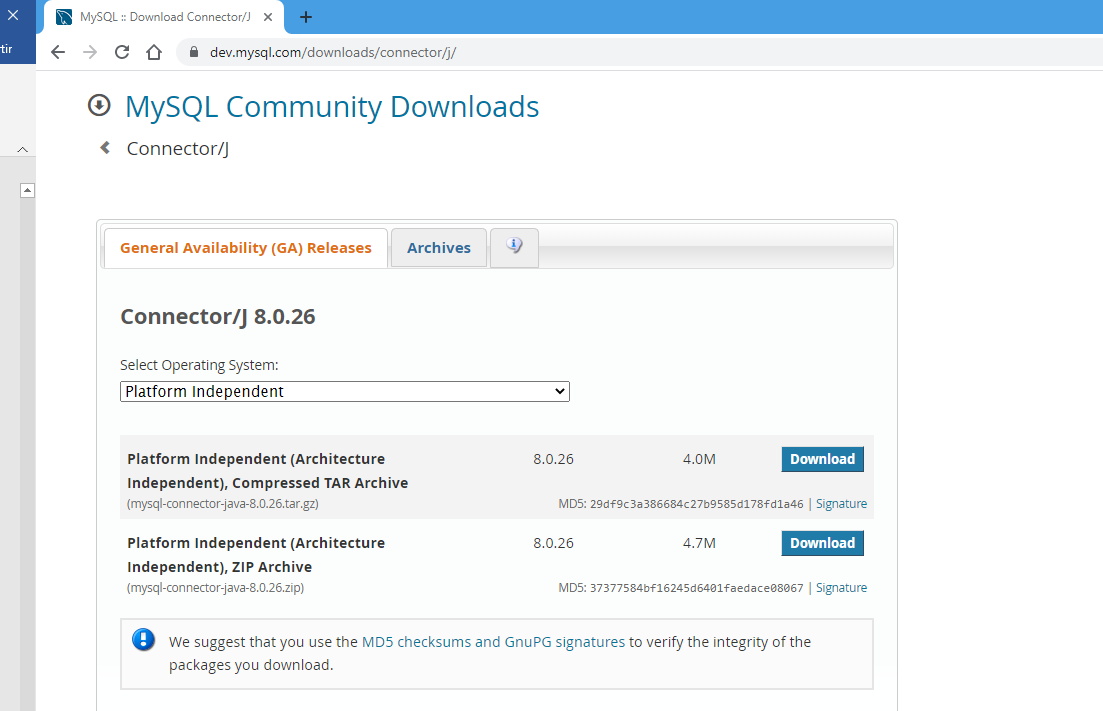


Imagen 1.4.1 descarga del archivo.

Para llevar el control de nuestros archivos es recomendable moverlos de ruta en nuestro equipo de cómputo para hacer uso de ellos de manera más sencilla.

Para hacer uso del conector JDBC es necesario ir al proyecto *TestWeb* de nuestro IDE NetBeans 12, en apartado de librerías damos clic derecho, seleccionamos *Add JAR/Folder* donde nos permitirá seleccionar el archivo JAR que previamente descargamos y también descomprimimos en nuestra ruta establecida, como se muestra en la imagen 1.4.2.

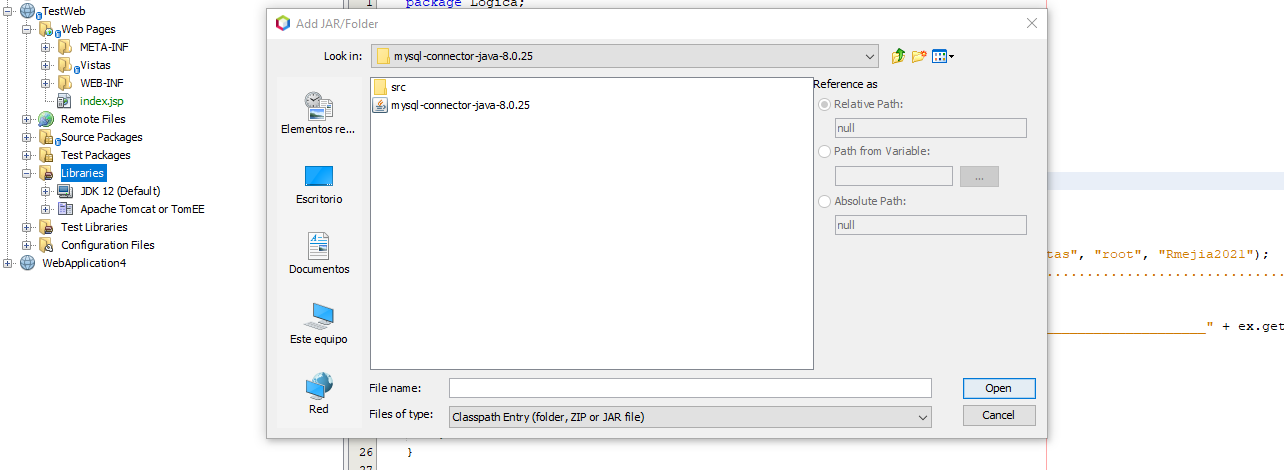


Imagen 1.4.2 Selección de archivo JAR.

# 4.0 Estructuración del patrón de diseño MVC en NetBeans 12.0.

Dentro del desarrollo de sistemas es necesario estructurar los proyectos de acuerdo a ciertas características ya que con ello nos permite seguir un patrón de diseño de manera más fácil y efectiva. Dentro de nuestra solución para estructurar el proyecto de acuerdo con el patrón de diseño MVC es necesario realizar las siguientes acciones:

1.-En el proyecto web *TestWeb* agregaremos los paquetes necesarios en el apartado de *Source Package* (Paquete de recursos), para ello para generar cada paquete dando clic derecho *New* (Nuevo) > *Java Package,* donde siguiendo la estructura del patrón de diseño MVC generaremos 3 paquetes los cuales nombraremos nuestros paquetes de la siguiente manera: Controlador(es) llamada también capa Controladora, Modelo(s) nombrada también capa de Modelo y Lógica (El paquete de lógica se realiza de manera opcional para dividir la capa Lógica del sistema de la parte Controladora ya que muchos métodos y clases regularmente se generan en el paquete Controlador).Al generar los paquetes necesarios para seguir el diseño de MVC estructuraremos nuestro proyecto de la siguiente manera como se muestra en la imagen 1.4.3.

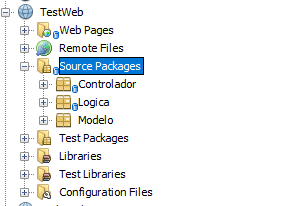


Imagen 1.4.3 Estructuración de paquetes.

2.-En el apartado *Web Pages* agregaremos la parte o también llamada capa de presentación de MVC, para ello dando clic derecho New (Nuevo) > Folder donde nombraremos nuestra estructura de vistas nombrándola de la misma manera tal como se muestra en la imagen 1.4.4.

