

Programación Web con Java  
Tienda de Comercio Electrónico (eCommerce)  
Versión: 3.0 (20220724\_1955)

MSc. Juan Antonio Castro Silva

July 2022



# Índice general

<b>1. Iteración 1: Base de Datos</b>	<b>5</b>
1.1. Introducción . . . . .	5
1.2. Modelo Entidad Relación - MER . . . . .	6
1.3. Crear los usuarios – Login Roles . . . . .	6
1.4. Crear la base de datos . . . . .	8
1.5. Crear las tablas . . . . .	9
1.5.1. Tabla categories: . . . . .	9
1.5.2. Tabla products: . . . . .	10
1.5.3. Tabla users: . . . . .	11
1.5.4. Tabla reviews: . . . . .	11
1.6. Insertar los datos . . . . .	12
1.6.1. Categories: . . . . .	12
1.6.2. Products: . . . . .	14
1.6.3. Users: . . . . .	15
1.6.4. Reviews: . . . . .	16
1.7. Asignar permisos . . . . .	17
<b>2. Iteración 2: Servicios Web (Back-End)</b>	<b>19</b>
2.1. Introducción . . . . .	19
2.2. Creación de la aplicación web . . . . .	19
2.2.1. Crear proyecto . . . . .	19
2.2.2. Convertir el proyecto web dinámico a un proyecto Maven . . . . .	21
2.2.3. Configurar las fuentes de datos (DataSources) . . . . .	23
2.3. Crear paquetes . . . . .	25
2.4. CREACIÓN DE LAS CLASES . . . . .	26
2.5. Configurar los Servicios Web del proyecto . . . . .	36
2.6. Correr el proyecto . . . . .	36
2.7. Probar los Servicios Web . . . . .	38
2.7.1. Create: . . . . .	38
2.7.2. Read: . . . . .	40
2.7.3. Update: . . . . .	40
2.7.4. Delete: . . . . .	41
<b>3. Iteración 3: Servicios Web (Front-End)</b>	<b>43</b>
3.1. Implementar el CRUD . . . . .	43
3.1.1. Archivo: category/index.jsp . . . . .	45
3.1.2. Archivo: portal/menu.jsp . . . . .	47
3.1.3. Archivo: portal/footer.jsp . . . . .	47
3.1.4. Archivo: util/table_header.jsp . . . . .	48

3.1.5. Archivo: category/category_form.jsp . . . . .	49
3.1.6. Archivo: category/category_table.jsp . . . . .	50
3.1.7. Archivo: js/category.js (Parte 1) . . . . .	52
3.1.8. Archivo: js/category.js (Parte 2) . . . . .	53
3.1.9. Archivo: js/category.js (Parte 3) . . . . .	54
3.2. Correr el proyecto . . . . .	56
3.3. Probar los servicios web . . . . .	56
<b>4. Iteración 4: Componentes AJAX</b>	<b>59</b>
4.1. Identificar los componentes . . . . .	59
4.2. Crear los Servicios Web . . . . .	59
4.2.1. Clase org.software.portal/PortalCategoryServerService.java . . . . .	61
<b>5. Iteración 5: Seguridad con JAAS</b>	<b>63</b>
5.1. Introducción . . . . .	63
5.2. Encriptar las claves . . . . .	63
5.3. Crear las Tablas . . . . .	65
5.3.1. Tabla: profiles . . . . .	65
5.3.2. Tabla: user_profiles . . . . .	65
5.4. Alimentar la base de datos . . . . .	67
5.5. Asignar Permisos . . . . .	68
5.6. Archivo de configuración del proyecto (POM) . . . . .	69
5.6.1. Dependencias . . . . .	69
5.7. Configurar el Servicio de Autenticación y Autorización (JAAS) . . . . .	71
5.7.1. Clase: org.software.jaas.UserPrincipal.java . . . . .	71
5.7.2. Clase: org.software.jaas.RolePrincipal.java . . . . .	72
5.7.3. Clase: org.software.jaas.SoftwareLoginModule.java . . . . .	73
5.7.4. Clase: org.software.jaas.Logout.java . . . . .	78
5.8. Acceso a la Base de Datos . . . . .	79
5.8.1. Clase: org.software.util.DataBase.java . . . . .	79
5.8.2. Clase: org.software.util.DBUtil.java . . . . .	81
5.8.3. Archivo de Configuración: META-INF/context.xml . . . . .	82
5.8.4. Archivo de Configuración: TOMCAT-HOME/conf/jaas.config . . . . .	83
5.8.5. Configurar el archivo WEB-INF/web.xml de la aplicación. . . . .	84
5.9. Aplicación Web . . . . .	87
5.9.1. Componente Web: portal/menu.jsp . . . . .	87
5.9.2. Componente Web: login/login.jsp . . . . .	89
5.9.3. Componente Web: login/error.jsp . . . . .	90
5.9.4. Componente Web: portal/forbidden.jsp . . . . .	91
5.9.5. Componente Web: user/index.jsp . . . . .	92
5.9.6. Hoja de Estilo: css/login.css . . . . .	93
5.10. Correr la Aplicación (Tomcat-Local) . . . . .	94
5.11. Pruebas de Seguridad . . . . .	96
5.11.1. Prueba de login . . . . .	97
5.11.2. Prueba del perfil Cliente . . . . .	97
5.11.3. Prueba acceso no autorizado . . . . .	98
5.11.4. Prueba Logout . . . . .	98
5.11.5. Prueba de recurso protegido – asegurado . . . . .	99
5.11.6. Prueba de los Servicios Web . . . . .	100

# Capítulo 1

## Iteración 1: Base de Datos

### 1.1. Introducción

El proyecto de tienda de comercio electrónico (ecommerce) comprende varias iteraciones, esta primera fase orientada a los Servicios Web, consta de cuatro grandes iteraciones:

1. Base de Datos
2. Servicios Web - RESTful (Back-End)
3. Servicios Web - RESTful (Front-End)
4. Componentes Web

## 1.2. Modelo Entidad Relación - MER

En esta iteración se va a utilizar el motor de base de datos PostgreSQL y pgAdmin para crear una base de datos relacional, en otras iteraciones se utilizará MySQL y bases de datos no relacionales (MongoDB y Firebase).

El Diagrama Entidad Relación muestra las tablas necesarias para implementar la primera fase (versión) del proyecto.

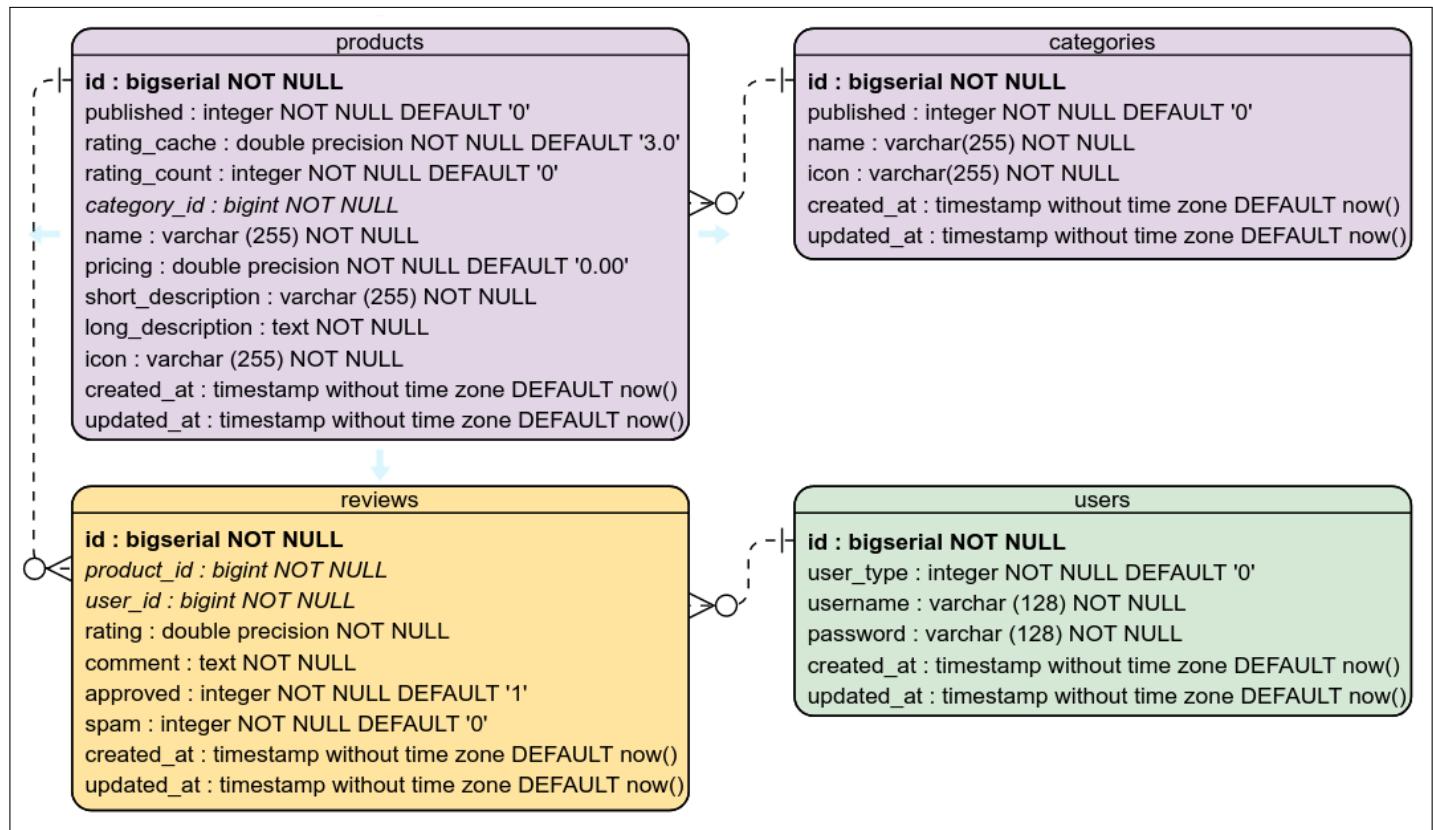


Figura 1.1: Diagrama Entidad Relación - DER

Para crear la base de datos abra pgAdmin desde una ventana de terminal – shell.

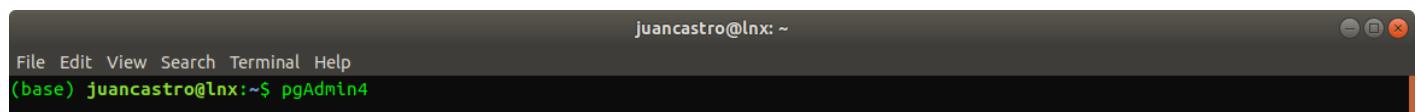


Figura 1.2: Abrir el pgAdmin

## 1.3. Crear los usuarios – Login Roles

Por cuestiones de seguridad el usuario postgres (DBA - DataBase Administrator), no se debe utilizar en las aplicaciones. Se debe crear un usuario propietario de la base de datos y otros para cada uno de los perfiles, en nuestro caso:

#	Perfil	Usuario	Clave	Descripción
1	Propietario	ecommerce	ecommerce	Propietario de la base de datos
2	Administrador	ecommerce_admin	234567	Administrador de una tienda
3	Cliente	ecommerce_client	345678	Usuario registrado
4	Invitado	ecommerce_guest	456789	Usuario no registrado

Para crear los usuarios (login roles) utilizando sentencias SQL, seleccione la base de datos [postgres] y haga clic arriba en el botón [Query Tool].

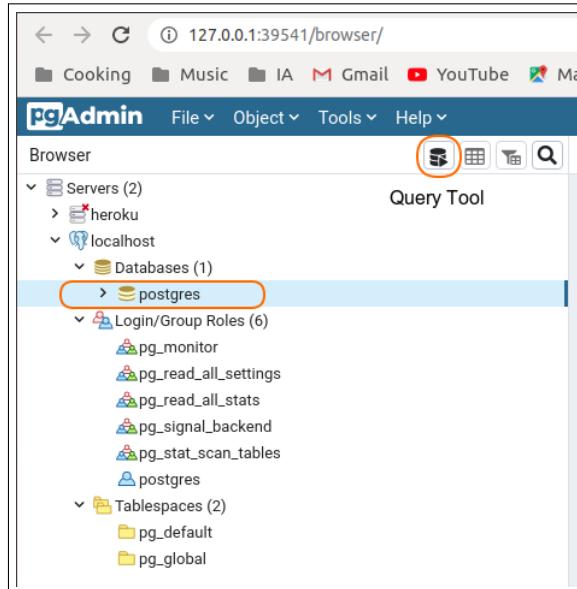


Figura 1.3: Query Tool

La sentencia SQL para crear un login role es:

```
CREATE ROLE ecommerce WITH LOGIN ENCRYPTED PASSWORD 'ecommerce';
```

En la ventana de [Query Tool] pegue o digite la sentencia SQL y haga clic arriba en el botón [Execute Query].

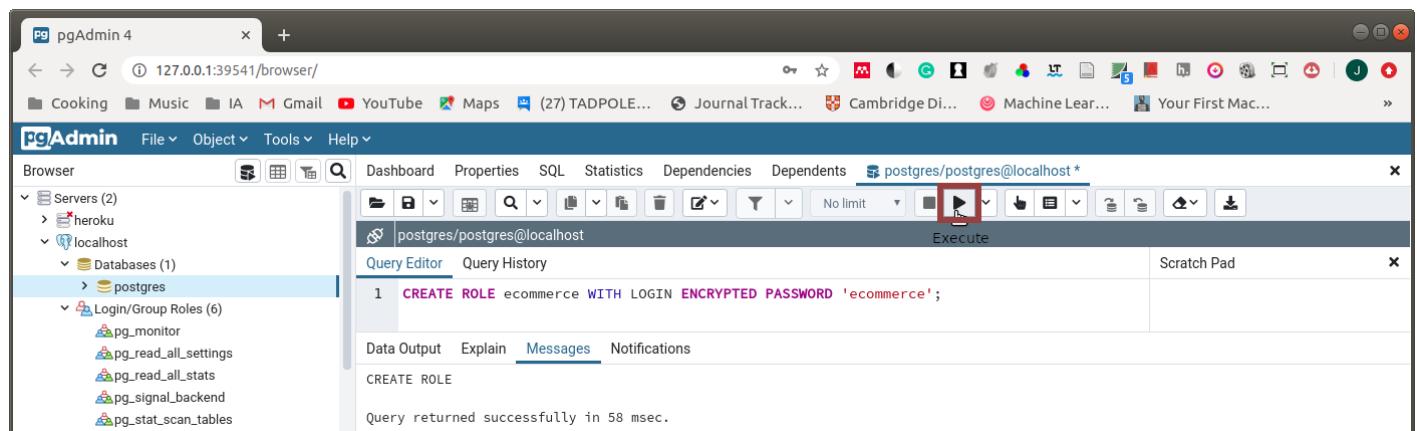


Figura 1.4: Execute Query

Para crear los perfiles restantes (ecommerce\_admin, ecommerce\_client y ecommerce\_guest), ejecute las sentencias SQL **CREATE ROLE** en la ventana de [Query Tool]:

```
CREATE ROLE ecommerce_admin WITH LOGIN ENCRYPTED PASSWORD '234567';
CREATE ROLE ecommerce_client WITH LOGIN ENCRYPTED PASSWORD '345678';
CREATE ROLE ecommerce_guest WITH LOGIN ENCRYPTED PASSWORD '456789';
```

Haga clic en la carpeta [Login Roles] con el botón derecho del mouse y seleccione [Refresh], para ver los usuarios creados.

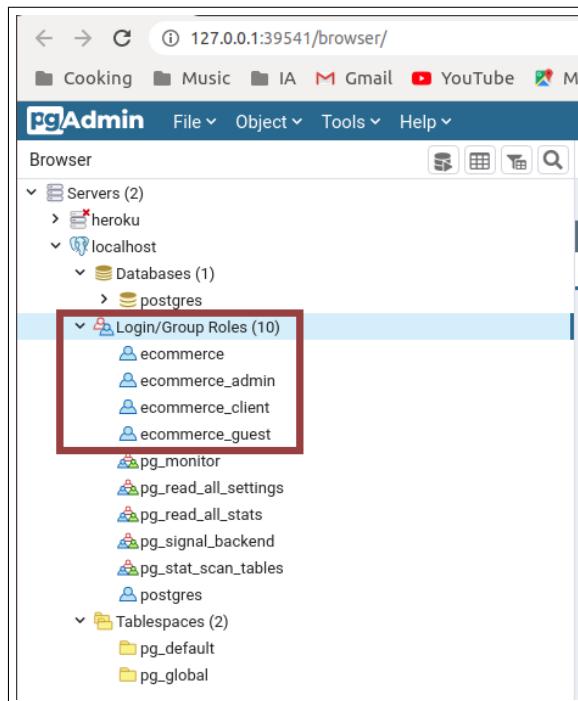


Figura 1.5: Crear usuarios - Login Roles

## 1.4. Crear la base de datos

Para crear la base datos, ejecute la sentencia SQL **CREATE DATABASE** en la ventana [Query Tool].

```
CREATE DATABASE ecommerce
WITH
OWNER = ecommerce
ENCODING = 'UTF8'
TABLESPACE = pg_default
CONNECTION LIMIT = -1;
```

En la ventana de [Query Tool] pegue o digite la sentencia SQL y haga clic arriba en el botón [Execute Query].

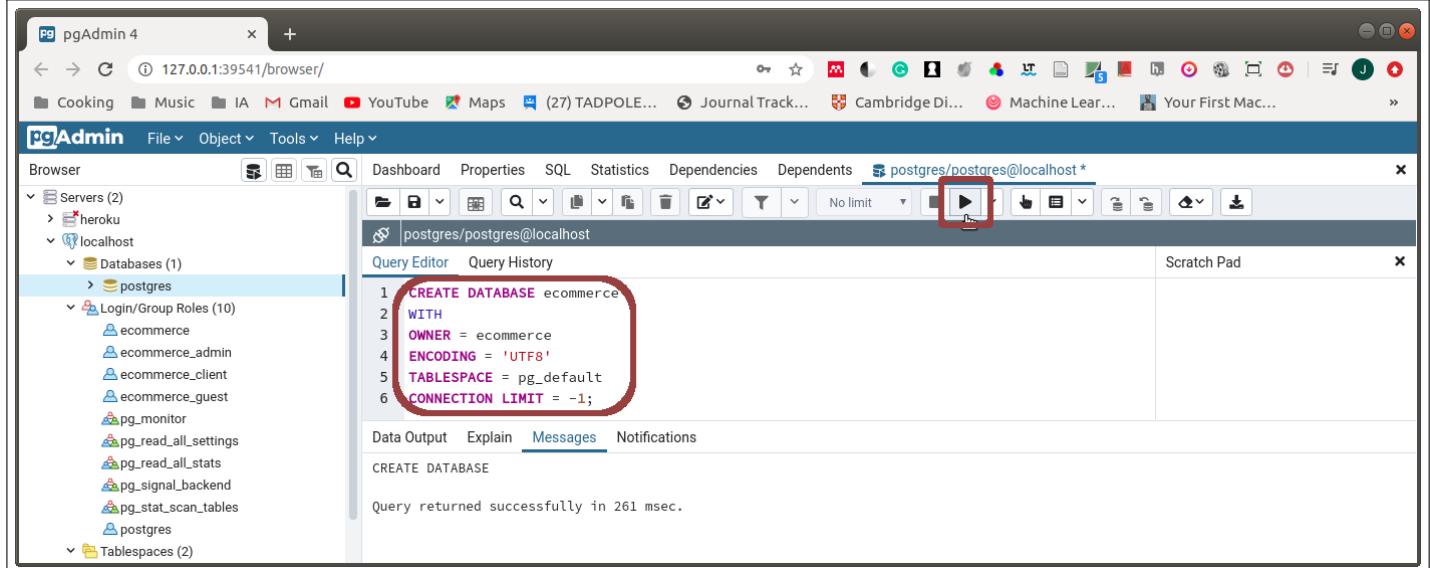


Figura 1.6: Create DataBase

Como resultado debe aparecer creada la base de datos (ecommerce).