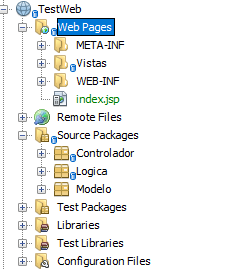


Imagen 1.4.4 Estructuración de la capa de presentación.

# 5.0 Prueba de integración con la base de datos MySQL.

En nuestro proyecto podemos generar pruebas de integración con el conector JDBC, esto nos permitirá saber si tenemos acceso a nuestra base de datos siguiendo el MVC, para ello nos apoyaremos de una clase esencial que a partir de aquí nombraremos como *clase* *Conexión.* En nuestro proyecto *TestWeb* generaremos una clase Java dentro del paquete Lógica con el nombre de Conexión, para ello dando clic derecho en el paquete lógica *New (Nuevo) > Java Class* la cual nos generará un archivo .java como se muestra en la imagen 1.4.5.

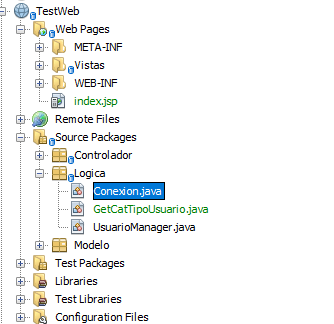


Imagen 1.4.5 Generación de clases de acceso a base de datos.

Para hacer uso del archivo JAR que nos permite la interacción con la base de datos MySQL que generamos anteriormente llamada *DBVENTAS* dentro de la clase *Conexión* utilizaremos el siguiente código que nos permitirá ingresar la cadena de conexión con la(s) base de datos, como se muestra a continuación.

package Logica**;**

**import** java**.**sql**.**Connection**;**

**import** java**.**sql**.**DriverManager**;**

public class Conexion **{**

Connection con**;**

private static String db**=**"dbventas"**;**

private static String usuario**=**"root"**;**

private static String password**=**"Rmejia2021"**;**

private static String conexion**=**"jdbc:mysql://localhost:3306/"**+**db**;**

public Conexion**()** **{**

**try** **{**

Class**.**forName**(**"com.mysql.jdbc.Driver"**)**

con **=** DriverManager**.**getConnection**(**conexion**,**usuario**,**password**);**

System**.**out**.**println**(**"Conexion exitosa con la base de datos dbventas.....................................\n"**);**

**}** **catch** **(**Exception ex**)** **{**

System**.**err**.**println**(**"Error de conexion con la base de datos:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" **+** ex**.**getMessage**());**

**}**

**}**

public Connection getConection**()** **{**

**return** con**;**

**}**

**}**

Para acceder a los métodos de nuestra clase *Conexión.java* tenemos un método publico llamado *getConection()* donde podemos acceder de manera rápida y sencilla con la base de datos *DBVENTAS* por medio del constructor de la clase Conexión.

Ejemplificando nuestra conexión a la base de datos en primera instancia requerimos obtener el catalogo de usuarios de nuestra base de datos *DBVENTAS* de la tabla *CatTipoUsuario,* necesitamos generar 3 partes.

1.-La parte de presentación de nuestro catalogo consultado, por lo cual requerimos generar un JSP (Java Server Pages) para presentar dicha tabla en el ambiente web.

Dicho antes el JSP con el objetivo de presentar la tabla anteriormente descrita con lenguaje HTML requerimos generar el archivo de la siguiente manera, en la carpeta *Vistas* dando clic derecho *New (Nuevo) > JSP* nos permitirá generar el archivo en la ruta nombrándolo de la siguiente manera CatalogoUsuarios.jsp tal como se muestra en la imagen 1.4.6*.*

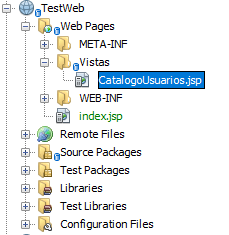


Imagen 1.4.6 Creación de JSP CatalogoUsuarios.jsp.

2.- Para Serializar o, dicho de otra manera, manejar nuestra consulta a la base de datos en objetos requerimos generar los también llamados *JavaBeans* que son clases que describen las entidades de nuestra base de datos en código Java, los cuales deben cumplir con ciertas normas como: Accesibilidad en métodos como getters y setters, las variables utilizadas para referirse a las propiedades de cada tabla requieren ser privadas y debe contener un constructor vacío.

Para generar un JavaBean se requiere generar una clase Java en el paquete llamado Modelo que creamos con anterioridad en el apartado 4.0; nombraremos esta clase en java como *CatTipoUsuario.java* y crearemos nuestra entidad como el siguiente código presentado a continuación.

package Modelo**;**

**import** java**.**io**.**Serializable**;**

public class CatTipoUsuario **implements** Serializable**{**

private int idcattipoUsuario**;**

private String descripcion**;**

public CatTipoUsuario**(){**

**}**

public int getIdcattipoUsuario**()** **{**

**return** idcattipoUsuario**;**

**}**

public void setIdcattipoUsuario**(**int idcattipoUsuario**)** **{**

**this.**idcattipoUsuario **=** idcattipoUsuario**;**

**}** public String getDescripcion**()** **{**

**return** descripcion**;**

**}** public void setDescripcion**(**String descripcion**)** **{**

**this.**descripcion **=** descripcion**;**

**}** **}**

3.-Para tener acceso a nuestra tabla *CatTipoUsuario* y obtener nuestra tabla en forma de objetos se requiere considerar la creación de un componente lógico que nos recopilara toda la información y nos la presentara en un método que denominaremos GetListCatTipoUsuario*()*, para la creación del mismo requerimos generar una clase Java en el paquete Lógica presentado en anteriores apartados, nombraremos a esta clase *GetCatTipoUsuario.java,* para finalizar con el siguiente código incorporaremos lo establecido en los pasos anteriores, utilizaremos la clase *Conexión* como una extensión de la clase creada, para hacer uso del método *getConection()* requerimos una variable de tipo *Connection*  donde asignaremos el método.

Para generar una consulta *Select* utilizaremos una variable de tipo String que ingresaremos en una instancia de la variable *Connection* llamada *prepareStatement*, para ejecutar la consulta generaremos una variable de tipo *ResultSet* dondeasignaremos la instancia *prepareStatement* con el método *executeQuery ()* el cual nos brindara la consulta establecida con anterioridad y para terminar asignaremos los resultados una lista de objetos de tipo *CatTipoUsuario* como se muestra en el siguiente código.

package Logica**;**

**import** Modelo**.**CatTipoUsuario**;**

**import** Modelo**.**Usuario**;**

**import** java**.**sql**.**Connection**;**

**import** java**.**sql**.**PreparedStatement**;**

**import** java**.**sql**.**ResultSet**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

public class GetCatTipoUsuario **extends** Conexion **{**

public List**<**CatTipoUsuario**>** GetListCatTipoUsuario**()** **{**

List**<**CatTipoUsuario**>** lstCatUsuario **=** **new** ArrayList**<>();**

**try** **{**

Connection con **=** getConection**();**

String query **=** "select \* from cattipousuario"**;**

PreparedStatement st **=** con**.**prepareStatement**(**query**);**

ResultSet rs **=** st**.**executeQuery**();**

**while** **(**rs**.**next**())** **{**

CatTipoUsuario ctu **=** **new** CatTipoUsuario**();**

ctu**.**setIdcattipoUsuario**(**rs**.**getInt**(**1**));**

ctu**.**setDescripcion**(**rs**.**getString**(**2**));**

lstCatUsuario**.**add**(**ctu**);**

**}**

System**.**out**.**println**(**"Registros obtenidos correctamente"**);**

con**.**close**();**

**}** **catch** **(**Exception E**)** **{**

System**.**out**.**println**(**"Error:" **+** E**.**getMessage**());**

**return** **null;** **}**

**return** lstCatUsuario**;** **}**

Para generar la presentación de nuestros resultados requerimos establecer una tabla en HTML como se muestra en la imagen 1.4.7, donde encontraremos elementos como la instancia a nuestro método antes generado y donde podemos presentar nuestros datos recorriendo nuestra lista en un ciclo for incrustado en el JSP y siempre cuidando la importación de paquetes en el encabezado del mismo como se muestra en la imagen 1.4.8.

<table id**=**"mytabla" **class=**"table table-striped table-bordered dt-responsive nowrap"**>**

**<**thead**>**

**<**tr**>**

**<**th**>**Id**</**th**>**

**<**th**>**Descripción**</**th**>**

**<**th**>**Opciones**</**th**>**

**</**tr**>**

**</**thead**>**

**<**tbody**>**

**<%**

List**<**CatTipoUsuario**>** listaCU**=(**List**<**CatTipoUsuario**>)** request**.**getAttribute**(**"listCatTipoUsuario"**);**

**for(**CatTipoUsuario u**:** listaCU**){** **%>**

**<**tr**>**

**<**td**>** **<%=** u**.**getIdcattipoUsuario**()%>** **</**td**>**

**<**td**>** **<%=** u**.**getDescripcion**()%>** **</**td**>**

**<**td**>**

**<button** type**=**"button" **class=**"btn btn-default" aria**-**label**=**"Left Align"

**onclick=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Editar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span **class=**"glyphicon glyphicon-pencil" aria**-hidden=**"true"**></**span**>**

**</button>**

**<button** type**=**"button" **class=**"btn btn-default" aria**-**label**=**"Left Align"

**onclick=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Eliminar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span **class=**"glyphicon glyphicon-remove" aria**-hidden=**"true"**></**span**>**

**</button>**

**</**td**>**

**<%**

**}**

**%>**

**</**tr**>**

**</**tbody**>**

**</**table**>**

Imagen 1.4.7 Presentación del Catálogo de tipo de usuarios.

**<%**@page **import=**"java.util.List" **%>**

**<%**@page **import=**"Modelo.CatTipoUsuario" **%>**

**<%**@page **import=**"Logica.CatTipoUsuarioManager" **%>**

**<%**@page **import=**"Controlador.ConsultaTipoUsuario" **%>**

**<%**@page **import=**"Controlador.ConsultaCatalogoTipoUsuarios" **%>**

Imagen 1.4.8 Importación de paquetes.

Para finalizar requerimos instanciar nuestro JSP para presentarse en el navegador, para ello en nuestro archivo *Index.jsp* agregaremos una referencia a nuestro *CatalogoUsuarios.jsp* con una etiqueta de referencia como se muestra en el siguiente código.

*<a href="Vistas/CatalogoUsuarios.jsp">CatalogoTipoUsuarios</a>*

Nota Importante: Para integrar presentación a la tabla se utilizaron scripts de tablas de Javascript llamado *datatable.js* y de Bootstrap 4.

Al momento de ejecutar el proyecto obtendremos una vista en el navegador como se presenta en la imagen 1.4.9 y al hacer clic se presentará la consulta en una tabla como se muestra en la imagen 1.5.0 de la tabla *CatTipoUsuario.*

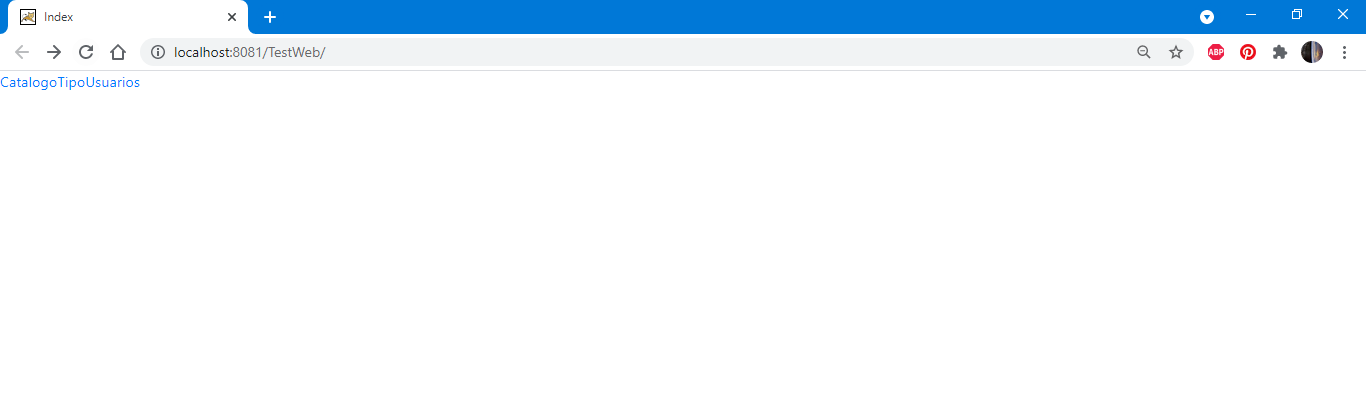


Imagen 1.4.9 Despliegue Ejemplo consulta a la base de datos.



Imagen 1.5.0 Despligue de una consulta en una table de JSP.

## 5.0 Ejemplo de un CRUD MVC utilizando Servlets y JSP.

Para generar un CRUD utilizando la tecnología java es necesario considerar en primera instancia generar componentes denominados Servlets que son clases en java especializadas en el manejo de información por solicitudes que apoyan a los JSP con las peticiones http con métodos como GET y POST que son métodos generales que envían información ya sea mediante la URL directamente lo cual lo especifica los métodos GET, este tipo de información es sencilla ya que de manera general estos métodos se utilizan para obtener información. Los métodos POST nos proporcionan solicitudes bilaterales de información ya se enviando información sensible para el usuario y obteniendo respuestas del mismo con más información.

En el siguiente ejemplo se muestra cómo se implementa un CRUD MVC utilizando esta tecnología.

Primeramente, en el archivo JSP llamado CatalogoTipoUsuarios generaremos los siguientes cambios:

1.-Crearemos una columna dentro de nuestra tabla en la cual se presentan los datos de la primera consulta vista en apartado anterior 5.0 con los elementos que se muestran en la imagen 1.5.1.