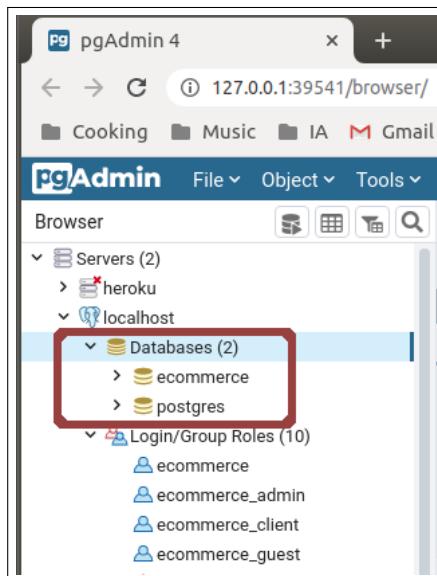


Figura 1.7: Create DataBase

## 1.5. Crear las tablas

Para crear las tablas, seleccione la base de datos (ecommerce) y haga clic en el botón [Query Tool] que se encuentra arriba en la barra de herramientas

### 1.5.1. Tabla categories:

Para crear la tabla categories, ejecute la sentencia **CREATE TABLE**:

```

CREATE TABLE categories (
    id bigserial NOT NULL,
    published integer NOT NULL DEFAULT '0',
    name varchar (255) NOT NULL,
    icon varchar (255) NOT NULL,
    created_at timestamp without time zone DEFAULT now(),
    updated_at timestamp without time zone DEFAULT now(),
    CONSTRAINT categories_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.categories OWNER TO ecommerce;

```

En la ventana [Query Tool] copie la sentencia SQL create table y haga clic en el botón de [Execute Query] que se encuentra arriba en la barra de herramientas, abajo en el panel de mensajes podrá ver el resultado de la ejecución.

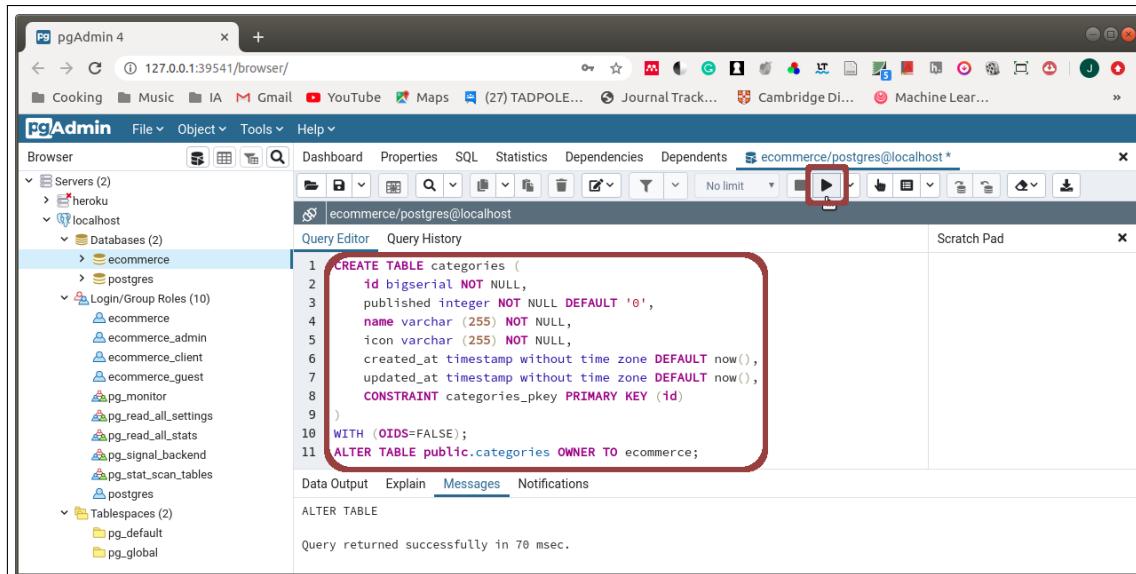


Figura 1.8: Crear tabla categories

Para crear las otras tablas de la base de datos, abra la ventana [Query Tool] y ejecute las sentencias SQL **CREATE TABLE**:

### 1.5.2. Tabla products:

La siguiente consulta SQL permite crear la tabla products.

```
CREATE TABLE products (
    id bigserial NOT NULL,
    published integer NOT NULL DEFAULT '0',
    rating_cache double precision NOT NULL DEFAULT '3.0',
    rating_count integer NOT NULL DEFAULT '0',
    category_id bigint NOT NULL,
    name varchar (255) NOT NULL,
    pricing double precision NOT NULL DEFAULT '0.00',
    short_description varchar (255) NOT NULL,
    long_description text NOT NULL,
    icon varchar (255) NOT NULL,
    created_at timestamp without time zone DEFAULT now (),
    updated_at timestamp without time zone DEFAULT now (),
    CONSTRAINT products_pkey PRIMARY KEY (id),
    CONSTRAINT products_category_id_fkey FOREIGN KEY (category_id)
        REFERENCES public.categories (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.products OWNER TO ecommerce;
```

### 1.5.3. Tabla users:

Para crear la tabla users ejecute la sentencia SQL **CREATE TABLE** en la ventana [Query Tool].

```
CREATE TABLE users (
    id bigserial NOT NULL,
    user_type integer NOT NULL DEFAULT '0',
    username varchar (128) NOT NULL,
    email varchar (128) NOT NULL,
    password varchar (128) NOT NULL,
    created_at timestamp without time zone DEFAULT now(),
    updated_at timestamp without time zone DEFAULT now(),
    CONSTRAINT users_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.users OWNER TO ecommerce;
```

### 1.5.4. Tabla reviews:

La siguiente consulta SQL permite crear la tabla reviews.

```
CREATE TABLE reviews (
    id bigserial NOT NULL,
    product_id bigint NOT NULL,
    user_id bigint NOT NULL,
    rating double precision NOT NULL,
    comment text NOT NULL,
    approved integer NOT NULL DEFAULT '1',
    spam integer NOT NULL DEFAULT '0',
    created_at timestamp without time zone DEFAULT now(),
    updated_at timestamp without time zone DEFAULT now(),
    CONSTRAINT reviews_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.reviews OWNER TO ecommerce;
```

Como resultado se deben haber creado las tablas.

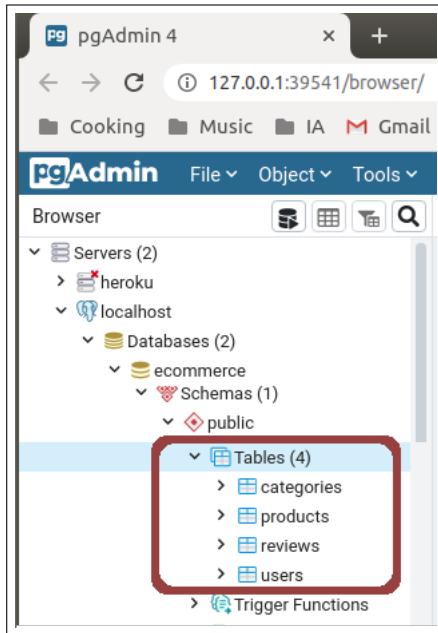


Figura 1.9: Tablas creadas

## 1.6. Insertar los datos

Seleccione la base de datos ecommerce, abra la ventana [Query Tool] y ejecute las sentencias SQL **INSERT** para inicializar los contenidos de la base de datos.

### 1.6.1. Categories:

Sentencias SQL para insertar los registros de la tabla categories.

```
INSERT INTO categories ("published", "name", "icon") VALUES
(1, 'Category One', 'category01.jpg'),
(1, 'Category Two', 'category02.jpg'),
(1, 'Category Three', 'category03.jpg'),
(1, 'Category Four', 'category04.jpg'),
(1, 'Category Five', 'category05.jpg');
```

Abra la ventana [Query Tool] y ejecute las sentencias SQL **INSERT**.

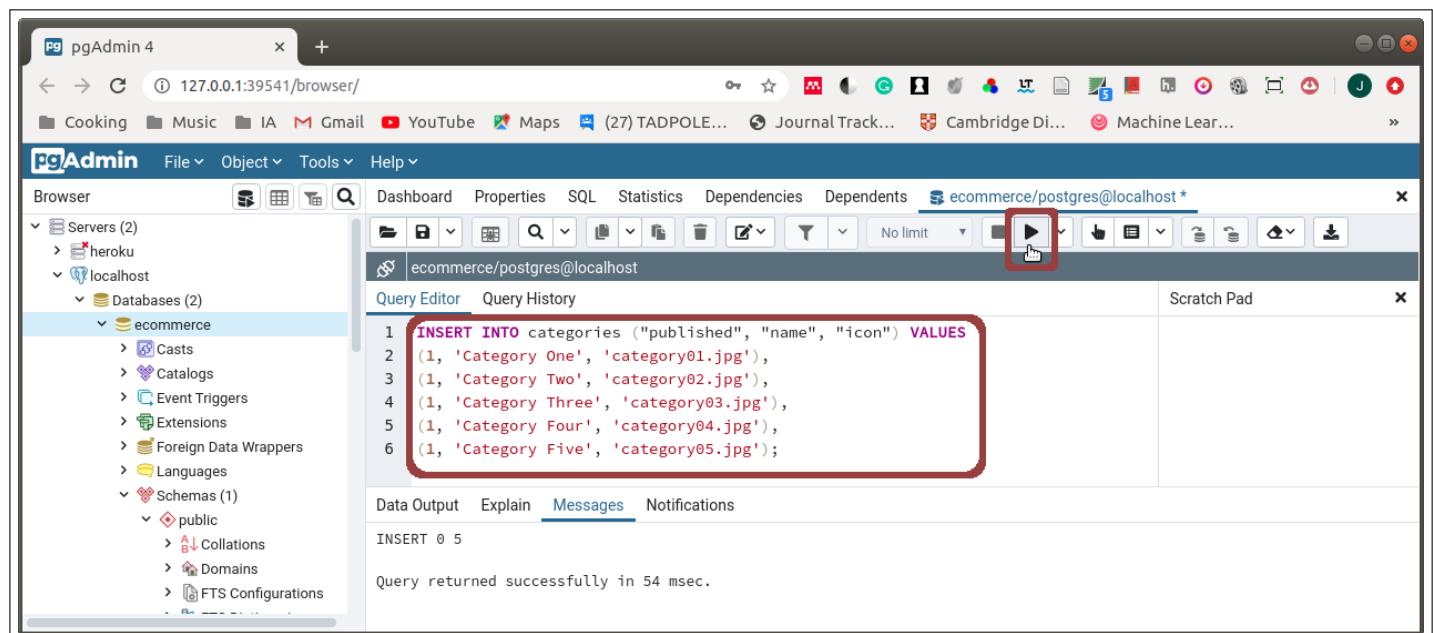


Figura 1.10: Insertar registros en la tabla categories

Abra una ventana de [Query Tool] y pruebe – consulte los datos insertados en la tabla categories, ejecutando la sentencia SQL **SELECT**.

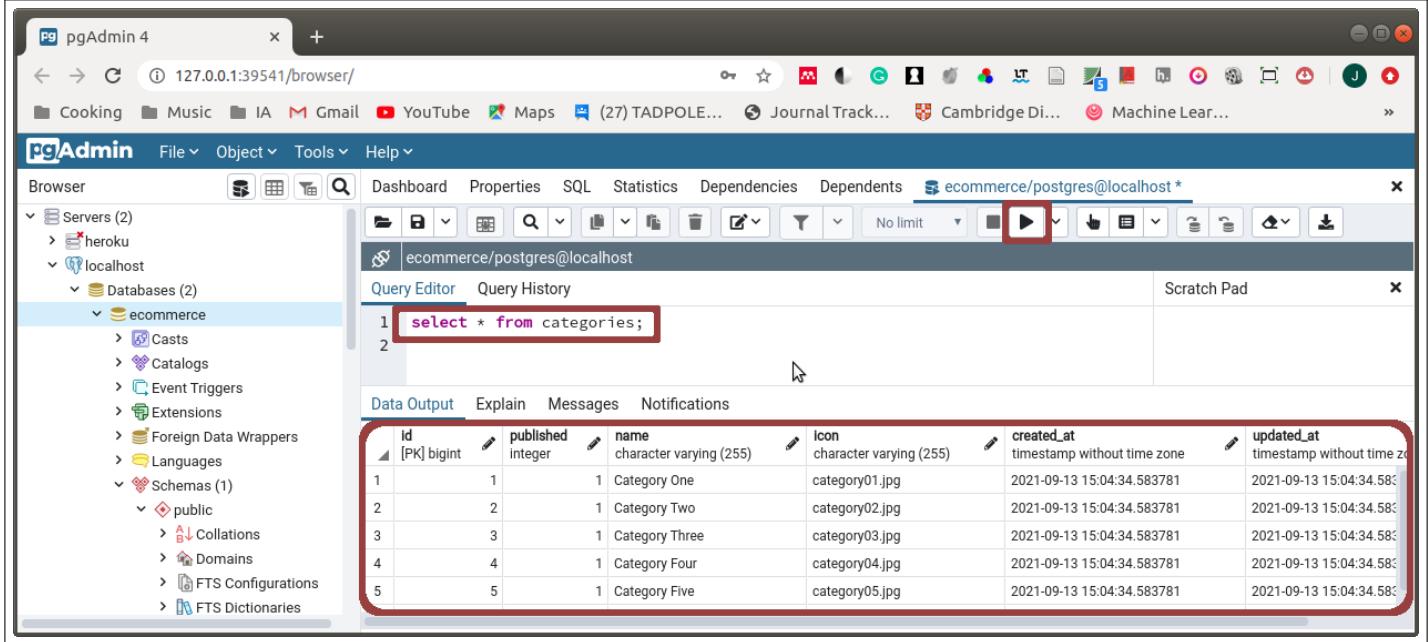


Figura 1.11: Consultar la tabla categories

### 1.6.2. Products:

para insertar los registros de la tabla products ejecute la siguiente consulta SQL en la ventana [Query Tool].

```
INSERT INTO products ("published", "rating_cache", "rating_count", "category_id", "name", "pricing", "short_description", "long_description", "icon") VALUES
(1, 3.0, 0, 1, 'First product', 20.99, 'Short description', 'Lorem ips ...',
'product01.jpg'),
(1, 3.0, 0, 1, 'Second product', 55.00, 'Short description', 'Lorem ips ...',
'product02.jpg'),
(1, 3.0, 0, 1, 'Third product', 65.00, 'Short description', 'Lorem ips ...',
'product03.jpg'),
(1, 3.0, 0, 1, 'Fourth product', 85.00, 'Short description', 'Lorem ips ...',
'product04.jpg'),
(1, 3.0, 0, 1, 'Fifth product', 95.00, 'Short description', 'Lorem ips ...',
'product05.jpg'),
(1, 3.0, 0, 2, 'Product 06', 35.00, 'Short description', 'Lorem ips ...',
'product06.jpg'),
(1, 3.0, 0, 2, 'Product 07', 45.00, 'Short description', 'Lorem ips ...',
'product07.jpg'),
(1, 3.0, 0, 3, 'Product 08', 52.00, 'Short description', 'Lorem ips ...',
'product08.jpg'),
(1, 3.0, 0, 3, 'Product 09', 62.00, 'Short description', 'Lorem ips ...',
'product09.jpg'),
(1, 3.0, 0, 4, 'Product 10', 14.00, 'Short description', 'Lorem ips ...',
'product10.jpg'),
(1, 3.0, 0, 4, 'Product 11', 18.00, 'Short description', 'Lorem ips ...',
'product11.jpg'),
(1, 3.0, 0, 5, 'Product 12', 40.00, 'Short description', 'Lorem ips ...',
'product12.jpg'),
(1, 3.0, 0, 5, 'Product 13', 44.00, 'Short description', 'Lorem ips ...',
'product13.jpg');
```

Para actualizar el campo long\_description de la tabla products, ejecute la siguiente consulta SQL.

```
UPDATE products SET long_description = 'Lorem ipsum dolor sit amet, consectetur adipisicing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam,
quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute
irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim
id est laborum';
```

Para consultar los registros de la tabla products, ejecute la sentencia SQL **SELECT**:

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' sidebar lists 'Servers (2)' (heroku and localhost) and 'localhost' databases (ecommerce). The 'ecommerce' database is selected. In the main area, the 'Query Editor' tab contains the SQL command: 'select \* from products;'. Below the editor, the 'Data Output' tab displays the results of the query:

	<b>Id</b>	<b>published</b>	<b>rating_cache</b>	<b>rating_count</b>	<b>category_id</b>	<b>name</b>	<b>pricing</b>	<b>short_description</b>
1	1	1	3	0	1	First product	20.99	Short description
2	2	1	3	0	1	Second product	55	Short description
3	3	1	3	0	1	Third product	65	Short description
4	4	1	3	0	1	Fourth product	85	Short description
5	5	1	3	0	1	Fifth product	95	Short description

Figura 1.12: Consultar la tabla products

### 1.6.3. Users:

Para insertar los registros de la tabla users, ejecute los siguientes consultas SQL en la ventana [Query Tool].

```
INSERT INTO users ("username", "email", "password") VALUES
('jose', 'jose.lopez@gmail.com', 'jlopez2018'),
('maria', 'maria.quintero@gmail.com', 'mquintero2018'),
('pedro', 'pedro.gutierrez@gmail.com', 'pgutierrez2018');
```

Para consultar los registros de la tabla users, ejecute la sentencia SQL **SELECT**:

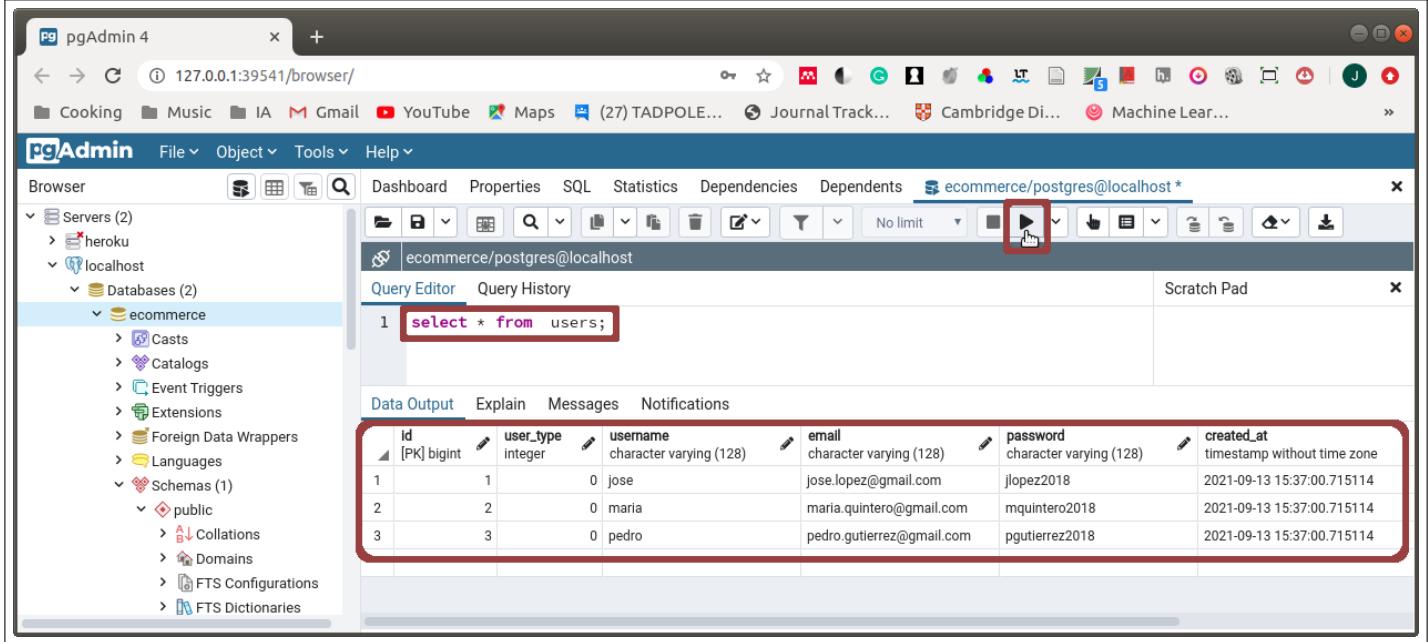


Figura 1.13: Consultar la tabla users

#### 1.6.4. Reviews:

Para insertar registros en la tabla reviews, ejecute los siguientes sentencias SQL en la ventana [Query Tool].

```
INSERT INTO reviews ("product_id", "user_id", "rating", "comment") VALUES
(1,1,4.5,'Comment 001'),
(1,2,3.5,'Comment 002'),
(1,3,4.0,'Comment 003'),
(2,1,2.5,'Comment 004'),
(2,2,3.5,'Comment 005'),
(2,3,3.0,'Comment 006'),
(3,1,4.0,'Comment 007'),
(3,2,3.5,'Comment 008'),
(3,3,2.0,'Comment 009'),
(4,1,5.0,'Comment 010'),
(4,2,3.0,'Comment 011'),
(4,3,3.5,'Comment 012'),
(5,1,3.0,'Comment 013'),
(5,2,4.5,'Comment 014'),
(5,3,4.0,'Comment 015');
```

Para consultar los registros de la tabla reviews, ejecute la sentencia SQL **SELECT**:

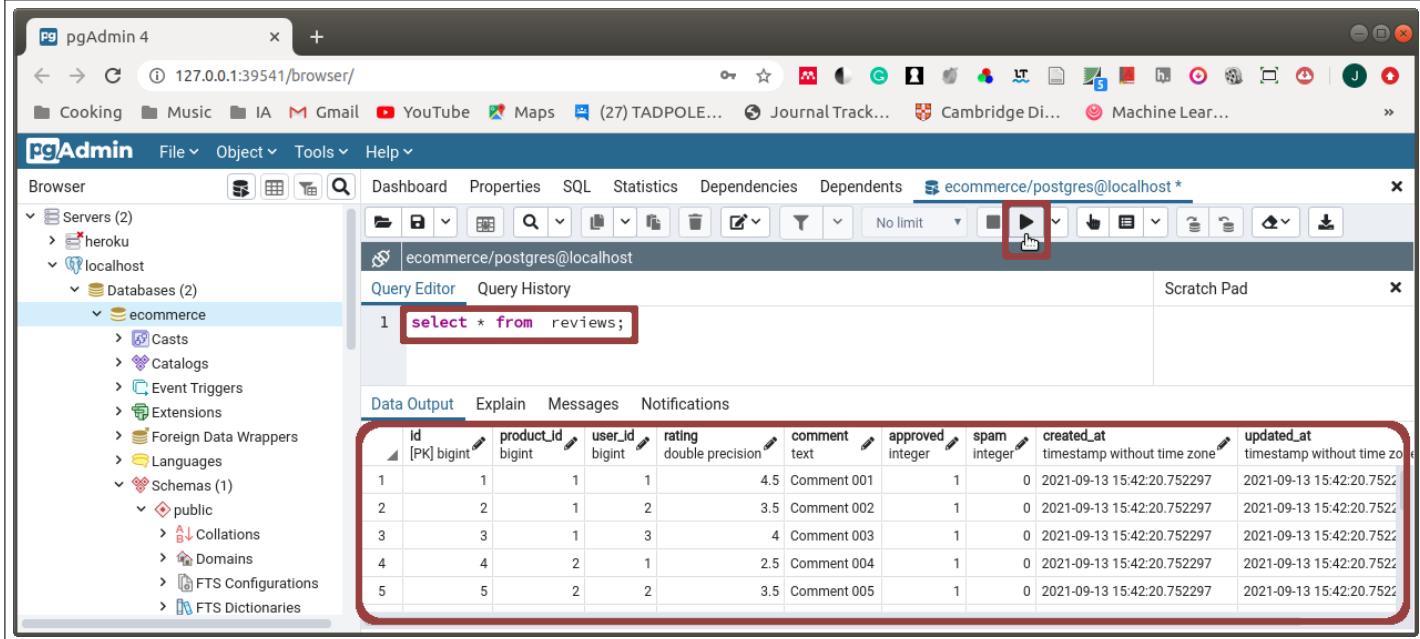


Figura 1.14: Consultar la tabla reviews

## 1.7. Asignar permisos

Para asignar los permisos, abra una ventana de [Query Tool] y ejecute la sentencia SQL grant. Los permisos de acceso para el usuario ecommerce\_admin, corresponden a las operaciones de gestión (consulta, inserción, modificación y eliminación de registros) propias del perfil **ADMINISTRADOR** (ecommerce\_admin).

```
grant select, insert, update, delete on categories to ecommerce_admin;
grant select, insert, update, delete on products to ecommerce_admin;
grant usage, select on all sequences in schema public to ecommerce_admin;
```

Para esta primera iteración los permisos de acceso para el usuario ecommerce\_client, corresponden al nivel de consulta (select).

```
grant select on categories to ecommerce_client;
grant select on products to ecommerce_client;
grant select on all sequences in schema public to ecommerce_client;
```

El usuario ecommerce\_guest, normalmente solo tiene permisos de consulta (select).

```
grant select on categories to ecommerce_guest;
grant select on products to ecommerce_guest;
grant select on all sequences in schema public to ecommerce_guest;
```

**NOTA:** Las tablas restantes y la parte de seguridad se implementarán en otras iteraciones.

# Capítulo 2

## Iteración 2: Servicios Web (Back-End)

### 2.1. Introducción

### 2.2. Creación de la aplicación web

Para este proyecto se creará una aplicación basada en web.

#### 2.2.1. Crear proyecto

Para crear un nuevo proyecto tipo web en eclipse, haga clic en el menú [File], [New] y en la opción [Dynamic Web Project].

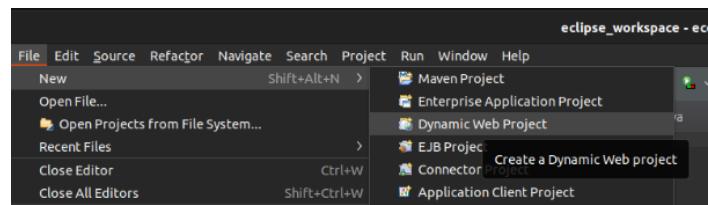


Figura 2.1: Crear Dynamic Web Project

Digite el nombre del proyecto (ecommerce), verifique el ambiente de ejecución objetivo, el servidor donde quiere correr la aplicación web [Target runtime] (Apache Tomcat v9.0) y haga clic abajo en el botón de [Next].

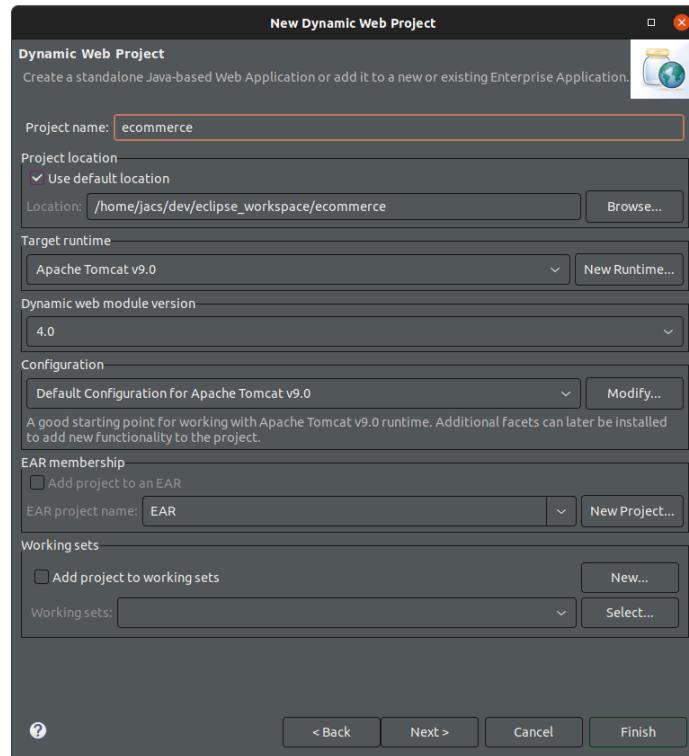


Figura 2.2: New Dynamic Web Project

Acepte los valores por defecto y haga clic abajo en el botón [Next].

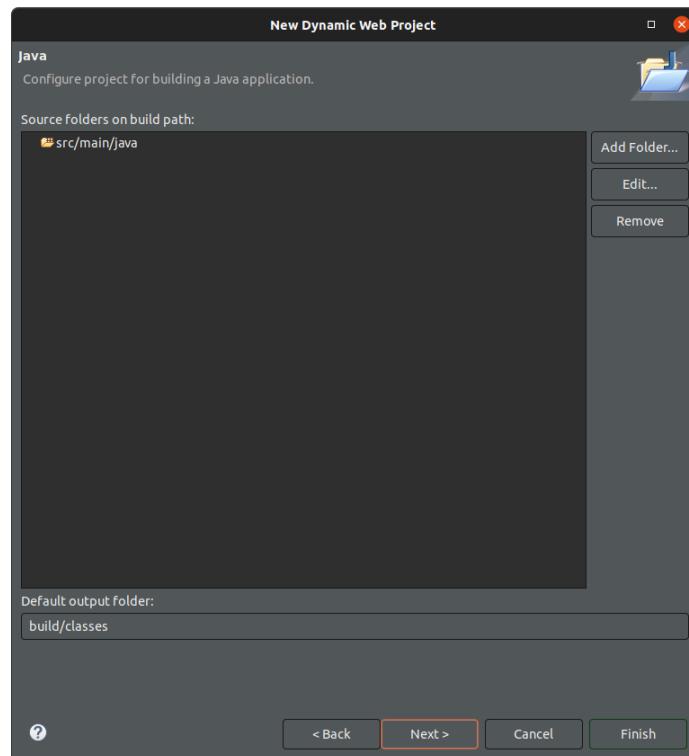


Figura 2.3: New Dynamic Web Project

Seleccione la opción [Generate web.xml deployment descriptor] y haga clic abajo en el botón [Finish].

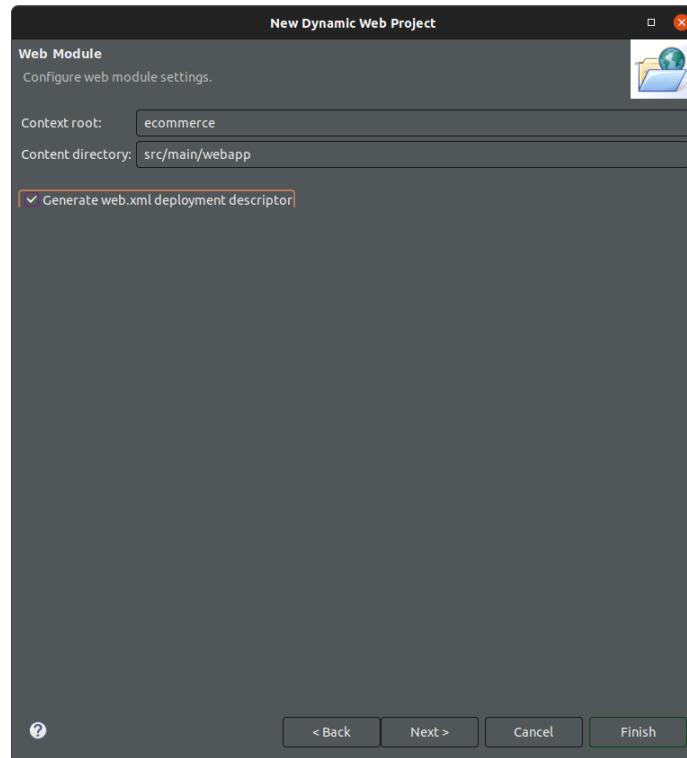


Figura 2.4: New Dynamic Web Project

Como resultado se ha creado el proyecto (ecommerce) en eclipse.

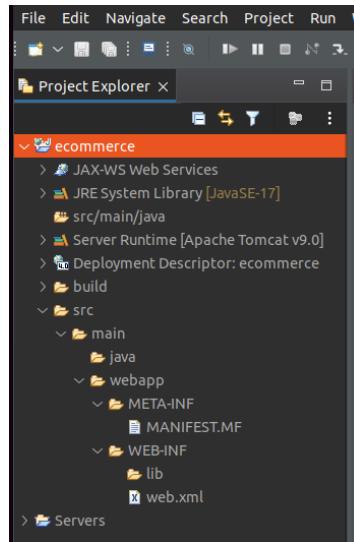


Figura 2.5: Proyecto ecommerce

## 2.2.2. Convertir el proyecto web dinámico a un proyecto Maven

Apache Maven es una herramienta de comprensión y gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM), Maven puede administrar la construcción, los informes y la documentación de un proyecto desde una pieza central de información. Para convertir el proyecto web dinámico, haga click con el botón derecho del mouse sobre el proyecto [ecommerce], abajo seleccione la opción [Configure] y finalmente haga click en [Convert to Maven Project].

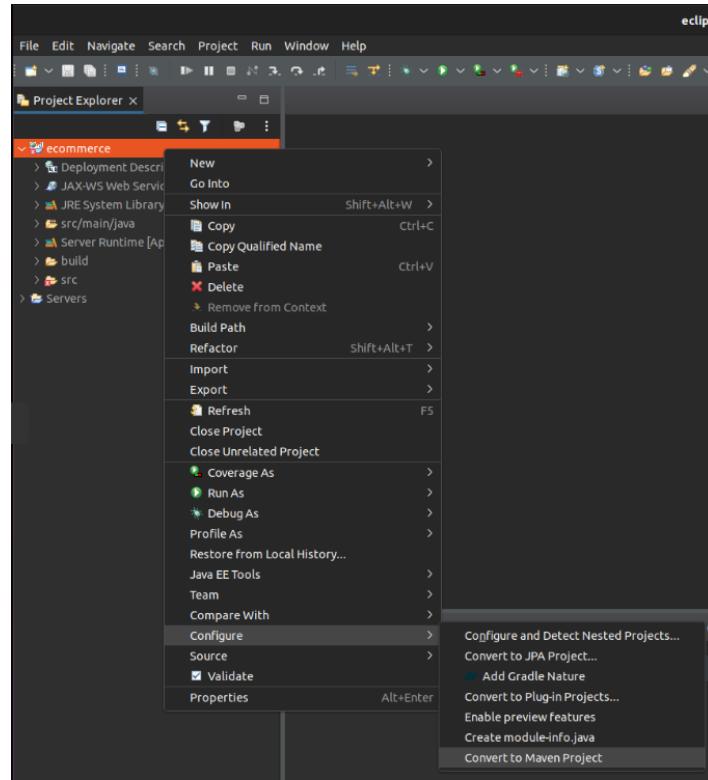


Figura 2.6: Convertir a un Proyecto Maven

En la ventana de [Create new POM] acepte los valores por defecto y haga click abajo a la derecha en el botón [Finish].

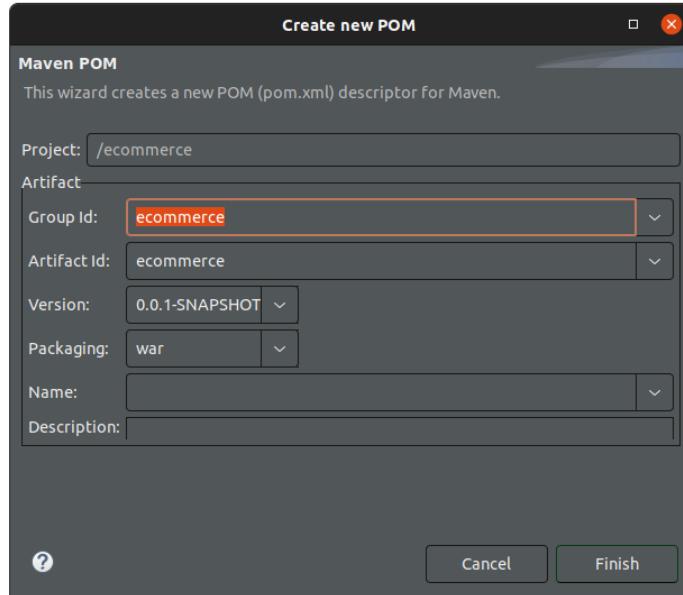


Figura 2.7: Crear un nuevo POM

Configure el archivo POM, el cual contiene la configuración del proyecto Maven. Para este caso se adiciona el nodo [dependencies] con la referencia a las librerías requeridas para trabajar con Servicios Web de tipo RestFUL, el formato de intercambio JSON y el driver de la base de datos de PostgreSQL.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>ecommerce</groupId>
  <artifactId>ecommerce</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <dependencies>
    <dependency>
      <groupId>org.glassfish.jersey.containers</groupId>
      <artifactId>jersey-container-servlet</artifactId>
      <version>2.29.1</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.inject</groupId>
      <artifactId>jersey-hk2</artifactId>
      <version>2.29.1</version>
    </dependency>
    <dependency>
      <groupId>org.glassfish.jersey.media</groupId>
      <artifactId>jersey-media-json-jackson</artifactId>
      <version>2.29.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
    <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.2.5</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
          <release>17</release>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.2.3</version>
      </plugin>
    </plugins>
  </build>
</project>
```

Listing 2.2.1: Archivo: pom.xml

### 2.2.3. Configurar las fuentes de datos (DataSources)

Para configurar los DataSources debe crear un archivo de contexto [context.xml]. Haga click con el botón derecho del mouse sobre la carpeta [META-INF], seleccione la opción [New] y finalmente haga click en [File].

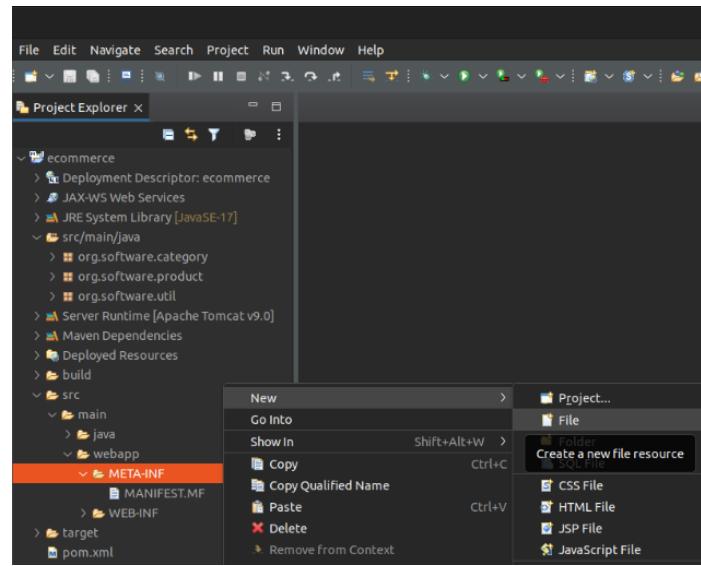


Figura 2.8: Crear nuevo archivo - context.xml

Digite el nombre del archivo [context.xml] y haga click abajo a la derecha en el botón [Finish].

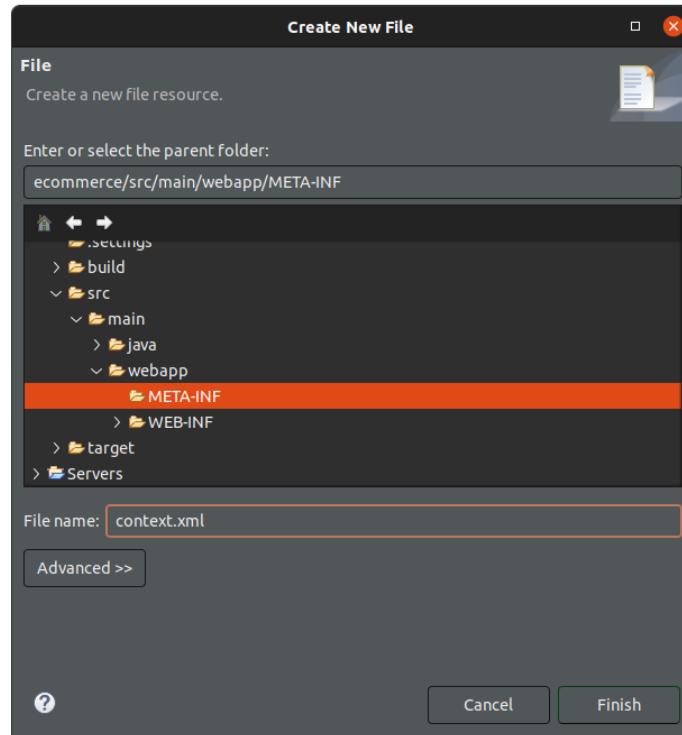


Figura 2.9: Archivo de contexto

En el archivo de contexto se definen las fuentes de datos (datasources), donde se especifica el nombre, el usuario de la base de datos y su clave, el nombre de la clase que implementa el driver y la url donde se encuentra la base de datos.

```
<Context>
    <Resource name="jdbc/eCommerceAdminDS"
        auth="Container" type="javax.sql.DataSource"
        maxTotal="50" maxIdle="30"
        maxWaitMillis="10000" username="ecommerce_admin"
        password="234567" driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/ecommerce"/>
    <Resource name="jdbc/eCommerceClientDS"
        auth="Container" type="javax.sql.DataSource"
        maxTotal="50" maxIdle="30"
        maxWaitMillis="10000" username="ecommerce_client"
        password="345678" driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/ecommerce"/>
    <Resource name="jdbc/eCommerceGuestDS"
        auth="Container" type="javax.sql.DataSource"
        maxTotal="50" maxIdle="30"
        maxWaitMillis="10000" username="ecommerce_guest"
        password="456789" driverClassName="org.postgresql.Driver"
        url="jdbc:postgresql://localhost:5432/ecommerce"/>
</Context>
```

Listing 2.2.2: Archivo: context.xml

## 2.3. Crear paquetes

La creación de las clases y paquetes es una implementación del Diagrama de Clases. Para esta iteración el diagrama de clases comprende los siguientes paquetes y clases.

### 1. org.software.category

- Category
- CategoryList
- CategoryService

### 2. org.software.product

- Product
- ProductList
- ProductService

### 3. org.software.util

- DataBase

Para crear un paquete haga clic con el botón derecho en la carpeta [src], seleccione la opción [New] y finalmente haga clic en [Package].

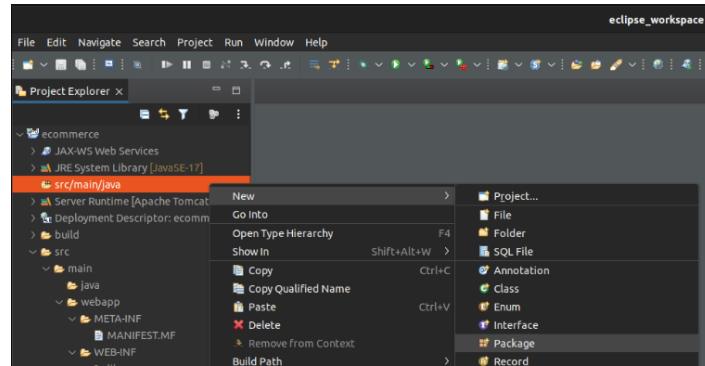


Figura 2.10: Crear un nuevo package

Digite el nombre del paquete [org.software.category] y haga click abajo en el botón [Finish].

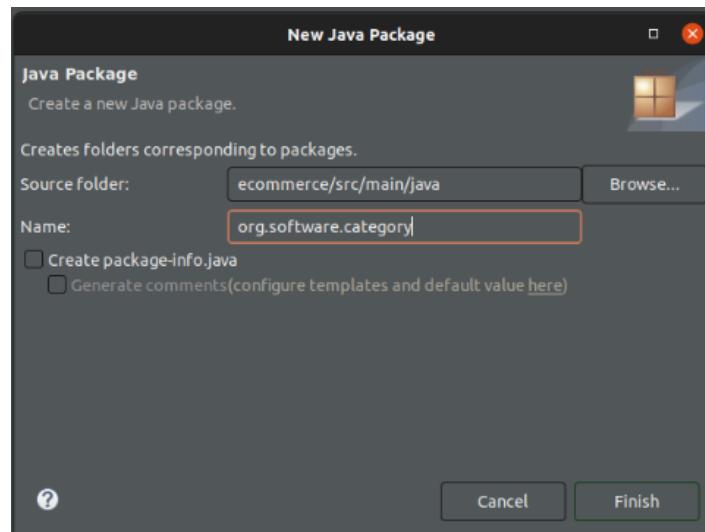


Figura 2.11: New Package

Los paquetes creados se muestran en la carpeta [src/main/java].

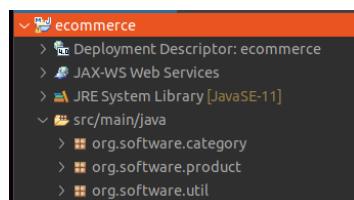


Figura 2.12: Packages

## 2.4. CREACIÓN DE LAS CLASES

Dentro del paquete org.software.category debe crear tres clases:

- Category
- CategoryList

- CategoryService

Para crear una clase haga click con el botón derecho del mouse en el paquete [org.software.category], seleccione [New] y finalmente haga click en [Class].

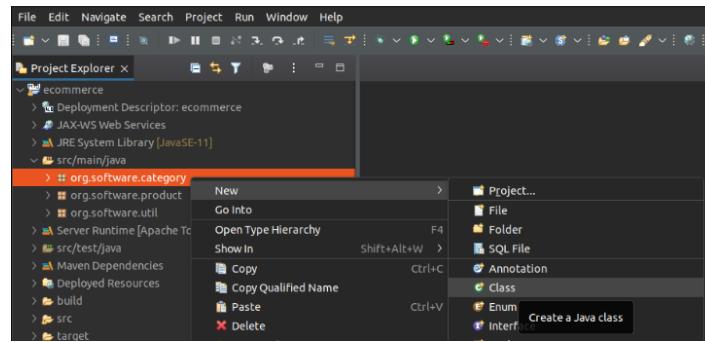


Figura 2.13: Packages

En la ventana [New Java Class], en el cajón de texto [Name] digite el nombre de la clase(Category) y haga click abajo en el botón [Finish].

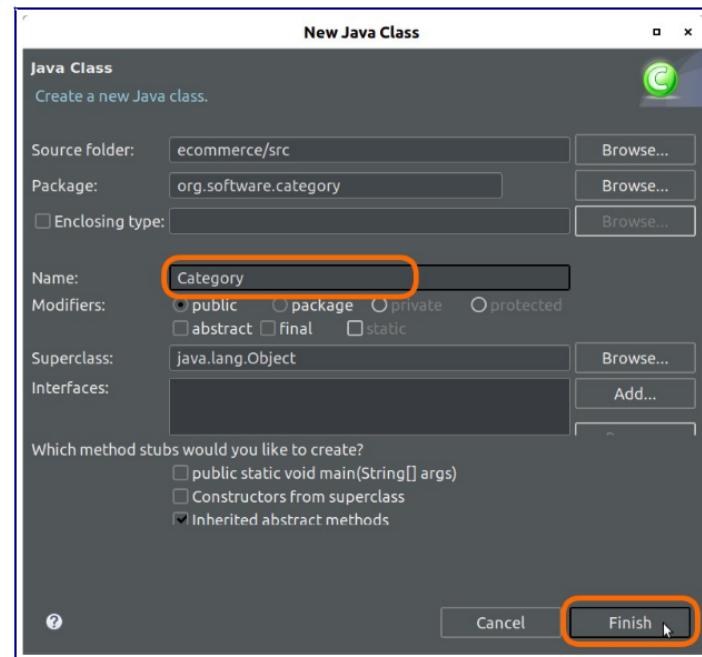


Figura 2.14: Classes

Dentro de la carpeta [src] podrá ver los paquetes y clases creadas, correspondientes al Diagrama de Clases.

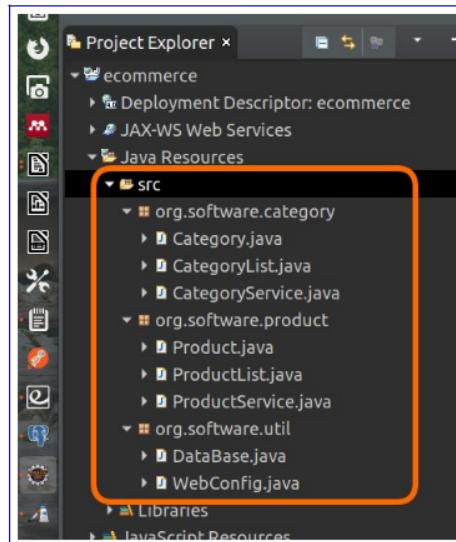


Figura 2.15: Classes

El código fuente de las clases se muestra a continuación:

#### Clase: org.software.util.DataBase

```
package org.software.util;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class DataBase {
    public Connection getConnection(String profile) {
        Connection connection = null;

        String JndiDataSourceName = "";
        if (profile.equals("admin")) {
            JndiDataSourceName = "eCommerceAdminDS";
        }
        if (profile.equals("client")) {
            JndiDataSourceName = "eCommerceClientDS";
        }
        if (profile.equals("guest")) {
            JndiDataSourceName = "eCommerceGuestDS";
        }
        System.out.println("JndiDataSourceName: " + JndiDataSourceName);
        try {
            Context ctx = new InitialContext();
            DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/" + JndiDataSourceName);
            connection = ds.getConnection();
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
        return connection;
    }
}
```

Listing 2.4.1: Clase: org.software.util.DataBase.java (Parte 1)

**Clase: org.software.util.DataBase**

```
public void closeObject(Connection connection) {
    try {
        connection.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void closeObject(PreparedStatement preparedStatement) {
    try {
        preparedStatement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void closeObject(Statement statement) {
    try {
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void closeObject(ResultSet resultSet) {
    try {
        resultSet.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Listing 2.4.2: Clase: org.software.util.DataBase.java (Parte 2)

**Clase: org.software.category.Category**

```
package org.software.category;

import java.sql.Statement;

public class Category {
    private long id;
    private int published;
    private String name;
    private String icon;

    public Category() {
        super();
    }

    public Category(long id, int published, String name, String icon) {
        super();
        this.id = id;
        this.published = published;
        this.name = name;
        this.icon = icon;
    }

    public long getId() {
        return id;
    }

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

    public int getPublished() {
        return published;
    }

    public void setPublished(int published) {
        this.published = published;
    }

    public String getName() {
        return name;
    }

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

    public String getIcon() {
        return icon;
    }

    public void setIcon(String icon) {
        this.icon = icon;
    }
}
```

Listing 2.4.3: Clase: org.software.category.Category.java

**Clase: org.software.category.CategoryList**

```
package org.software.category;

import java.util.List;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "categories")
public class CategoryList {
    private List<Category> items;

    public CategoryList() {
    }

    public CategoryList(List<Category> items) {
        this.items = items;
    }

    @XmlElement(name = "data")
    public List<Category> getItems() {
    }
}
```

Listing 2.4.4: Clase: org.software.category.CategoryList.java

**Clase: org.software.category.CategoryService**

```
package org.software.category;

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

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

import org.software.util.DataBase;

@Path("/category")
public class CategoryService {
    @POST
    @Path("/")
    @Consumes({ "application/json" })
    @Produces("application/json")
    public Response add(Category category) {
        DataBase database = new DataBase();
        Connection connection1 = null;
        PreparedStatement preparedStatement1 = null;
        String sql = "";
        String mensaje = "";
        int inserteds = 0;
        try {
            connection1 = database.getConnection("admin");
            sql = "INSERT INTO categories(name, icon)";
            sql += " VALUES (?, ?)";
            preparedStatement1 = connection1.prepareStatement(sql);
            preparedStatement1.setString(1, category.getName());
            preparedStatement1.setString(2, category.getIcon());
            inserteds = preparedStatement1.executeUpdate();
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        } finally {
            database.closeObject(preparedStatement1);
            database.closeObject(connection1);
        }
        if (inserteds > 0) {
            mensaje = "{\"mensaje\":\"Adicionar OK\"}";
            return Response.status(200).entity(mensaje).build();
        } else {
            mensaje = "{\"mensaje\":\"Error al adicionar\"}";
            return Response.status(400).entity(mensaje).build();
        }
    }
}
```

Listing 2.4.5: Clase: org.software.category.CategoryService.java (Part 1)

**Clase: org.software.category.CategoryService**

```
@GET  
@Path("/")  
@Produces("application/json")  
// @Produces("application/xml")  
public CategoryList getAll() {  
    ArrayList<Category> categoryList = new ArrayList<Category>();  
    DataBase database = new DataBase();  
    Connection connection1 = null;  
    Statement statement1 = null;  
    ResultSet rs1 = null;  
    String sql = "";  
    try {  
        connection1 = database.getConnection("admin");  
        statement1 = connection1.createStatement();  
        sql = "select * from categories";  
        rs1 = statement1.executeQuery(sql);  
        while (rs1.next()) {  
            int id = rs1.getInt("id");  
            int published = rs1.getInt("published");  
            String name = rs1.getString("name");  
            String icon = rs1.getString("icon");  
            Category category = new Category();  
            category.setId(id);  
            category.setPublished(published);  
            category.setName(name);  
            category.setIcon(icon);  
            categoryList.add(category);  
        }  
    } catch (Exception e) {  
        System.out.println("Error: " + e.toString());  
    } finally {  
        database.closeObject(rs1);  
        database.closeObject(statement1);  
        database.closeObject(connection1);  
    }  
    return new CategoryList(categoryList);  
}
```

Listing 2.4.6: Clase: org.software.category.CategoryService.java (Part 2)

## Clase: org.software.category.CategoryService

```

93  @PUT
94  @Path("/{id}")
95  @Consumes({ "application/json" })
96  @Produces("application/json")
97  public Response update(Category category, @PathParam(value = "id") int id) {
98      DataBase database = new DataBase();
99      Connection connection1 = null;
100     PreparedStatement preparedStatement1 = null;
101     String sql = "";
102     String mensaje = "";
103     int updates = 0;
104     try {
105         connection1 = database.getConnection("admin");
106         sql = "UPDATE categories SET published=?, name=?, icon=?";
107         sql += " WHERE id=?";
108         preparedStatement1 = connection1.prepareStatement(sql);
109         preparedStatement1.setInt(1, category.getPublished());
110         preparedStatement1.setString(2, category.getName());
111         preparedStatement1.setString(3, category.getIcon());
112         preparedStatement1.setInt(4, id);
113         updates = preparedStatement1.executeUpdate();
114     } catch (Exception e) {
115         System.out.println("Error: " + e.toString());
116     } finally {
117         database.closeObject(preparedStatement1);
118         database.closeObject(connection1);
119     }
120     if (updates > 0) {
121         mensaje = "{\"mensaje\":\"Modificar OK\"}";
122         return Response.status(200).entity(mensaje).build();
123     } else {
124         mensaje = "{\"mensaje\":\"Error al modificar\"}";
125         return Response.status(400).entity(mensaje).build();
126     }
127 }
128
129 @DELETE
130 @Path("/{id}")
131 @Consumes({ "application/json" })
132 @Produces("application/json")
133 public Response adicionar(@PathParam(value = "id") int id) {
134     DataBase database = new DataBase();
135     Connection connection1 = null;
136     PreparedStatement preparedStatement1 = null;
137     String sql = "";
138     String mensaje = "";
139     int deleteds = 0;
140     try {
141         connection1 = database.getConnection("admin");
142         sql = "DELETE FROM categories WHERE id=?";
143         preparedStatement1 = connection1.prepareStatement(sql);
144         preparedStatement1.setInt(1, id);
145         deleteds = preparedStatement1.executeUpdate();
146     } catch (Exception e) {
147         System.out.println("Error: " + e.toString());
148     } finally {
149         database.closeObject(preparedStatement1);
150         database.closeObject(connection1);
151     }
152     if (deleteds > 0) {
153         mensaje = "{\"mensaje\":\"Eliminar OK\"}";
154         return Response.status(200).entity(mensaje).build();
155     } else {
156         mensaje = "{\"mensaje\":\"Error al eliminar\"}";
157         return Response.status(400).entity(mensaje).build();
158     }
159 }

```

Listing 2.4.7: Clase: org.software.category.CategoryService.java (Part 3)

## 2.5. Configurar los Servicios Web del proyecto

Para configurar los Servicios Web debe modificar el archivo de configuración del proyecto [web.xml] y adicionar los nodos servlet y servlet-mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
  <display-name>ecommerce</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>Ecommerce REST Service</servlet-name>
    <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>org.software</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Ecommerce REST Service</servlet-name>
    <url-pattern>/ws/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Listing 2.5.1: Archivo: web.xml

## 2.6. Correr el proyecto

Para correr el proyecto, haga clic con el botón derecho del mouse sobre el proyecto (ecommerce), menú [Run As] y seleccione la opción [Run on Server].