En la imagen 1.5.1 incluimos la manera de navegar entre los JSP incluidos en la carpeta Vistas que serán llamadas para consultar mediante el identificador del elemento seleccionado ya que estos generan un identificador por cada elemento dentro del ciclo *For* que se creó con anterioridad.

Este tipo de solicitudes son de tipo GET las cuales incluyen en la URL de la llamada los JSP la información básica para consultar rápidamente las solicitudes que se requieren; a continuación, se mostrará cómo se obtiene la información enviada desde el JSP.

**<**td**>**

**<button** type**=**"button" **class=**"btn btn-default" aria**-**label**=**"Left Align"

**onclick=**"location.href='../Vistas/AccionesCatalogoTipoUsuarios/Editar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span **class=**"glyphicon glyphicon-pencil" aria**-hidden=**"true"**></**span**>**

**</button>**

**<button** type**=**"button" **class=**"btn btn-default" aria**-**label**=**"Left Align"

**onclick=**"location.href='../Vistas/AccionesCatalogoTipoUsuarios/Eliminar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span **class=**"glyphicon glyphicon-remove" aria**-hidden=**"true"**></**span**>**

**</button>**

**</**td**>**

Imagen 1.5.1 Generación de enlaces entre JSP.

Dentro de la carpeta Vistas incluiremos una nueva carpeta de nombre AccionesCatalogoTipoUsuarios como creamos en apartados anteriores, generaremos 3 archivos de tipo JSP nombrados como se muestra en la imagen 1.5.2, estos archivos nos permitirán manejar las funciones básicas en la base de datos DBVENTAS denominadas con el acrónimo CRUD (Create, Read, Update y Delete).

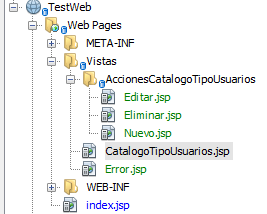


Imagen 1.5.2 Estructura general de carpetas para la generación CRUD.

Se utilizará el archivo Nuevo.jsp donde crearemos nuevos elementos identificando el MVC enviando información a la base de datos, en el caso de la tabla CatTipoUsuario, esta solo requiere la descripción del nuevo elemento, para este caso crearemos una entrada de texto con la etiqueta de *name* por la cual recuperaremos el valor ingresado por el usuario, tal como se muestra en el código 1.5.3; para enviar los datos es necesario insertar un elemento *HTML button*  de tipo *submit*  como se muestra en la imagen 1.5; para enviar la solicitud al *Servlets* donde es necesario crearlo en el paquete *Controlador*  con el nombre de *NuevoTipoUsuario.*

*Nota: En nuevas versiones de IDE como es el caso de NetBeans 12.0 apache, al generar los Servlets de manera nativa, es necesario considerar renombrar el paquete JAVAX a JAKARTA para hacer uso de los elementos http para utilizar los diferentes métodos de los mismos.*

**<%** String path **=** request**.**getContextPath**();** **%>**

**<form** method**=**"post" action**=**"<%= path %>/NuevoTipoUsuario"**>**

**<**br**/>**

**<**br**/>**

**<**div **class=**"row col-md-12"**>**

**<**div **class=**"col-md-4" **>**

**<**input **class=**"form-control" type**=**"text" name**=**"descripcion" placeholder**=**"Descripción"**/>**

**</**div**>**

**</**div**>**

**<**div **class=**"row col-md-12"**>**

**<**div **class=**"col-md-3"**>**

**<button** type**=**"submit" **class=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"1" name**=**"submitCatalogo"**>**

Regresar

**</button>**

**</**div**>**

**<**div **class=**"col-md-3"**>**

**<button** type**=**"submit" **class=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"2" name**=**"submitCatalogo"**>**

Guardar

**</button>**

**</**div**>**

**</**div**>**

**</form>**

Código 1.5.3 Envió de datos hacia el Servet.

Para atender las solicitudes en un Servlet como es el caso del nombrado *NuevoTipoUsuario* es necesario es necesario recopilar la información con el siguiente código en la imagen 1.5.4, donde por cada elemento con información es necesario obtenerlo por su nombre descrito en el JSP y obtenido por parámetro en *request* y obtenido por el método *getParameter* incorporando el nombre del elemento.

String valor **=** request**.**getParameter**(**"descripcion"**);**

Imagen 1.5.4 Recepción de información.

Para enviar la información a la base de datos, como lo describe el patrón de diseño MVC es necesario de crear un objeto de tipo *CatTipoUsuario enviando* en el método *setDescripcion* la descripcion obtenida y enviando el objeto con la información con un objeto auxiliar de envió a la base de datos y obteniendo como respuesta de la ejecución de la consulta como se muestra a continuación:

package Controlador**;**

**import** Logica**.**CatTipoUsuarioManager**;**

**import** Modelo**.**CatTipoUsuario**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** jakarta**.**servlet**.**ServletException**;**

**import** jakarta**.**servlet**.**annotation**.**WebServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServletRequest**;**

**import** jakarta**.**servlet**.**http**.**HttpServletResponse**;**

@WebServlet**(**name **=** "NuevoTipoUsuario"**,** urlPatterns **=** **{**"/NuevoTipoUsuario"**})**

public class NuevoTipoUsuario **extends** HttpServlet **{**

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

response**.**setContentType**(**"text/html;charset=UTF-8"**);**

**try** **(**PrintWriter out **=** response**.**getWriter**())** **{**

int valorBoton **=** Integer**.**parseInt**(**request**.**getParameter**(**"submitCatalogo"**));**

String valor **=** request**.**getParameter**(**"descripcion"**);**

**if** **(**valorBoton **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

**if** **(!**valor**.**isEmpty**())** **{**

CatTipoUsuario u **=** **new** CatTipoUsuario**();**

u**.**setDescripcion**(**valor**);**

CatTipoUsuarioManager registrar **=** **new** CatTipoUsuarioManager**();**

int registro **=** registrar**.**registrarCatTipoUsuario**(**u**);**

**if** **(**registro **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Error.jsp"**);**

**}** **}** **}**

**}**

**}**

Método auxiliar dentro de una clase extendida de la clase Conexión.

**import** Modelo**.**CatTipoUsuario**;**

**import** java**.**sql**.**Connection**;**

**import** java**.**sql**.**PreparedStatement**;**

**import** java**.**sql**.**ResultSet**;**

**import** java**.**sql**.**SQLException**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**List**;**

public class CatTipoUsuarioManager **extends** Conexion **{**

public int registrarCatTipoUsuario **(**CatTipoUsuario ctu**)** **{**

**try** **{**

int estatus **=** 0**;**

Connection con **=** getConection**();**

String query **=** "insert into cattipousuario (Descripcion)values(?)"**;**

PreparedStatement st **=** con**.** prepareStatement**(**query**);**

st**.** setString**(**1**,** ctu**.** getDescripcion **());**

estatus **=** st**.**executeUpdate**();**

con**.**close**();**

**return** estatus**;**

**}** **catch** **(**Exception r**)** **{**

**return** 0**;**

**}**

**}**

**}**

Como resultado final esperamos una navegación e integración como la siguiente imagen 1.5.6 donde incorporamos un botón con enlace hacia el *JSP Nuevo* de acuerdo al diseño que se requiera.