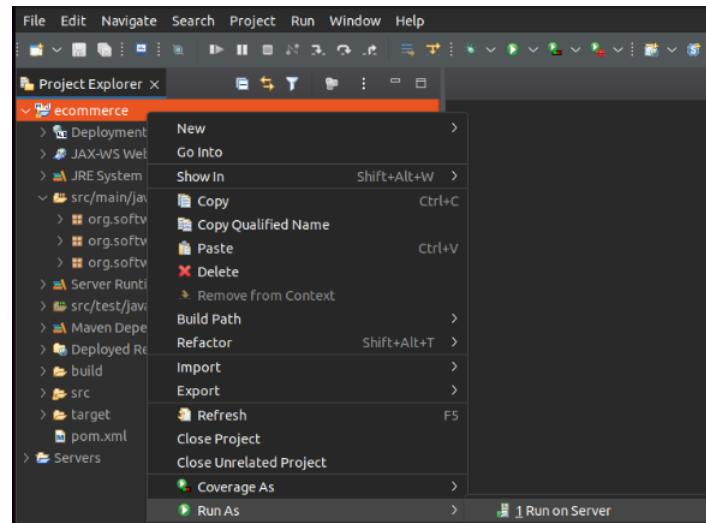


Figura 2.16: Ejecutar el proyecto

Verifique el servidor de aplicaciones donde desea correr la aplicación web [Runtime Server] y haga clic abajo en el botón [Next].

Para probar la operación de lectura del CRUD (Read - método GET), en un navegador abra la url (<http://localhost:8080/ecommerce/ws/category>). Podrá ver el listado de todos los registros de la tabla categories en formato JSON.

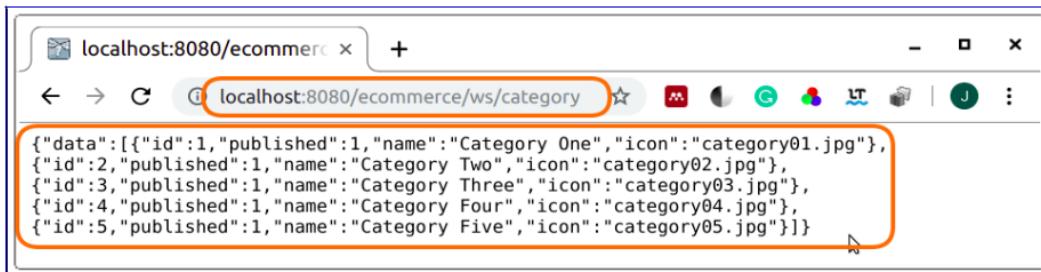


Figura 2.17: Probar el método Read - GET

## 2.7. Probar los Servicios Web

La prueba de los servicios web la podemos hacer de diversas formas, para este proyecto se utilizará la aplicación Postman. Descargue el instalador de Postman desde la página web oficial: (<https://www.getpostman.com/downloads/>).

Existen instaladores para diversos sistemas operativos.

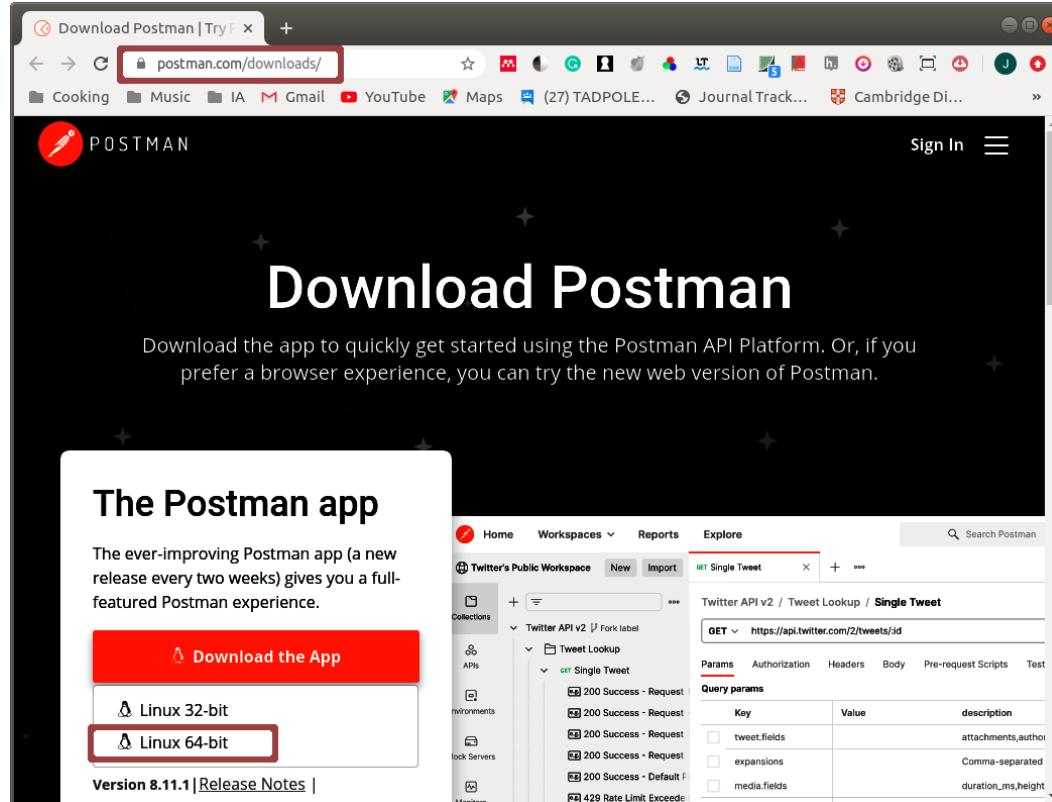


Figura 2.18: Descargar Postman

Para instalar Postman abra una ventana de Terminal – Shell, ingrese a la carpeta donde lo descargó y ejecute el comando (tar – xvzf) para descomprimirlo. Para ejecutar Postman, ingrese a la carpeta donde lo instaló y haga clic en el archivo ejecutable [Postman]. Para utilizar Postman debe crear una cuenta de usuario. Digite los datos requeridos y haga clic en el botón [Create free account].

### 2.7.1. Create:

Para probar la operación **Create** (Add), seleccione el método POST, digite la url (<http://localhost:8080/ecommerce/ws/category>), seleccione la ficha [Body], botón de radio [Raw], el formato de texto [JSON (application/json)]. Digite el texto en formato JSON con los valores que desea adicionar:

```
{"name": "Food", "icon": "food.png"}
```

Y haga clic en el botón [Send]. Abajo en la ficha [Body] podrá ver la respuesta del servidor en formato JSON.

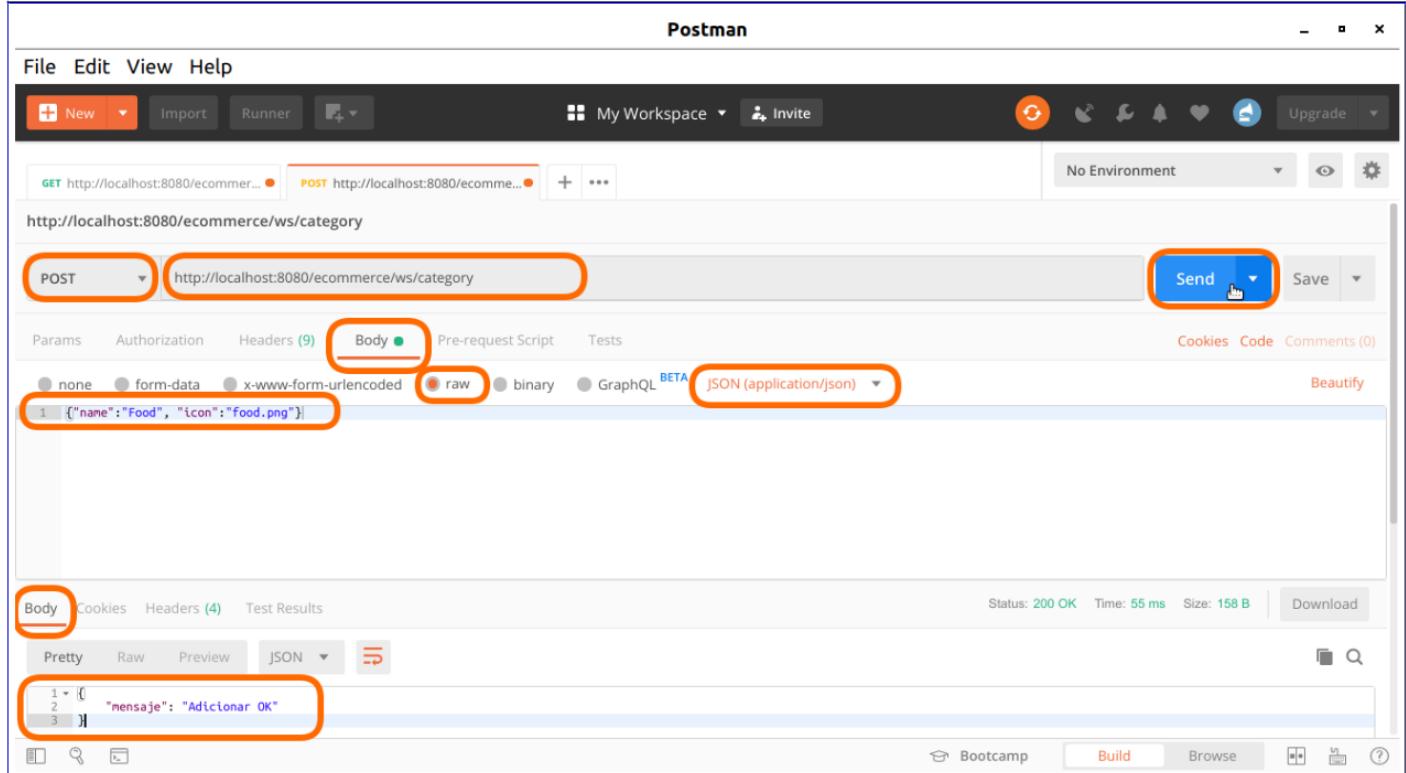


Figura 2.19: Descargar Postman

Para verificar el resultado de la operación Create en el administrador de la base de datos pgAdmin, seleccione la tabla [categories] de la base de datos [ecommerce] y haga click arriba en el botón [View Data].

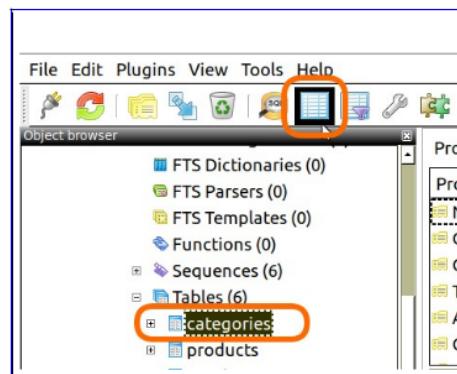


Figura 2.20: Seleccionar la tabla categories - pgAdmin4

Se puede ver que el ultimo registro creado tiene el name: Food y el icon: food.png.

	<b>[PK] bigserial</b>	<b>published</b>	<b>name</b> <b>character varying(255)</b>	<b>icon</b> <b>character varying(255)</b>	<b>created_at</b> <b>timestamp without time zone</b>	<b>updated_at</b> <b>timestamp without time zone</b>
1	1	1	Category One	categoryv01.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
2	2	1	Category Two	categoryv02.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
3	3	1	Category Three	categoryv03.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
4	4	1	Category Four	categoryv04.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
5	5	1	Category Five	categoryv05.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
6	6	0	Food	food.png	2019-07-14 16:11:51.972782	2019-07-14 16:11:51.972782

Figura 2.21: Ver cambios en la base de datos - Create

## 2.7.2. Read:

Para probar la opción Read (Consultar), seleccione el método GET, digite la url (<http://localhost:8080/ecommerce/ws/category>), haga click en el botón [Send]. En la parte de abajo en la ficha [Body] podrá ver los resultados enviados por el servidor.

```

GET http://localhost:8080/ecommerce/ws/category
[{"id": 1, "published": 1, "name": "Category One", "icon": "categoryv01.jpg"}]
  
```

Figura 2.22: Probar Servicio Web en Postman - Read

## 2.7.3. Update:

Para probar la operación Update, seleccione el método PUT, digite la url (<http://localhost:8080/ecommerce/ws/category/1>), seleccione la ficha [Body], botón de radio [Raw], el formato de texto [JSON (application/json)]. Digite el texto en formato JSON con los valores que desea adicionar "name": "Computers", "icon": "computer.png".

y haga click en el botón [Send]. Abajo en la ficha [Body] podrá ver la respuesta del servidor en formato JSON. El 1 al final de la url corresponde al id del registro que se desea modificar.

The screenshot shows the Postman application interface. In the top left, it says 'Activities Postman'. The top bar includes 'File Edit View Help', 'New', 'Import', 'Runner', 'My Workspace', 'Invite', and 'Upgrade'. Below the bar, there are several tabs: 'POST http://localhost:8080/ecommerce...', 'GET http://localhost:8080/ecommerce...', 'PUT http://localhost:8080/ecommerce...', and 'http://localhost:8080/ecommerce/ws/category/1'. The 'PUT' tab is selected. The URL 'http://localhost:8080/ecommerce/ws/category/1' is entered in the 'Request URL' field. The 'Method' dropdown is set to 'PUT'. The 'Body' tab is selected, and the 'raw' radio button is checked. The raw JSON payload is: `{"name": "Computers", "icon": "computer.png"}`. To the right, there is a 'Send' button with a blue outline. Below the request area, the status bar shows 'Status: 200 OK', 'Time: 29 ms', and 'Size: 158 B'. The bottom section shows the response body: `1 { 2 "mensaje": "Modificar OK" 3 }`.

Figura 2.23: Probar Servicio Web en Postman - Update

Verifique en la base de datos (ecommerce) que las modificaciones se están realizando en la tabla [categories]. Se puede ver que el registro con id: 1 fue modificado.

Edit Data - localhost (localhost:5432) - ecommerce - public.categories						
	<input type="checkbox"/>	<input type="checkbox"/>				
	<input type="checkbox"/>	<input type="checkbox"/>				
1	1	0	Computers	computer.png	2018-11-19 12:02:19.49203	2018-11-1
2	2	1	Category Two	categoryv02.jpg	2018-11-19 12:02:19.49203	2018-11-1
3	3	1	Category Three	categoryv03.jpg	2018-11-19 12:02:19.49203	2018-11-1

Figura 2.24: Ver cambios en la base de datos - Update

#### 2.7.4. Delete:

Para probar la operación Delete, seleccione el método DELETE, digite la url (`http://localhost:8080/ecommerce/ws/category/6`), y haga click en el botón [Send]. Abajo en la ficha [Body] podrá ver la respuesta del servidor en formato JSON. El 6 al final de la url corresponde al id del registro que se desea eliminar.

The screenshot shows the Postman interface with a DELETE request to `http://localhost:8080/ecommerce/ws/category/6`. The response body contains the JSON object `{"mensaje": "Eliminar OK"}`.

KEY	VALUE	DESCRIPTION
Key	Value	Description

Figura 2.25: Probar Servicio Web en Postman - Delete

Verifique en la base de datos (ecommerce) que en la tabla (categories) el registro con id: 6 ha sido eliminado.