**<**div **class=**"col-lg-5"**>**

**<%** String path **=** request**.**getContextPath**();** **%>**

**<button** type**=**"button" **class=**"btn btn-primary" aria**-**label**=**"Left Align"

**onclick=**"location.href='../Vistas/AccionesCatalogoTipoUsuarios/Nuevo.jsp'"**>**

Nuevo

**</button>**

**</**div**>**

Imagen 1.5.6 Enlace al JSP para la creación de elementos.

Dado las características anteriores se muestran los resultados en las imágenes 1.5.7, 1.5.8 y 1.5.9.

*Nota: Se utilizaron librerías de Bootstrap para darle diseño a la entrega final.*

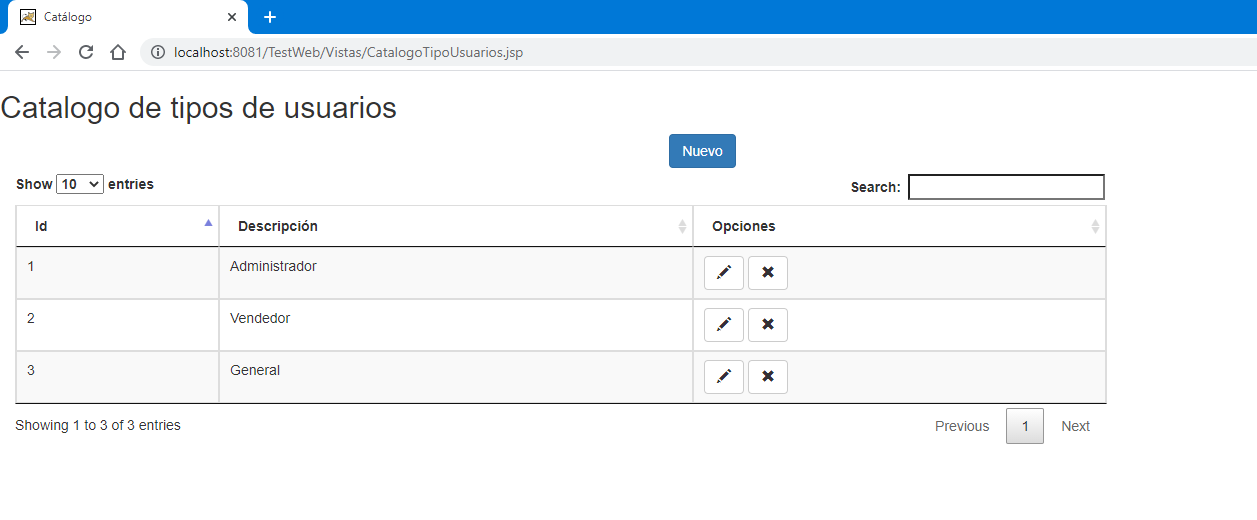


Imagen 1.5.7 Consulta de registros.

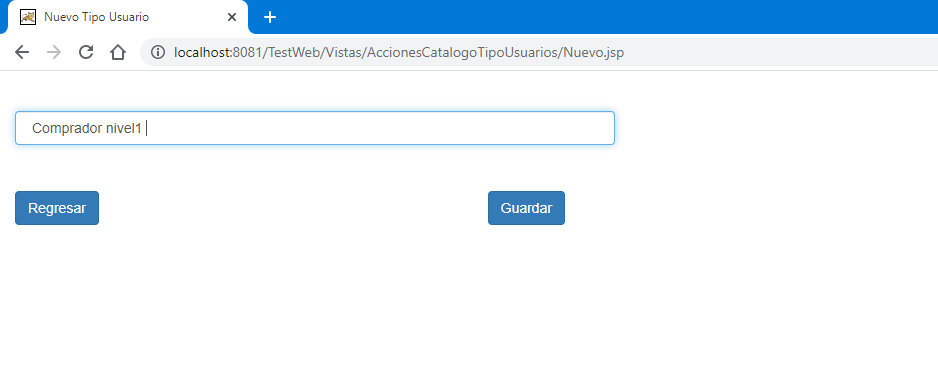


Imagen 1.5.8 Formulario de creación de registros.

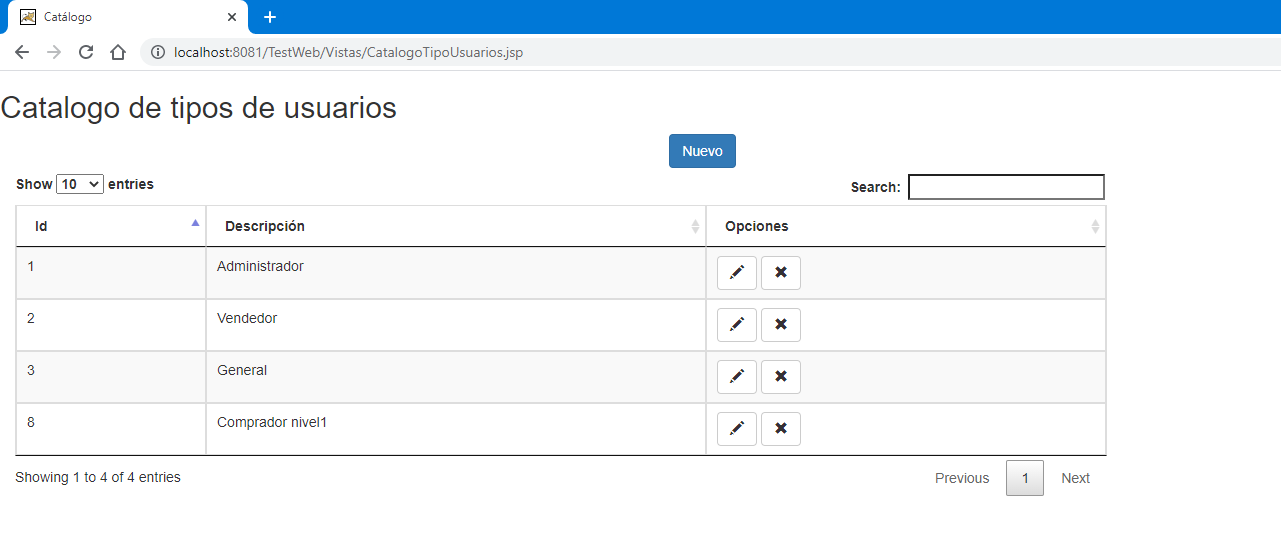


Imagen 1.5.8 Actualización de registros.

## 5.1 Consulta de registros y envió de información desde un Servlets hacia un JSP

En esta sección se concentrará el método de envío de información desde un Servlets hacia un JSP mediante peticiones http.