	<b>id</b> [PK] bigserial	<b>published</b> integer	<b>name</b> character varying(255)	<b>icon</b> character varying(255)	crea...
1	1	0	Computers	computer.ipa	2018-01-01 00:00:00
2	2	1	Category Two	categov02.ipa	2018-01-01 00:00:00
3	3	1	Category Three	categov03.ipa	2018-01-01 00:00:00
4	4	1	Category Four	categov04.ipa	2018-01-01 00:00:00
5	5	1	Category Five	categov05.ipa	2018-01-01 00:00:00
*					

Figura 2.26: Ver cambios en la base de datos - Delete

# Capítulo 3

## Iteración 3: Servicios Web (Front-End)

### 3.1. Implementar el CRUD

Abra el eclipse y dentro del folder WebContent cree las siguientes carpetas:

- category
- portal
- util
- js



### 3.1.1. Archivo: category/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!doctype html>
<html lang="en">
<head>
<!-- Required meta tags --&gt;
&lt;meta charset="utf-8"&gt;
&lt;meta name="viewport"
  content="width=device-width, initial-scale=1, shrink-to-fit=no"&gt;
<!-- Bootstrap CSS --&gt;
&lt;link rel="stylesheet"
  href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"&gt;
&lt;link rel="stylesheet"
  href="https://cdn.datatables.net/1.10.19/css/dataTables.bootstrap4.min.css" /&gt;
&lt;link rel="stylesheet"
  href="https://cdn.datatables.net/select/1.3.0/css/dataTables.select.min.css" /&gt;
&lt;title&gt;eCommerce&lt;/title&gt;
&lt;/head&gt;
&lt;body&gt;
<!-- Navigation --&gt;
&lt;jsp:include page="../portal/menu.jsp" /&gt;
<!-- Page Content --&gt;
&lt;div class="container"&gt;
&lt;br/&gt;&lt;br/&gt;&lt;br/&gt;
&lt;h3&gt;Gestión de Categorías&lt;/h3&gt;
&lt;jsp:include page="../util/table_header.jsp" /&gt;
&lt;jsp:include page="category_table.jsp" /&gt;
&lt;jsp:include page="category_form.jsp" /&gt;
&lt;br/&gt;&lt;br/&gt;
&lt;/div&gt;
<!-- /.container --&gt;
<!-- Footer --&gt;
&lt;jsp:include page="../portal/footer.jsp" /&gt;
<!-- jQuery first, then Popper.js, then Bootstrap JS --&gt;

&lt;script src="https://code.jquery.com/jquery-3.3.1.js"&gt;&lt;/script&gt;
&lt;script
  src="https://cdn.datatables.net/1.10.19/js/dataTables.min.js"&gt;
&lt;/script&gt;
&lt;script
  src="https://cdn.datatables.net/1.10.19/js/dataTables.bootstrap4.min.js"&gt;
&lt;/script&gt;
&lt;script
  src="https://cdn.datatables.net/select/1.3.0/js/dataTables.select.min.js"&gt;
&lt;/script&gt;
&lt;script
  src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"&gt;
&lt;/script&gt;
&lt;script src="../js/category.js"&gt;&lt;/script&gt;
&lt;/body&gt;
&lt;/html&gt;</pre>
```

Listing 3.1.1: category/index.jsp

Este es el archivo inicial que verá el usuario cuando acceda desde un navegador al sitio ecommerce/-category. Aquí se vinculan las hojas de estilo (CSS) (`<link rel="stylesheet" .../>`), se asocian los archivos de JavaScript (JS) (`<script src="..."></script>`), se incluyen las Páginas de Servidor de Java (JSP) (`<jsp:include page="..."/>`), es decir componentes web y los elementos de HTML que conforman esta página inicial.

En esta aplicación web se utiliza los frameworks – librerías de Boostrap y DataTables. Se hace un llamado a los archivos que están en la web, asumiendo que la aplicación está desplegada en un servidor en la nube. Si el despliegue es en una intranet sería mejor descargar los archivos de las librerías e incluirlos dentro del proyecto.

### 3.1.2. Archivo: portal/menu.jsp

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
  <div class="container">
    <a class="navbar-brand" href="#">Start Bootstrap</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
      data-target="#navbarResponsive" aria-controls="navbarResponsive"
      aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="collapse navbar-collapse" id="navbarResponsive">
      <ul class="navbar-nav ml-auto">
        <li class="nav-item active"><a class="nav-link" href="..">Home
          <span class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item"><a class="nav-link" href="#">About</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Services</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Category</a></li>
        <li class="nav-item"><a class="nav-link" href="#">Contact</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Listing 3.1.2: portal/menu.jsp

Este componente web corresponde a la barra de navegación (nav), el menú horizontal que está arriba en el cabezote de la página.

### 3.1.3. Archivo: portal/footer.jsp

```
<footer class="py-5 bg-dark">
  <div class="container">
    <p class="m-0 text-center text-white">Copyright © Your
      Website 2019</p>
  </div>
  <!-- /.container -->
</footer>
```

Listing 3.1.3: portal/footer.jsp

Este componente web corresponde al pie de página (footer) que está ubicado en la parte de abajo de la página.

### 3.1.4. Archivo: util/table\_header.jsp

```
<div class="float-right">
  <button type="button" class="btn btn-primary" id="adicionar">Adicionar</button>
  <button type="button" class="btn btn-success" id="modificar">Modificar</button>
  <button type="button" class="btn btn-info" id="eliminar">Eliminar</button>
</div>
<br/>
<br/>
<div class="alert alert-success" id="success-alert">
  <strong id="success_title"></strong><span id="success_message"></span>
</div>
<div class="alert alert-danger" id="error-alert">
  <strong id="error_title"></strong><span id="error_message"></span>
</div>
```

Listing 3.1.4: util/table\_header.jsp

Este componente web corresponde al cabezote (header) de la tabla que incluye los botones del CRUD (Adicionar, Modificar, Eliminar) y los componentes de los mensajes de alerta para retroalimentar al usuario.

### 3.1.5. Archivo: category/category\_form.jsp

```
<!-- Modal -->
<div class="modal fade" id="categoryModal" tabindex="-1" role="dialog"
aria-labelledby="exampleModalLabel" aria-hidden="true">
<div class="modal-dialog" role="document">
<form id="categoryForm" method="POST" action="/ws/category/">
<input type="hidden" id="id" name="id" value="" />
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title" id="exampleModalLabel">Category Form</h5>
<button type="button" class="close" data-dismiss="modal"
aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
</div>
<div class="modal-body">
<div class="form-group">
<label for="name">Name</label> <input type="text"
class="form-control" id="name" name="name" placeholder="">
</div>
<div class="form-group">
<label for="icon">Icon</label> <input type="text"
class="form-control" id="icon" name="icon" placeholder="">
</div>
<div class="form-group">
<label for="published">Published</label>
<!-- selected -->
<select id="published" name="published" class="form-control">
<option value="1">Published</option>
<option value="0">Not published</option>
</select>
</div>
</div>
<!-- end modal-body -->
<div class="modal-footer">
<button type="button" class="btn btn-secondary"
data-dismiss="modal">Cerrar</button>
<button type="button" class="btn btn-primary" id="sendJSON">Save changes</button>
</div>
</div>
<!-- end modal-content -->
</form>
</div>
</div>
```

Listing 3.1.5: category/category\_form.jsp

Este componente web corresponde al formulario modal que permite al usuario editar la información que se desea actualizar (insertar, modificar, eliminar).

### 3.1.6. Archivo: category/category\_table.jsp

Este componente web corresponde a la tabla HTML para la gestión de los datos, de la tabla categories de la base de datos. Aquí se definen las filas (`<tr>`) del encabezado (header) y pie de página (footer) de la tabla HTML.

```
<table id="categoryTable" class="table table-striped table-bordered" style="width: 100%">
  <thead>
    <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Icon</th>
      <th>Published</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <th>Id</th>
      <th>Name</th>
      <th>Icon</th>
      <th>Published</th>
    </tr>
  </tfoot>
</table>
```

Listing 3.1.6: category/category\_table.jsp



### 3.1.7. Archivo: js/category.js (Parte 1)

```
$(document).ready(function() {
    // var selected_class = "active";
    var selected_class = "selected";
    $("#success-alert").hide();
    $("#error-alert").hide();

    var table = $('#categoryTable').DataTable({
        "ajax" : ".../ws/category",
        "columns" : [ {
            "data" : "id"
        }, {
            "data" : "name"
        }, {
            "data" : "icon"
        }, {
            "data" : "published"
        } ]
    });

    $('#categoryTable tbody').on('click', 'tr', function() {
        if ($(this).hasClass(selected_class)) {
            $(this).removeClass(selected_class);
        } else {
            table.$('tr.' + selected_class).removeClass(selected_class);
            $(this).addClass(selected_class);
        }
    });

    function ajaxCallRequest(f_method, f_url, f_data) {
        var f_contentType = 'application/json; charset=UTF-8';
        $.ajax([
            url : f_url,
            type : f_method,
            contentType : f_contentType,
            dataType : 'json',
            data : f_data,
            error: function(data) {
                var title = "Error !";
                var message = "Error al ejecutar la operación";
                $("#error_title").text(title);
                $("#error_message").text(message);
                $("#categoryModal .close").click();
                $("#error-alert").fadeTo(2000, 500).slideUp(500, function() {
                    $("#error-alert").slideUp(500);
                });
            },
            success: function(data) {
                var jsonResult = JSON.stringify(data);
                var parsed = JSON.parse(jsonResult);
                var success = parsed.success;
                var message = parsed.mensaje
                var title = "Success !";
                if(success == false){
                    title = "Error !";
                }

                $("#success_title").text(title);
                $("#success_message").text(message);
                $("#categoryModal .close").click();
                $("#success-alert").fadeTo(2000, 500).slideUp(500, function() {
                    $("#success-alert").slideUp(500);
                });
                table.ajax.reload();
            }
        });
    }
});
```

### 3.1.8. Archivo: js/category.js (Parte 2)

```
function ajaxCallRequest(f_method, f_url, f_data) {
    var f_contentType = 'application/json; charset=UTF-8';
    $.ajax({
        url : f_url,
        type : f_method,
        contentType : f_contentType,
        dataType : 'json',
        data : f_data,
        error: function(data) {
            var title = "Error !";
            var message = "Error al ejecutar la operación";
            $("#error_title").text(title);
            $("#error_message").text(message);
            $("#categoryModal .close").click();
            $("#error-alert").fadeTo(2000, 500).slideUp(500, function() {
                $("#error-alert").slideUp(500);
            });
        },
        success: function(data) {
            var jsonResult = JSON.stringify(data);
            var parsed = JSON.parse(jsonResult);
            var success = parsed.success;
            var message = parsed.mensaje
            var title = "Success !";
            if(success == false){
                title = "Error !";
            }

            $("#success_title").text(title);
            $("#success_message").text(message);
            $("#categoryModal .close").click();
            $("#success-alert").fadeTo(2000, 500).slideUp(500, function() {
                $("#success-alert").slideUp(500);
            });
            table.ajax.reload();
        }
    });
}

$("#sendJSON").click(function(event) {
    event.preventDefault();
    var form = $('#categoryForm');
    var method = form.attr('method');
    var url = form.attr('action');
    if (method != "POST") {
        var id = document.getElementById("id").value;
        url = url + id;
    }
    var jsonData = {};
    $.each($(form).serializeArray(), function() {
        jsonData[this.name] = this.value;
    });
    var data = JSON.stringify(jsonData);
    console.log(data);
    ajaxCallRequest(method, url, data);
});
```

Listing 3.1.8: js/category.js (Parte 2)

### 3.1.9. Archivo: js/category.js (Parte 3)

```

function editar(method){
    var id = 0;
    var name = "";
    var icon = "";
    var published = -1;
    if(method != "POST"){
        var data = table.rows('.' + selected_class).data()[0];
        if (data == undefined) {
            var title = "Error!";
            var operacion = "";
            if(method == "PUT"){
                operacion = "modificar.";
            }
            else{
                operacion = "eliminar.";
            }
            var message = "Por favor seleccione el registro que desea ";
            message += operacion;
            $("#error_title").text(title);
            $("#error_message").text(message);
            $("#error-alert").fadeTo(2000, 500).slideUp(500, function() {
                $("#error-alert").slideUp(500);
            });
            return;
        }
        else{
            id = data.id;
            name = data.name;
            icon = data.icon;
            published = data.published;
        }
    }
    if(method == "POST"){
        $("#sendJSON").html('Adicionar');
    }
    if(method == "PUT"){
        $("#sendJSON").html('Modificar');
    }
    if(method == "DELETE"){
        $("#sendJSON").html('Eliminar');
    }
    $("#categoryForm").attr("method", method);
    document.getElementById("id").value = id;
    document.getElementById("name").value = name;
    document.getElementById("icon").value = icon;
    document.getElementById("published").value = published;
    $('#categoryModal').modal('show');
}
$("#adicionar").click(function(event) {
    editar("POST");
});
$("#modificar").click(function(event) {
    editar("PUT");
});
$("#eliminar").click(function() {
    editar("DELETE");
});
}

```

Listing 3.1.9: js/category.js (Parte 3)

Este script implementa las funciones encargadas de hacer el llamado a los servicios web.

En este script (JavaScript) se crea la instancia (objeto) table de la clase DataTable [var table = \$('#categoryTable').DataTable(...)], se programa el evento click sobre el elemento tbody de la tabla

el cual permite seleccionar las filas [\$( '#categoryTable tbody' ).on('click'...)], se implementa el llamar do AJAX (Asynchronous JavaScript And XML) a los servicios web y la visualización de la respuesta [function ajaxCallRequest(f\_method, f\_url, f\_data)...], se programa el evento click sobre el botón del formulario modal (#sendJSON) para capturar los datos del formulario, convertirlos al formato JSON y enviarlos via AJAX [\$( "#sendJSON" ).click(function(event) ...)], se creó un método editar para gestio nar la captura de los datos de la fila seleccionada y mostrarlos en el formulario, validar que el usuario seleccione el registro (fila) que desea actualizar (solo para modificar y eliminar) y mostrar el formu lario modal [function editar(method)...]; y finalmente se implementó la programación para capturar el evento click de los botones Adicionar [\$( "#adicionar" ).click(function(event) ...], Modificar [\$( "#modi ficar" ).click(function(event) ...] y Eliminar [\$( '#eliminar' ).click(function() ...].

## 3.2. Correr el proyecto

Para correr el proyecto, haga click con el botón derecho del mouse sobre el proyecto (ecommerce), menú [Run As] y seleccione la opción [Run on Server].

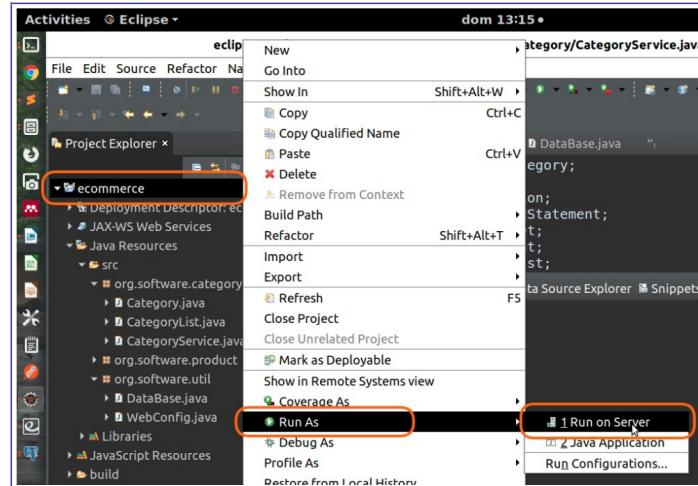


Figura 3.1: Correr el proyecto)

En la ficha [Console] se muestra el contexto web registrado en el servidor para acceder a la aplicación (/ecommerce) y se indica que la aplicación ha sido desplegada.

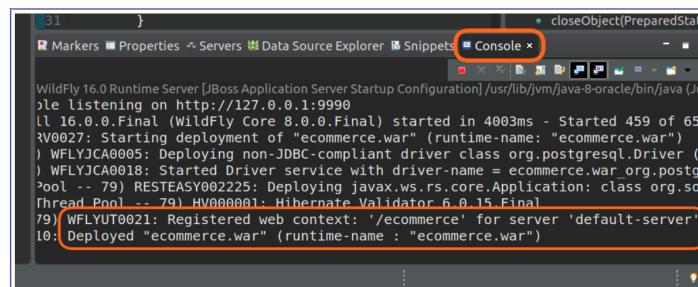


Figura 3.2: Despliegue de la aplicación web)

## 3.3. Probar los servicios web

Para probar los servicios web (CRUD), en un navegador abra la url: (<http://localhost:8080/ecommerce/ws/category>). Podrá ver la página que permite gestionar los datos de la tabla categories.

The screenshot shows a web browser window titled "eCommerce" with the URL "localhost:8080/ecommerce/category/". The page displays a table of categories with columns: Id, Name, Icon, and Published. The table contains six rows of data. At the top right, there are buttons for "Adicionar" (blue), "Modificar" (green), and "Eliminar" (teal). Below the table, there is a search bar and a navigation bar with links: Home, About, Services, Category, Contact. The footer of the page contains the text "Copyright © Your Website 2019".

Id	Name	Icon	Published
55	Categoría 55	categoria_55.jpg	1
56	Categoría 56	categoria_56.jpg	1
57	Categoría 57	categoria_57.jpg	1
58	Categoría 58	categoria_58.jpg	1
Id	Name	Icon	Published

Show 10 entries Search:

Showing 31 to 34 of 34 entries

Previous 1 2 3 4 Next

Copyright © Your Website 2019

Figura 3.3: Probar los servicios web)



# Capítulo 4

## Iteración 4: Componentes AJAX

### 4.1. Identificar los componentes

Para esta aplicación se va a utilizar una plantilla de bootstrap llamada shop-homepage, la cual se encuentra en la siguiente url:

[<https://startbootstrap.com/templates/shop-homepage/>].

Para descargar esta plantilla haga click al lado derecho en el botón [Free Download].

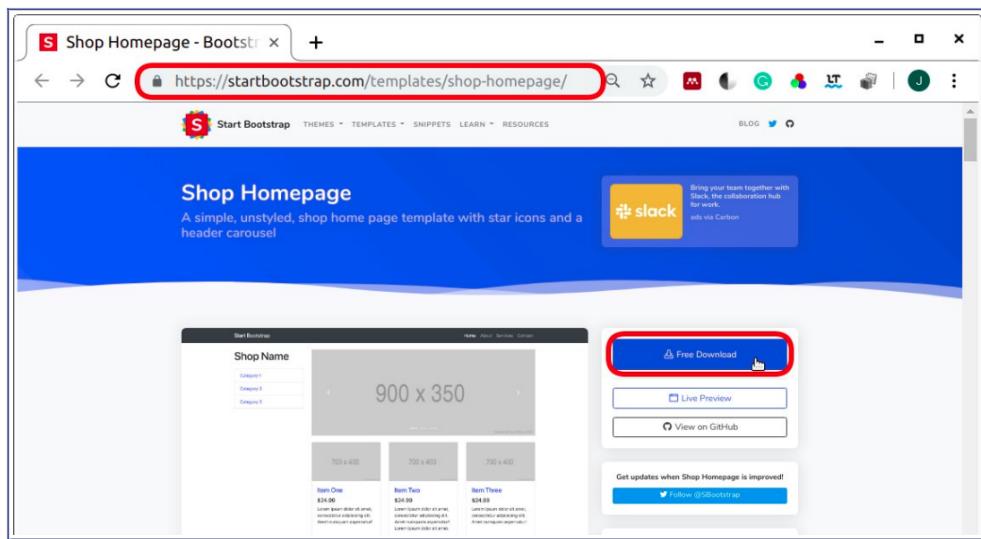


Figura 4.1: Plug-in DataTables

Al analizar la página [shop-homepage] se pueden identificar los componentes web que la definen

### 4.2. Crear los Servicios Web

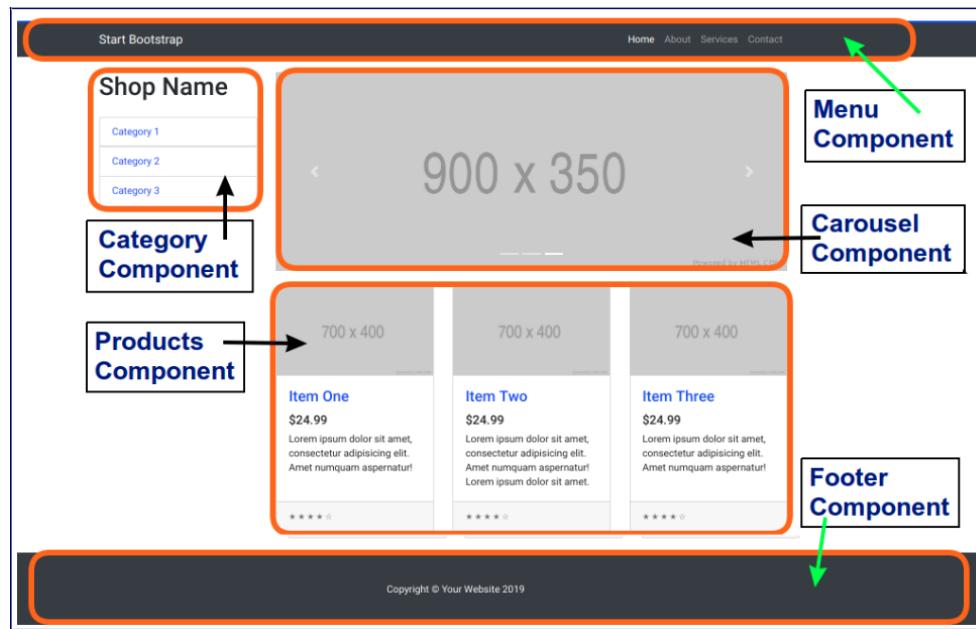


Figura 4.2: Plug-in DataTables

#### 4.2.1. Clase org.software.portal/PortalCategoryServervice.java