Para generar solicitudes de información de consulta sin parámetros se requiere hacer la petición al Servlet con los distintos métodos como GET y POST, en el siguiente ejemplo se requiere obtener la información y colocarla en una tabla para posteriores tratamientos siguiendo el patrón de diseño MVC el cual establece que se requiere solicitar información desde un JSP pasando la solicitud al Servlet y generar la solicitud para enviar dicha solicitud al JSP.

Para ello requerimos generar un Servlet nombrado *ConsultaCatalogoTipoUsuarios.java* el cual nos permitirá obtener las solicitudes generadas en desde una acción anterior, en el siguiente código se muestra como se genera la consulta de elementos de tipo *CattipoUsuario* el cual regresaremos en una variable de tipo *request.setAttribute*  el cual recibe dos parámetros*: String* el cual identifica el atributo a utilizar, *Object* el cual puede ser uno o varios objetos utilizando listas genéricas. Para que este proceso se complete se requiere enviar a un JSP específico para ello se debe utilizar **request*.getRequestDispatcher("/Vistas/CatalogoTipoUsuarios.jsp").forward(request, response);***  el cual nos permite enviar la información registrando los parámetros a utilizar asi como la ruta específica del JSP a redireccionar, tal como se muestra en el siguiente código:

package Controlador**;**

**import** Logica**.**CatTipoUsuarioManager**;**

**import** Modelo**.**CatTipoUsuario**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** jakarta**.**servlet**.**ServletException**;**

**import** jakarta**.**servlet**.**annotation**.**WebServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServletRequest**;**

**import** jakarta**.**servlet**.**http**.**HttpServletResponse**;**

**import** java**.**util**.**List**;**

@WebServlet**(**name **=** "ConsultaTipoUsuario"**,** urlPatterns **=** **{**"/ConsultaTipoUsuario"**})**

public class ConsultaTipoUsuario **extends** HttpServlet **{**

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

response**.**setContentType**(**"text/html;charset=UTF-8"**);**

**try** **(**PrintWriter out **=** response**.**getWriter**())** **{**

CatTipoUsuarioManager usc **=** **new** CatTipoUsuarioManager **();**

List**<**CatTipoUsuario**>** lista **=** usc**.**GetListCatTipoUsuario**();**

request**.**setAttribute**(**"listCatTipoUsuario"**,** lista**);**

request**.**getRequestDispatcher**(**"/Vistas/CatalogoTipoUsuarios.jsp"**).**forward**(**request**,** response**);**

**}**

**}**

@Override

protected void doGet**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

processRequest**(**request**,** response**);**

**}**

@Override

protected void doPost**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

processRequest**(**request**,** response**);**

**}**

@Override

public String getServletInfo**()** **{**

**return** "Short description"**;**

**}**// </editor-fold>

**}**

Para obtener la información en el código anterior se requiere utilizar el método *GetListCatTipoUsuario* el cual permite obtener la información desde la base de datos utilizando objetos para su posterior envío hacia el JSP.

Para utilizar esta solicitud se requiere un enlace del SERVLET en cuestión para ello dentro del JSP se requiere agregar la siguiente línea de código, la cual nos especifica la ruta donde se estarán utilizando la información desde el Servlet.

**<%**@page **import=**"Controlador.ConsultaCatalogoTipoUsuarios" **%>**

Para tratar la información es necesario generar una tabla HTML la cual nos ayudara a organizar la información asi como los posibles métodos del CRUD como la edición y la eliminación del elemento seleccionado.

Para utilizar el parámetro y generar una lista de objetos para iterar los elementos y mostrarlos en forma de tabla es necesario utilizar la siguiente línea de código, donde además utilizaremos un ciclo for iterando los elementos identificando a cada uno con una variable que obtenga los distintos métodos del objeto utilizado:

**<**table id**=**"mytabla" class**=**"table table-striped table-bordered dt-responsive nowrap"**>**

**<**thead**>**

**<**tr**>**

**<**th**>**Id**</**th**>**

**<**th**>**Descripción**</**th**>**

**<**th**>**Opciones**</**th**>**

**</**tr**>**

**</**thead**>**

**<**tbody**>**

**<%**

List**<**CatTipoUsuario**>** listaCU**=(**List**<**CatTipoUsuario**>)** request**.**getAttribute**(**"listCatTipoUsuario"**);**

**for(**CatTipoUsuario u**:** listaCU**){**

**%>**

**<**tr**>**

**<**td**>** **<%=** u**.**getIdcattipoUsuario**()%>** **</**td**>**

**<**td**>** **<%=** u**.**getDescripcion**()%>** **</**td**>**

**<**td**>**

**<**button type**=**"button" class**=**"btn btn-default" aria**-**label**=**"Left Align"

onclick**=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Editar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span class**=**"glyphicon glyphicon-pencil" aria**-**hidden**=**"true"**></**span**>**

**</**button**>**

**<**button type**=**"button" class**=**"btn btn-default" aria**-**label**=**"Left Align"

onclick**=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Eliminar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span class**=**"glyphicon glyphicon-remove" aria**-**hidden**=**"true"**></**span**>**

**</**button**>**

**</**td**>**

**<%**

**}**

**%>**

**</**tr**>**

**</**tbody**>**

**</**table**>**

Para la utilización de este método en el archivo JSP de nombre Index se requiere hacer modificaciones tales como agregar un tag *Form* el cual nos permitirá enviar la solicitud al servlet utilizando un botón de tipo *submit* como se muestra en el siguiente código:

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

**<**input class**=**"btn btn-primary" type**=**"submit" value**=**"Entrar" **/>**

**</**form**>**

El extracto de código nos permite enviar un método post hacia el servlet *CatTipoUsuario* sin envío de parámetros. El resultado se muestra en la imagen 1.5.9.