```
package org.software.portal;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import org.software.category.Category;
import org.software.category.CategoryList;
import org.software.util.DataBase;

@Path("/portal")
public class PortalCategoryService {
    @GET
    @Path("/categories")
    @Produces("application/json")
    public CategoryList read() {
        ArrayList<Category> categoryList = new ArrayList<Category>();
        DataBase database = new DataBase();
        Connection connection1 = null;
        Statement statement1 = null;
        ResultSet rs1 = null;
        String sql = "";
        try {
            connection1 = database.getConnection("guest");
            statement1 = connection1.createStatement();
            sql = "select * from categories where published = 1";
            sql += " order by name";
            rs1 = statement1.executeQuery(sql);
            while (rs1.next()) {
                int id = rs1.getInt("id");
                int published = rs1.getInt("published");
                String name = rs1.getString("name");
                String icon = rs1.getString("icon");
                Category category = new Category();
                category.setId(id);
                category.setPublished(published);
                category.setName(name);
                category.setIcon(icon);
                categoryList.add(category);
            }
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        } finally {
            database.closeObject(rs1);
            database.closeObject(statement1);
            database.closeObject(connection1);
        }
        return new CategoryList(categoryList);
    }
}
```

Listing 4.2.1: Clase: org.software.portal/PortalCategoryServervice.java



# Capítulo 5

## Iteración 5: Seguridad con JAAS

### 5.1. Introducción

En este proyecto se va a utilizar el Servicio de Autorización y Autenticación de Java - (JAAS - Java Authentication and Authorization Service), pronunciado como "Jazz", para implementar una Seguridad Basada en Roles. JAAS es una Interfaz de Programación de Aplicaciones (API) que permite a las aplicaciones Java acceder a servicios de control de autenticación y acceso.

=

---

**Note:** Se recomienda importar el proyecto en eclipse y no copiar el código de los archivos PDF.

---

### 5.2. Encriptar las claves

Para encriptar las claves de los usuarios se debe activar la librería pgcrypto en el motor de base de datos PostgreSQL. Para ello seleccione la base de datos [ecommerce] y ejecute el siguiente comando en la ventana de SQL [Query Tool].

```
CREATE EXTENSION pgcrypto;
```

La función encode() de la librería pgcrypto permite encriptar una clave con el algoritmo md5 y codificar el resultado en base64:

```
SELECT encode(digest('clave_a_encriptar', 'md5'), 'base64');
```

Para actualizar las claves de los usuarios, ejecute las siguientes consultas SQL:

```
UPDATE users SET password = encode(digest('pgutierrez2018', 'md5'), 'base64') WHERE username =
→ 'pedro';
UPDATE users SET password = encode(digest('mquintero2018', 'md5'), 'base64') WHERE username =
→ 'maria';
UPDATE users SET password = encode(digest('jlopez2018', 'md5'), 'base64') WHERE username = 'jose';
```

Para ver las claves encriptadas, ejecute una sentencia SELECT.

```
SELECT * from users;
```

La Figura 5.2 podemos ver que la columna a la derecha presenta el campo password con las claves encriptadas.

2	1	0	iose	iose.lopez@gmail.com	o0i2zmb0n1h/a4ev0Z10fw==
3	2	0	maria	maria.quintero@gmail.com	ICD0TdR2VeZNVGhxIKxaFw==
4	3	0	pedro	pedro.autierrez@gmail.com	ArYun2FnNK48rMrlzd1LX0==

Figura 5.1: Datos de los usuarios

## 5.3. Crear las Tablas

El Diagrama Entidad Relación muestra las tablas necesarias para implementar el modelo de seguridad basada en roles. El modelo permite que un usuario tenga uno o más perfiles.

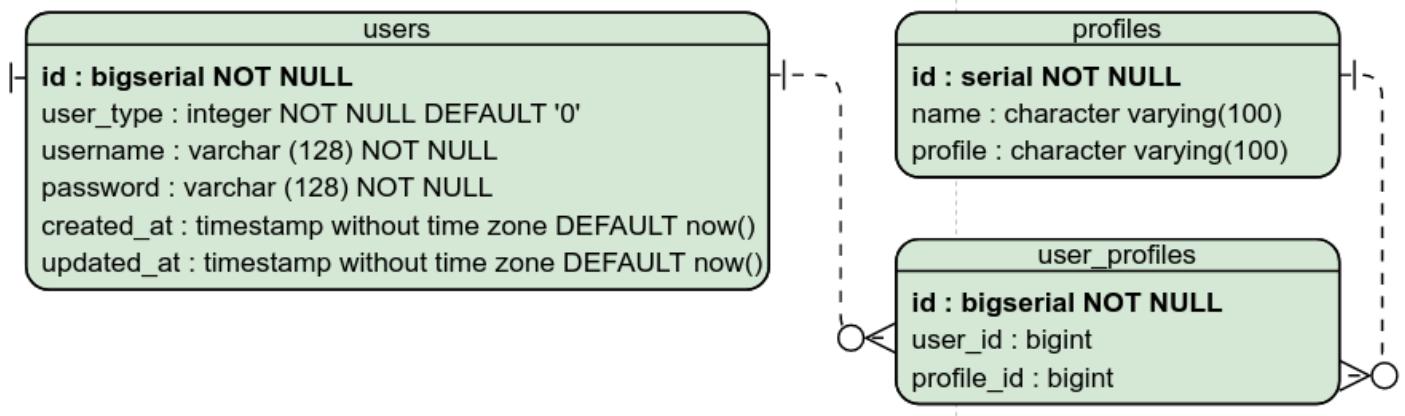


Figura 5.2: Diagrama Entidad Relación - DER

La tabla users ya fue creada, para crear las tablas (profiles y user\_profiles), ejecute las siguientes sentencias en el editor de SQL:

### 5.3.1. Tabla: profiles

La tabla profiles permite almacenar los perfiles asociados al subsistema de seguridad.

```

CREATE TABLE public.profiles
(
    id serial NOT NULL,
    name character varying(100),
    profile character varying(100),
    CONSTRAINT group_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.profiles
OWNER TO ecommerce;
GRANT ALL ON TABLE public.profiles TO ecommerce;
  
```

Listing 5.3.1: Tabla: profiles

### 5.3.2. Tabla: user\_profiles

La tabla user\_profiles permite asignar uno o más perfiles a un usuario.

```
CREATE TABLE public.user_profiles
(
    id bigserial NOT NULL,
    user_id bigint,
    profile_id bigint,
    CONSTRAINT user_profiles_pkey PRIMARY KEY (id),
    CONSTRAINT user_profiles_profile_fkey FOREIGN KEY (profile_id)
        REFERENCES public.profiles (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT user_profiles_user_fkey FOREIGN KEY (user_id)
        REFERENCES public.users (id) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.user_profiles OWNER TO ecommerce;
GRANT ALL ON TABLE public.user_profiles TO ecommerce;
```

Listing 5.3.2: Tabla: user\_profiles

## 5.4. Alimentar la base de datos

Las siguientes sentencias SQL permiten adicionar los perfiles del subsistema de seguridad.

```
INSERT INTO profiles (name, profile) VALUES ('Administrador', 'ADMINISTRATOR');
INSERT INTO profiles (name, profile) VALUES ('Cliente', 'CLIENT');
```

Listing 5.4.1: Adicionar perfiles

Las siguientes sentencias SQL insertan en la base de datos los perfiles asociados a los usuarios.

```
INSERT INTO user_profiles (user_id, profile_id) VALUES
(1,1),
(2,2),
(3,1),
(3,2);
```

Listing 5.4.2: Adicionar perfiles de usuario

## 5.5. Asignar Permisos

Para asignar los permisos de acceso a las tablas, ejecute el comando GRANT. Al usuario de la base de datos ecommerce\_admin se le asignan los permisos de seleccionar, modificar, insertar y eliminar en las tablas de usuarios, perfiles y perfiles de usuario (users, profiles y user\_profiles).

```
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE public.users TO ecommerce_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE public.profiles TO ecommerce_admin;
GRANT SELECT, UPDATE, INSERT, DELETE ON TABLE public.user_profiles TO ecommerce_admin;
GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO ecommerce_admin;
```

Listing 5.5.1: Asignar permisos

## 5.6. Archivo de configuración del proyecto (POM)

Un Modelo de Objetos de Proyecto (Project Object Model) o POM es la unidad de trabajo fundamental en Maven. Es un archivo XML que contiene información acerca del proyecto y detalles de configuración usados por Maven para construir el proyecto.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.software</groupId>
  <artifactId>ecommerce5</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>org-software-jaas</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>
```

Listing 5.6.1: Archivo de configuración del proyecto: pom.xml (Part 1: General properties)

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.8.1</version>
      <configuration>
        <release>17</release>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.2.3</version>
    </plugin>
  </plugins>
</build>
```

Listing 5.6.2: Archivo de configuración del proyecto: pom.xml (Part 2: build properties)

### 5.6.1. Dependencias

En la sección del dependencias del archivo POM se especifican los drivers (librerías) que se requieren para construir el proyecto.

```
<dependencies>
    <dependency>
        <groupId>org.glassfish.jersey.containers</groupId>
        <artifactId>jersey-container-servlet</artifactId>
        <version>2.29.1</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.inject</groupId>
        <artifactId>jersey-hk2</artifactId>
        <version>2.29.1</version>
    </dependency>
    <dependency>
        <groupId>org.glassfish.jersey.media</groupId>
        <artifactId>jersey-media-json-jackson</artifactId>
        <version>2.29.1</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.postgresql/postgresql -->
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.5</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.glassfish/javax.json -->
    <dependency>
        <groupId>org.glassfish</groupId>
        <artifactId>javax.json</artifactId>
        <version>1.1.4</version>
    </dependency>
</dependencies>
```

Listing 5.6.3: Archivo de configuración del proyecto: pom.xml (Part 3: dependencies)

## 5.7. Configurar el Servicio de Autenticación y Autorización (JAAS)

### 5.7.1. Clase: org.software.jaas.UserPrincipal.java

```
1 package org.software.jaas;
2
3 import java.security.Principal;
4
5 public class UserPrincipal implements Principal {
6
7     private String name;
8
9     public UserPrincipal(String name) {
10         super();
11         this.name = name;
12     }
13
14     public void setName(String name) {
15         this.name = name;
16     }
17
18     @Override
19     public String getName() {
20         return name;
21     }
22 }
23 }
```

Listing 5.7.1: Clase: org.software.jaas.UserPrincipal.java

### 5.7.2. Clase: org.software.jaas.RolePrincipal.java

```
1 package org.software.jaas;
2
3 import java.security.Principal;
4
5 public class RolePrincipal implements Principal {
6
7     private String name;
8
9     public RolePrincipal(String name) {
10         super();
11         this.name = name;
12     }
13
14     public void setName(String name) {
15         this.name = name;
16     }
17
18     @Override
19     public String getName() {
20         return name;
21     }
22 }
23 }
```

Listing 5.7.2: Clase: org.software.jaas.RolePrincipal.java

### 5.7.3. Clase: org.software.jaas.SoftwareLoginModule.java

```
1 package org.software.jaas;
2
3 import java.io.IOException;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6 import java.sql.Connection;
7 import java.sql.PreparedStatement;
8 import java.sql.ResultSet;
9 import java.sql.Statement;
10 import java.util.ArrayList;
11 import java.util.Base64;
12 import java.util.List;
13 import java.util.Map;
14
15 import javax.security.auth.Subject;
16 import javax.security.auth.callback.Callback;
17 import javax.security.auth.callback.CallbackHandler;
18 import javax.security.auth.callback.NameCallback;
19 import javax.security.auth.callback.PasswordCallback;
20 import javax.security.auth.callback.UnsupportedCallbackException;
21 import javax.security.auth.login.LoginException;
22 import javax.security.auth.spi.LoginModule;
23
24 import org.software.category.Category;
25 import org.software.util.DBUtil;
26 import org.software.util.DataBase;
27
28 public class SoftwareLoginModule implements LoginModule {
29
30     private CallbackHandler handler;
31     private Subject subject;
32     private UserPrincipal userPrincipal;
33     private RolePrincipal rolePrincipal;
34     private String login;
35     private List<String> userGroups;
```

Listing 5.7.3: Clase: org.software.jaas.SoftwareLoginModule.java (Part 1)

```
37 @Override
38 public void initialize(Subject subject, CallbackHandler callbackHandler,
39                         Map<String, ?> sharedState, Map<String, ?> options) {
40
41     handler = callbackHandler;
42     this.subject = subject;
43 }
```

Listing 5.7.4: Clase: org.software.jaas.SoftwareLoginModule.java (Part 2)

```
46 public String getEncryptedPassword(String plainPassword) {  
47     String encryptedPassword = "";  
48  
49     MessageDigest md;  
50     try {  
51         byte[] bytesOfMessage = plainPassword.getBytes("UTF-8");  
52  
53         md = MessageDigest.getInstance("MD5");  
54  
55         encryptedPassword = Base64.getEncoder().encodeToString(md.digest(bytesOfMessage));  
56     } catch (Exception e) {  
57         System.out.println("Error: " + e.toString());  
58     }  
59  
60     return encryptedPassword;  
61 }  
62 }
```

Listing 5.7.5: Clase: org.software.jaas.SoftwareLoginModule.java (Part 3)

```
64 public List<String> getProfiles(String username) {  
65     List<String> userGroups = new ArrayList<String>();  
66  
67     DataBase database = new DataBase();  
68     Connection connection1 = null;  
69     PreparedStatement preparedStatement1 = null;  
70     ResultSet rs1 = null;  
71  
72     String sql = "";  
73  
74     try {  
75         connection1 = database.getConnection("admin");  
76  
77         sql = "select profiles.profile from users, user_profiles, profiles";  
78         sql += " where users.id = user_profiles.user_id";  
79         sql += " and user_profiles.profile_id = profiles.id";  
80         sql += " and username = ?";  
81  
82         preparedStatement1 = connection1.prepareStatement(sql);  
83         preparedStatement1.setString(1, username);  
84         rs1 = preparedStatement1.executeQuery();  
85  
86         while (rs1.next()) {  
87             String profile = rs1.getString("profile");  
88             userGroups.add(profile);  
89         }  
90     } catch (Exception e) {  
91         System.out.println("Error: " + e.toString());  
92     } finally {  
93         database.closeObject(rs1);  
94         database.closeObject(preparedStatement1);  
95         database.closeObject(connection1);  
96     }  
97  
98     return userGroups;  
99 }
```

Listing 5.7.6: Clase: org.software.jaas.SoftwareLoginModule.java (Part 4)

```
103 public String getDBPassword(String username) {  
104     String dbPassword = "";  
105  
106     DataBase database = new DataBase();  
107     Connection connection1 = null;  
108     PreparedStatement preparedStatement1 = null;  
109     ResultSet rs1 = null;  
110     String sql = "";  
111     try {  
112         connection1 = database.getConnection("admin");  
113  
114         sql = "SELECT password from users";  
115         sql += " where username = ?";  
116  
117         preparedStatement1 = connection1.prepareStatement(sql);  
118         preparedStatement1.setString(1, username);  
119         rs1 = preparedStatement1.executeQuery();  
120  
121         while (rs1.next()) {  
122             dbPassword = rs1.getString("password");  
123         }  
124     } catch (Exception e) {  
125         System.out.println("Error: " + e.toString());  
126     }  
127     finally {  
128         database.closeObject(rs1);  
129         database.closeObject(preparedStatement1);  
130         database.closeObject(connection1);  
131     }  
132  
133     return dbPassword;  
134 }  
135
```

Listing 5.7.7: Clase: org.software.jaas.SoftwareLoginModule.java (Part 5)

```

138
139     @Override
140     public boolean login() throws LoginException {
141
142         Callback[] callbacks = new Callback[2];
143         callbacks[0] = new NameCallback("login");
144         callbacks[1] = new PasswordCallback("password", true);
145
146         try {
147             handler.handle(callbacks);
148             String name = ((NameCallback) callbacks[0]).getName();
149             String password = String.valueOf(((PasswordCallback) callbacks[1]))
150                 .getPassword());
151
152             // Here we validate the credentials against a
153             // Database authentication/authorization provider.
154
155             String encryptedPassword = getEncryptedPassword(password);
156             String dbPassword = getDBPassword(name);
157
158             if (name != null && password != null
159                 && encryptedPassword.equals(dbPassword)) {
160                 login = name;
161
162                 userGroups = getProfiles(name);
163
164                 return true;
165             }
166
167             // If credentials are NOT OK we throw a LoginException
168             throw new LoginException("Authentication failed");
169
170         } catch (IOException e) {
171             throw new LoginException(e.getMessage());
172         } catch (UnsupportedCallbackException e) {
173             throw new LoginException(e.getMessage());
174         }
175     }

```

Listing 5.7.8: Clase: org.software.jaas.SoftwareLoginModule.java (Part 6)

```

177
178     @Override
179     public boolean commit() throws LoginException {
180
181         userPrincipal = new UserPrincipal(login);
182         subject.getPrincipals().add(userPrincipal);
183
184         if (userGroups != null && userGroups.size() > 0) {
185             for (String groupName : userGroups) {
186                 rolePrincipal = new RolePrincipal(groupName);
187                 subject.getPrincipals().add(rolePrincipal);
188             }
189
190         }
191
192         return true;
193     }

```

Listing 5.7.9: Clase: org.software.jaas.SoftwareLoginModule.java (Part 7)