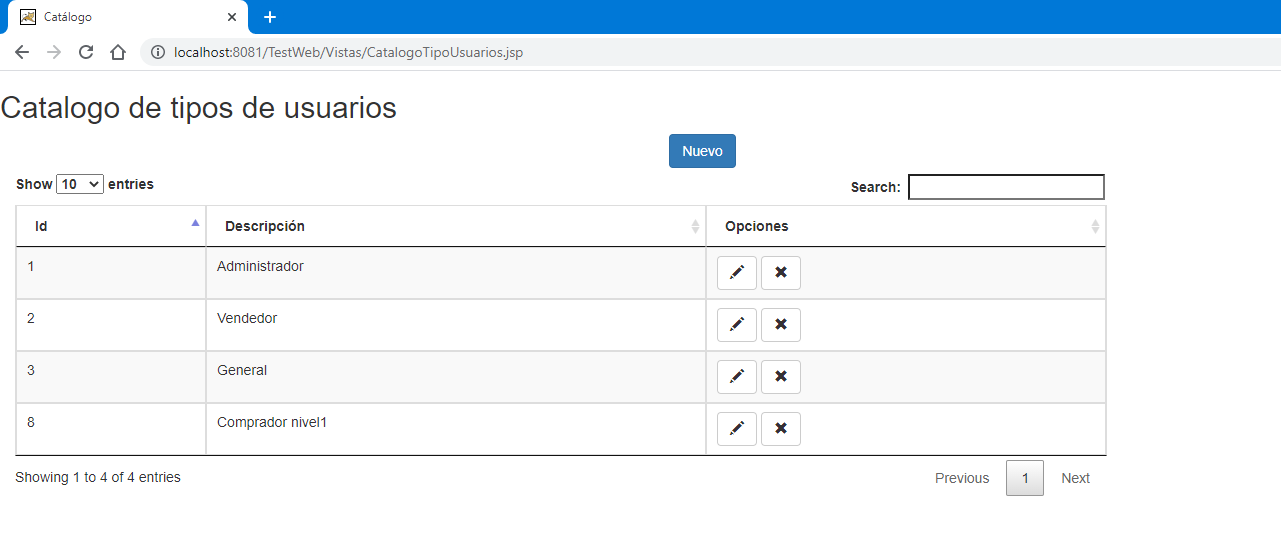


Imagen 1.5.9 Consulta de registros.

## 5.2 Edición y eliminación de registros

Para completar con el CRUD MVC en este apartado utilizaremos elementos generado en el código anterior el cual se encuentran indexados dentro de la tabla de resultados elementos como botones con referencia a distintos JSP como se muestra en el siguiente extracto de código:

**<**button type**=**"button" class**=**"btn btn-default" aria**-**label**=**"Left Align"

onclick**=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Editar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span class**=**"glyphicon glyphicon-pencil" aria**-**hidden**=**"true"**></**span**>**

**</**button**>**

**<**button type**=**"button" class**=**"btn btn-default" aria**-**label**=**"Left Align"

onclick**=**"location.href ='../Vistas/AccionesCatalogoTipoUsuarios/Eliminar.jsp?id=<%= u.getIdcattipoUsuario()%>'"**>**

**<**span class**=**"glyphicon glyphicon-remove" aria**-**hidden**=**"true"**></**span**>**

**</**button**>**

Estos botones envían el identificador del objeto bilateral ya que este envía información para que el Servlet pueda hacer uso de ella para que el usuario pueda ver y editar dicha información.

Para organizar la información dentro de la carpeta vistas utilizaremos subcarpeta que nombraremos como *AccionesCatalogoTipoUsuarios*  la cual creamos en apartados anteriores.

Para esta ocasión utilizaremos la vista *Editar.jsp* donde consultaremos el elemento a editar y mediante un Servlet se enviará la información tal como se muestra en el siguiente código.

Mediante una etiqueta *Form enviaremos* la edición de los campos con la información del objeto, se utilizará un botón de tipo *Submit* para enviar la actualización.

**<%** String path **=** request**.**getContextPath**();** **%>**

**<%**

int id**=**Integer**.**parseInt**(**request**.**getParameter**(**"id"**));**

CatTipoUsuarioManager usc**=** **new** CatTipoUsuarioManager**();**

CatTipoUsuario e**=** usc**.**GetCatTipoUsuario**(**id**);**

**%>**

**<**form method**=**"post" action**=**"<%= path %>/EditarTipoUsuario"**>**

**<**div class**=**"row col-md-12"**>**

**<**div class**=**"col-md-4" **>**

**<**input class**=**"form-control" readonly**=**"true" type**=**"text" name**=**"idS" id**=**"idS" placeholder**=**"id" value**=**"<%= e.getIdcattipoUsuario()%>" **/>**

**</**div**>**

**<**div class**=**"col-md-4" **>**

**<**input class**=**"form-control" type**=**"text" name**=**"descripcion" placeholder**=**"Descripción" value**=**"<%= e.getDescripcion()%>"**/>**

**</**div**>**

**</**div**>**

**<**div class**=**"row col-md-12"**>**

**<**div class**=**"col-md-3"**>**

**<**button type**=**"submit" class**=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"1" name**=**"submitCatalogo"**>**

Regresar

**</**button**>**

**</**div**>**

**<**div class**=**"col-md-3"**>**

**<**button type**=**"submit" class**=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"2" name**=**"submitCatalogo"**>**

Actualizar

**</**button**>**

**</**div**>**

**</**div**>**

**</**form**>**

Para tratar la información generaremos un Servlet de nombre *EditarTipoUsuario.java* en el cual especificaremos los campos obtenidos y enviaremos la información a la base de datos en la tabla correspondiente, para justificar el MVC se regresará a la pantalla principal con la actualización correspondiente.

package Controlador**;**

**import** Logica**.**CatTipoUsuarioManager**;**

**import** Modelo**.**CatTipoUsuario**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** jakarta**.**servlet**.**ServletException**;**

**import** jakarta**.**servlet**.**annotation**.**WebServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServletRequest**;**

**import** jakarta**.**servlet**.**http**.**HttpServletResponse**;**

@WebServlet**(**name **=** "EditarTipoUsuario"**,** urlPatterns **=** **{**"/EditarTipoUsuario"**})**

public class EditarTipoUsuario **extends** HttpServlet **{**

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

response**.**setContentType**(**"text/html;charset=UTF-8"**);**

**try** **(**PrintWriter out **=** response**.**getWriter**())** **{**

int valorBoton **=** Integer**.**parseInt**(**request**.**getParameter**(**"submitCatalogo"**));**

String idd**=**request**.**getParameter**(**"idS"**);**

String valor **=** request**.**getParameter**(**"descripcion"**);**

int id **=** Integer**.**parseInt**(**idd**);**

**if** **(**valorBoton **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

**if** **(!**valor**.**isEmpty**())** **{**

CatTipoUsuario u **=** **new** CatTipoUsuario**();**

u**.**setIdcattipoUsuario**(**id**);**

u**.**setDescripcion**(**valor**);**

CatTipoUsuarioManager registrar **=** **new** CatTipoUsuarioManager**();**

int registro **=** registrar**.**actualizarCatTipoUsuario**(**u**);**

**if** **(**registro **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/Error.jsp"**);**

**}**

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

**}**

**}**

**}**

**}**

}

Para generar la eliminación de registros de la misma manera utilizaremos las opciones de la tabla creada con anterioridad enviando el Id como parámetro para consulta de la información, en este caso no se podrá editar dicha consulta, para ello solo se mostrará la información y para enviar la información se enviará como parámetro al Servlet, en este caso se utilizará la vista de eliminar descrita por el siguiente código:

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

**<%**@ page **import=**"Controlador.ConsultaCatalogoTipoUsuarios" **%>**

**<%**@page **import=**"Modelo.CatTipoUsuario" **%>**

**<%**@page **import=**"Logica.CatTipoUsuarioManager" **%>**

**<!**DOCTYPE html**>**

**<**html**>**

**<**head**>**

**<**meta http**-**equiv**=**"Content-Type" content**=**"text/html; charset=UTF-8"**>**

**<**title**>**Eliminar**</**title**>**

**</**head**>**

**<**body**>**

**<**h1 style**=**"color: red"**>**¿Estas seguro de eliminar este registro**?</**h1**>**

**<%** String path **=** request**.**getContextPath**();** **%>**

**<%**

int id**=**Integer**.**parseInt**(**request**.**getParameter**(**"id"**));**

CatTipoUsuarioManager usc**=** **new** CatTipoUsuarioManager**();**

CatTipoUsuario e**=** usc**.**GetCatTipoUsuario**(**id**);**

**%>**

**<**form method**=**"post" action**=**"<%= path %>/EliminarTipoUsuario"**>**

**<**div class**=**"col-md-4" **>**

**<**input class**=**"form-control" readonly**=**"true" type**=**"text" name**=**"idS" id**=**"idS" placeholder**=**"id" value**=**"<%= e.getIdcattipoUsuario()%>" **/>**

**</**div**>**

**<**div class**=**"col-md-4" **>**

**<**input class**=**"form-control" readonly**=**"true" type**=**"text" name**=**"descripcion" placeholder**=**"Descripción" value**=**"<%= e.getDescripcion()%>"**/>**

**</**div**>**

**<**div class**=**"row col-md-12"**>**

**<**div class**=**"col-md-3"**>**

**<**button type**=**"submit" class**=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"1" name**=**"submitCatalogo"**>**

Regresar

**</**button**>**

**<**button type**=**"submit" class**=**"btn btn-primary" aria**-**label**=**"Left Align" value**=**"2" name**=**"submitCatalogo"**>**

Eliminar

**</**button**>**

**</**div**>**

**</**div**>**

**</**form**>**

**</**body**>**

**</**html**>**

A continuación, se generará el servlet *EliminarTipoUsuario.java* donde enviaremos el identificador que se desea eliminar, por consiguiente y para completar el MVC se enviará una redirección a la tabla principal con la eliminación del registro seccionado.

Como se muestra en el siguiente código:

package Controlador**;**

**import** Logica**.**CatTipoUsuarioManager**;**

**import** java**.**io**.**IOException**;**

**import** java**.**io**.**PrintWriter**;**

**import** jakarta**.**servlet**.**ServletException**;**

**import** jakarta**.**servlet**.**annotation**.**WebServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServlet**;**

**import** jakarta**.**servlet**.**http**.**HttpServletRequest**;**

**import** jakarta**.**servlet**.**http**.**HttpServletResponse**;**

@WebServlet**(**name **=** "EliminarTipoUsuario"**,** urlPatterns **=** **{**"/EliminarTipoUsuario"**})**

public class EliminarTipoUsuario **extends** HttpServlet **{**

protected void processRequest**(**HttpServletRequest request**,** HttpServletResponse response**)**

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

response**.**setContentType**(**"text/html;charset=UTF-8"**);**

**try** **(**PrintWriter out **=** response**.**getWriter**())** **{**

int valorBoton **=** Integer**.**parseInt**(**request**.**getParameter**(**"submitCatalogo"**));**

String idd**=**request**.**getParameter**(**"idS"**);**

int id **=** Integer**.**parseInt**(**idd**);**

**if** **(**valorBoton **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

CatTipoUsuarioManager registrar **=** **new** CatTipoUsuarioManager**();**

int registro **=** registrar**.**borrarCatTipoUsuario**(**id**);**

**if** **(**registro **==** 1**)** **{**

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/CatalogoTipoUsuarios.jsp"**);**

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

response**.**sendRedirect**(**request**.**getContextPath**()** **+** "/Vistas/Error.jsp"**);**

**}**

**}**

**}**

**}**

**}**

AJUSTES A LA SECCIÓN:

Terminar el CRUD para tipos catálogo de USUARIOS “Ejemplo de un CRUD MVC utilizando Servlets y JSP”, realizar:

1. Agregar la operación de CONSULTA utilizando MVC (JSP Y SERVLETS Y ACCESO A BD). Solo se tenía el JSP y mandaba de manera directa la consulta a la BD, integrar el Servlet.
2. Las operaciones de ALTA, BAJAS Y ACTUALIZACIÓN de tipos de usuarios ya están completas utilizando las tecnologías antes descritas (JSP y SERVLETS). Por lo que le pido realizar lo siguiente:
   1. Del libro llamado “Introducción a la Programación Web con Java: JSP y Servlets, JavaServer Faces” existen 3 secciones las cuales le pido revisar:
      1. **Paso de información de un JSP a un Servlet**, página 91.
      2. **Paso de información de un Servlet a un JSP**, página 94.
      3. **Comunicación entre Servlets y clases Java**, página 97. Esta sección habla de cómo acceder a una fuente de datos mediante clases Java que para nuestro caso es la BD.
   2. En el reporte detallado se van a integrar las mismas secciones ya que son determinantes en el desarrollo y entendimiento de un proyecto web en MVC, solo que aplicado al ejemplo de la tesina. El reporte debe incluir para todas las operaciones (consultas, altas, bajas y actualizaciones) para el CRUD de catálogo de usuarios.
   3. Por favor le pido se apegue a como se envían los datos desde el JSP hasta el acceso a los datos (pasando obviamente por los Servlets) y viceversa (regreso de datos al Servlet y pasando los datos al JSP), esto lo puede corroborar en las secciones del libro que le describí en un inciso anterior. Haga caso omiso a este inciso si el envió de parámetros entre JSP’s y Servlets (y viceversa) esta apegado a los lineamientos del libro, ya que es el estándar utilizado para paso de argumentos en una aplicación web en MVC (ya revisé los ejemplos del libro y son funcionales, solo que en el libro la fuente es un archivo).
3. En las secciones donde se tenga que agregar código fuente, utilizar un tipo de fuente diferente para distinguir del contenido temático de la tesina.
4. Las imágenes solo se utilizarán para imágenes alusivas a arquitecturas, esquemas, estructuras de un proyecto, gráficos, capturas de pantalla del sistema en función, etc.
5. Favor de no avanzar en los demás CRUD, si el CRUD de catálogo de usuarios no está apegado como se describe en el libro, si es lo contrario favor de continuar con los demás. Evidentemente habrá unas cosas que serán distintas (estilos, variables, etc) inclusive si se agregan demás paquetes (como el paquete negocio) no hay problema alguno, pero lo que debe ser similar es la estructura del proyecto, envió de parámetros entre JSP y Servlets (viceversa), voy a revisar a detalle lo que hasta el momento se realizó en sesión MEET del día 28 de Julio del 2021 para verificar la estructura y como se envían los parámetros entre JSP y SERVLETS.
6. Colocar una imagen alusiva a la estructura del proyecto en NETBEANS una vez terminado el CRUD de cattipodeusuarios.
7. Colocar una imagen representativa del funcionamiento correcto de cada una de las operaciones hechas a la tabla de catálogo de usuarios (representando el flujo de información entre vistas (JSP)).