```
193 @Override  
194 public boolean abort() throws LoginException {  
195     return false;  
196 }
```

Listing 5.7.10: Clase: org.software.jaas.SoftwareLoginModule.java (Part 8)

```
198 @Override  
199 public boolean logout() throws LoginException {  
200     subject.getPrincipals().remove(userPrincipal);  
201     subject.getPrincipals().remove(rolePrincipal);  
202     return true;  
203 }
```

Listing 5.7.11: Clase: org.software.jaas.SoftwareLoginModule.java (Part 9)

### 5.7.4. Clase: org.software.jaas.Logout.java

```
1 package org.software.jaas;
2
3 import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9 import javax.servlet.http.HttpSession;
10
11 /**
12 * Servlet implementation class Logout
13 */
14 @WebServlet("/Logout")
15 public class Logout extends HttpServlet {
16     private static final long serialVersionUID = 1L;
17
18     public Logout() {
19         super();
20     // TODO Auto-generated constructor stub
21     }
22
23     /**
24      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
25      *      response)
26      */
27     protected void doGet(HttpServletRequest request, HttpServletResponse response)
28         throws ServletException, IOException {
29         HttpSession session = request.getSession();
30         session.invalidate();
31         response.sendRedirect("./");
32     }
33 }
```

Listing 5.7.12: Clase: org.software.jaas.Logout.java

## 5.8. Acceso a la Base de Datos

### 5.8.1. Clase: org.software.util.DataBase.java

```
1 package org.software.util;  
2  
3 import java.sql.Connection;  
4 import java.sql.PreparedStatement;  
5 import java.sql.ResultSet;  
6 import java.sql.SQLException;  
7 import java.sql.Statement;  
8  
9 import javax.naming.Context;  
10 import javax.naming.InitialContext;  
11 import javax.sql.DataSource;
```

Listing 5.8.1: Clase: org.software.util.DataBase.java (Part 1)

```
14 public Connection getConnection(String profile) {  
15     Connection connection = null;  
16  
17     String JndiDataSourceName = "";  
18  
19     if (profile.equals("admin")) {  
20         JndiDataSourceName = "eCommerceAdminDS";  
21     }  
22     if (profile.equals("client")) {  
23         JndiDataSourceName = "eCommerceClientDS";  
24     }  
25     if (profile.equals("guest")) {  
26         JndiDataSourceName = "eCommerceGuestDS";  
27     }  
28  
29     System.out.println("JndiDataSourceName: " + JndiDataSourceName);  
30  
31     try {  
32         Context ctx = new InitialContext();  
33         DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/" + JndiDataSourceName);  
34  
35         connection = ds.getConnection();  
36     } catch (Exception e) {  
37         System.out.println("Error: " + e.toString());  
38     }  
39     return connection;  
40 }  
41 }
```

Listing 5.8.2: Clase: org.software.util.DataBase.java (Part 2)

```
43     public void closeObject(Connection connection) {
44         try {
45             connection.close();
46         } catch (SQLException e) {
47             e.printStackTrace();
48         }
49     }
```

Listing 5.8.3: Clase: org.software.util.DataBase.java (Part 3)

```
51     public void closeObject(PreparedStatement preparedStatement) {
52         try {
53             preparedStatement.close();
54         } catch (SQLException e) {
55             e.printStackTrace();
56         }
57     }
```

Listing 5.8.4: Clase: org.software.util.DataBase.java (Part 4)

```
59     public void closeObject(Statement statement) {
60         try {
61             statement.close();
62         } catch (SQLException e) {
63             e.printStackTrace();
64         }
65     }
```

Listing 5.8.5: Clase: org.software.util.DataBase.java (Part 5)

```
67     public void closeObject(ResultSet resultSet) {
68         try {
69             resultSet.close();
70         } catch (SQLException e) {
71             e.printStackTrace();
72         }
73     }
```

Listing 5.8.6: Clase: org.software.util.DataBase.java (Part 6)

### 5.8.2. Clase: org.software.util.DBUtil.java

```
1 package org.software.util;
2
3 import java.sql.Connection;
4 import java.sql.PreparedStatement;
5 import java.sql.ResultSet;
6 //import java.sql.Statement;
7 //import java.sql.Timestamp;
8 //import java.text.SimpleDateFormat;
9
10 public class DBUtil {
11
12     public String getValue(String sql, String id) {
13
14         String value = "";
15
16         DataBase database = new DataBase();
17         Connection connection1 = null;
18         PreparedStatement preparedStatement1 = null;
19         ResultSet rs1 = null;
20
21         try {
22             connection1 = database.getConnection("admin");
23
24             preparedStatement1 = connection1.prepareStatement(sql);
25             preparedStatement1.setString(1, id);
26             rs1 = preparedStatement1.executeQuery();
27
28             while (rs1.next()) {
29                 value = rs1.getString(1);
30             }
31         } catch (Exception e) {
32             System.out.println("Error: " + e.toString());
33         }
34         finally {
35             database.closeObject(rs1);
36             database.closeObject(preparedStatement1);
37             database.closeObject(connection1);
38         }
39
40         return value;
41     }
42 }
43 }
```

Listing 5.8.7: Clase: org.software.util.DBUtil.java

### 5.8.3. Archivo de Configuración: META-INF/context.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Context>
3   <Realm className="org.apache.catalina.realm.JAASRealm"
4     appName="SoftwareLogin"
5     userClassNames="org.software.jaas.UserPrincipal"
6     roleClassNames="org.software.jaas.RolePrincipal" />
7   <Resource name="jdbc/eCommerceAdminDS" auth="Container"
8     type="javax.sql.DataSource" maxTotal="50" maxIdle="30"
9     maxWaitMillis="10000" username="ecommerce_admin" password="234567"
10    driverClassName="org.postgresql.Driver"
11    url="jdbc:postgresql://localhost:5432/ecommerce" />
12   <Resource name="jdbc/eCommerceClientDS" auth="Container"
13     type="javax.sql.DataSource" maxTotal="50" maxIdle="30"
14     maxWaitMillis="10000" username="ecommerce_client" password="345678"
15     driverClassName="org.postgresql.Driver"
16     url="jdbc:postgresql://localhost:5432/ecommerce" />
17   <Resource name="jdbc/eCommerceGuestDS" auth="Container"
18     type="javax.sql.DataSource" maxTotal="50" maxIdle="30"
19     maxWaitMillis="10000" username="ecommerce_guest" password="456789"
20     driverClassName="org.postgresql.Driver"
21     url="jdbc:postgresql://localhost:5432/ecommerce" />
22 </Context>
```

Listing 5.8.8: Archivo de Configuración: META-INF/context.xml

#### 5.8.4. Archivo de Configuración: TOMCAT-HOME/conf/jaas.config

Se debe definir el archivo de configuración de JAAS. Lo puede nombrar jaas.config y ubicarlo en la carpeta conf (configuracion) de tomcat:

TOMCAT-HOME/conf/jaas.config

El archivo tiene el siguiente contenido:

```
1 SoftwareLogin {  
2     org.software.jaas.SoftwareLoginModule required debug=true;  
3 };
```

Listing 5.8.9: Archivo de Configuración: TOMCAT-HOME/conf/jaas.config

Este archivo define la configuración de autenticación para la aplicación SoftwareLogin. Tenga en cuenta que es el mismo nombre que se usó en appName dentro del archivo context.xml justo arriba.

### 5.8.5. Configurar el archivo WEB-INF/web.xml de la aplicación.

Es necesario configurar el archivo web.xml de la aplicación para utilizar el modo de autenticación apropiado.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  id="WebApp_ID" version="4.0">
  <display-name>ecommerce5</display-name>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
```

Listing 5.8.10: Archivo: WEB-INF/web.xml (Part 1)

La sección 5.8.11 define la configuración de los Servicios Web.

```
<servlet>
  <servlet-name>Ecommerce REST Service</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <init-param>
    <param-name>jersey.config.server.provider.packages</param-name>
    <param-value>org.software</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Ecommerce REST Service</servlet-name>
  <url-pattern>/ws/*</url-pattern>
</servlet-mapping>
```

Listing 5.8.11: Archivo: WEB-INF/web.xml (Part 2)

El nodo 5.8.12 crea una restricción de seguridad para la carpeta user y todos los archivos que estén por dentro y le autoriza el acceso únicamente a los usuarios que tengan el perfil ADMINISTRATORS o CLIENT.

La sección 5.8.15 define que el método de autenticación utilizará un formulario de login.

El listado 5.8.16 establece que la aplicación tendrá dos roles de seguridad (ADMINISTRATOR y CLIENT).

La sección 5.8.17 asigna la página personalizada para el tipo de error 403 (Acceso prohibido).

```
<security-constraint>
    <display-name>Restricción de Seguridad - Usuarios</display-name>
    <web-resource-collection>
        <web-resource-name>Area de User</web-resource-name>
        <url-pattern>/user/*</url-pattern>
        <http-method>GET</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ADMINISTRATOR</role-name>
        <role-name>CLIENT</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 5.8.12: Archivo: WEB-INF/web.xml (Part 3)

```
<security-constraint>
    <display-name>Security Constraint - Category</display-name>
    <web-resource-collection>
        <web-resource-name>Category Management</web-resource-name>
        <url-pattern>/ws/category/*</url-pattern>
        <url-pattern>/category/*</url-pattern>
        <http-method>POST</http-method>
        <http-method>GET</http-method>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>ADMINISTRATOR</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 5.8.13: Archivo: WEB-INF/web.xml (Part 4)

```
<security-constraint>
    <display-name>Security Constraint - Order</display-name>
    <web-resource-collection>
        <web-resource-name>Client Orders</web-resource-name>
        <url-pattern>/order/*</url-pattern>
        <http-method>POST</http-method>
        <http-method>GET</http-method>
        <http-method>PUT</http-method>
        <http-method>DELETE</http-method>
    </web-resource-collection>
    <auth-constraint>
        <role-name>CLIENT</role-name>
        <role-name>ADMINISTRATOR</role-name>
    </auth-constraint>
    <user-data-constraint>
        <transport-guarantee>NONE</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 5.8.14: Archivo: WEB-INF/web.xml (Part 5)

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>eCommerceSecurity</realm-name>
    <form-login-config>
        <form-login-page>/login/login.jsp</form-login-page>
        <form-error-page>/login/error.jsp</form-error-page>
    </form-login-config>
</login-config>
```

Listing 5.8.15: Archivo: WEB-INF/web.xml (Part 6)

```
<!-- roles-(perfiles o grupos) válidos para la aplicación web -->
<security-role>
    <role-name>ADMINISTRATOR</role-name>
</security-role>
<security-role>
    <role-name>CLIENT</role-name>
</security-role>
```

Listing 5.8.16: Archivo: WEB-INF/web.xml (Part 7)

```
<error-page>
    <error-code>403</error-code>
    <location>/forbidden.jsp</location>
</error-page>
```

Listing 5.8.17: Archivo: WEB-INF/web.xml (Part 8)

## 5.9. Aplicación Web

Cambios en el código fuente.

### 5.9.1. Componente Web: portal/menu.jsp

```
<%
    String username = "";
    try {
        username = request.getUserPrincipal().getName();
    } catch (Exception e) {
        username = "";
    }
%>
```

Listing 5.9.1: Componente: portal/menu.jsp (Part 1)

```
<!-- Navigation -->
<nav class="navbar navbar-expand-lg navbar-dark bg-dark fixed-top">
    <div class="container">
        <a class="navbar-brand" href="#">Start Bootstrap</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse"
            data-target="#navbarResponsive" aria-controls="navbarResponsive"
            aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ml-auto">
                <li class="nav-item active"><a class="nav-link" href="#">Home
                    <span class="sr-only">(current)</span>
                </a></li>
                <li class="nav-item"><a class="nav-link" href="#">Orders</a></li>
                <li class="nav-item"><a class="nav-link" href="#">Category</a></li>
                <li class="nav-item"><a class="nav-link" href="#">Cart</a>
                    <button type="button" class="btn btn-primary">
                        Shopping Cart <span id="shopping_cart" class="badge
                            badge-light">0</span>
                    </button>
                </a></li>
            </ul>
        </div>
    </div>
</nav>
```

Listing 5.9.2: Componente: portal/menu.jsp (Part 2)

```
<%
    if (username.length() == 0) {
%>
<li class="nav-item"><a class="nav-link" href="../user/">
    <button type="button" class="btn btn-success">Login</button>
</a></li>
<%
    }
%>
```

Listing 5.9.3: Componente: portal/menu.jsp (Part 3)

```
<%
    if (username.length() > 0) {
%>
<li class="nav-item">
    <div class="btn-group nav-link" role="group">
        <button id="btnGroupDrop1" type="button"
            class="btn btn-primary dropdown-toggle" data-toggle="dropdown"
            aria-haspopup="true" aria-expanded="false">
            <%=username%>
        </button>
        <div class="dropdown-menu" aria-labelledby="btnGroupDrop1">
            <a class="dropdown-item" href="../user/">Control Panel</a> <a
                class="dropdown-item" href="../purchase/">Purchases</a> <a
                class="dropdown-item" href="../Logout">Logout</a>
        </div>
    </div>
</li>
<%
    }
%>

    </ul>
</div>
</div>
</nav>
```

Listing 5.9.4: Componente: portal/menu.jsp (Part 4)

```
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.js"></script>

<script src="../js/portal.js" type="text/javascript"></script>
<script type="text/javascript">
    updateItemsCount();
</script>
```

Listing 5.9.5: Componente: portal/menu.jsp (Part 5)

### 5.9.2. Componente Web: login/login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <link href="../css/login.css" rel="stylesheet">
    <title>eCommerce</title>
</head>
<body>
    <!-- Navigation -->
    <jsp:include page="../portal/menu.jsp" />
    <!-- Page Content -->
    <div class="container login-container">
        <br /> <br /> <br />
        <div class="row">
            <div class="col-md-3"></div>
            <div class="col-md-6 login-form-1">
                <h3>Ingresar al Sistema</h3>
                <form action="j_security_check" method="post">
                    <div class="form-group">
                        <input type="text" class="form-control" name="j_username"
                            placeholder="Your Email *" value="" />
                    </div>
                    <div class="form-group">
                        <input type="password" class="form-control" name="j_password"
                            placeholder="Your Password *" value="" />
                    </div>
                    <div class="form-group">
                        <input type="submit" class="btnSubmit" value="Login" />
                    </div>
                    <div class="form-group">
                        <a href="#" class="ForgotPwd">Forgot Password?</a>
                    </div>
                </form>
            </div>
            <div class="col-md-3"></div>
        </div>
    </div>
    <!-- /.container -->
    <!-- Footer -->
    <jsp:include page="../portal/footer.jsp" />
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>
</html>
```

Listing 5.9.6: Componente: login/login.jsp

### 5.9.3. Componente Web: login/error.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <link href="../css/login.css" rel="stylesheet">
    <title>eCommerce</title>
</head>
<body>
    <!-- Navigation -->
    <jsp:include page="../portal/menu.jsp" />
    <!-- Page Content -->
    <div class="container">
        <br /> <br /> <br />
        <div class="row">
            <div class="col-md-12">
                <h2>Error al digitar el usuario o la clave.</h2>
                <a class="nav-link" href="user">
                    <button type="button" class="btn btn-danger">Volver a
                        intentar.</button>
                </a>
            </div>
        </div>
        <br /> <br /> <br />
    </div>
    <!-- /.container -->
    <!-- Footer -->
    <jsp:include page="../portal/footer.jsp" />
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>
</html>
```

Listing 5.9.7: Componente: login/error.jsp

#### 5.9.4. Componente Web: portal/forbidden.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
    <link href="../css/login.css" rel="stylesheet">
    <title>eCommerce</title>
</head>
<body>
    <!-- Navigation -->
    <jsp:include page="../portal/menu.jsp" />
    <!-- Page Content -->
    <div class="container">
        <br /> <br /> <br />
        <div class="row">
            <div class="col-md-12">
                <h2>Acceso no permitido, perfil no valido.</h2>
                <a class="nav-link" href="#">
                    <button type="button" class="btn btn-danger">Home</button>
                </a>
            </div>
        </div>
        <br /> <br /> <br />
    </div>
    <!-- /.container -->
    <!-- Footer -->
    <jsp:include page="../portal/footer.jsp" />
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>
</html>
```

Listing 5.9.8: Componente: portal/forbidden.jsp

### 5.9.5. Componente Web: user/index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport"
        content="width=device-width, initial-scale=1, shrink-to-fit=no">
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
        href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
<title>eCommerce</title>
</head>
<body>
    <!-- Navigation -->
    <jsp:include page="../portal/menu.jsp" />
    <!-- Page Content -->
    <div class="container">
        <br /> <br /> <br />
        <div class="list-group">
            <%
                String usuario = request.getUserPrincipal().getName();
                out.println("<h3>Bienvenido: " + usuario + "</h3>");
                if (request.isUserInRole("CLIENT")) {
            %>
                <a href="../order" class="list-group-item list-group-item-action">
                    Pedidos</a>
            <%
                }
                if (request.isUserInRole("ADMINISTRATOR")) {
            %>
                <a href="../category" class="list-group-item list-group-item-action">
                    Gestión de Categorías</a>
            <%
                }
            %>
        </div>
        <br /> <br />
    </div>
    <!-- /.container -->
    <!-- Footer -->
    <jsp:include page="../portal/footer.jsp" />
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.js"></script>
    <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
</body>
</html>
```

Listing 5.9.9: Componente: user/index.jsp

### 5.9.6. Hoja de Estilo: css/login.css

```
.login-container {
    margin-top: 5%;
    margin-bottom: 5%;
}
.login-form-1 {
    padding: 5%;
    box-shadow: 0 5px 8px 0 rgba(0, 0, 0, 0.2), 0 9px 26px 0
               rgba(0, 0, 0, 0.19);
}
.login-form-1 h3 {
    text-align: center;
    color: #333;
}
.login-form-2 {
    padding: 5%;
    background: #0062cc;
    box-shadow: 0 5px 8px 0 rgba(0, 0, 0, 0.2), 0 9px 26px 0
               rgba(0, 0, 0, 0.19);
}
.login-form-2 h3 {
    text-align: center;
    color: #fff;
}
.login-container form {
    padding: 10%;
}
.btnSubmit {
    width: 50%;
    border-radius: 1rem;
    padding: 1.5%;
    border: none;
    cursor: pointer;
}
.login-form-1 .btnSubmit {
    font-weight: 600;
    color: #fff;
    background-color: #0062cc;
}
.login-form-2 .btnSubmit {
    font-weight: 600;
    color: #0062cc;
    background-color: #fff;
}
.login-form-2 .ForgetPwd {
    color: #fff;
    font-weight: 600;
    text-decoration: none;
}
.login-form-1 .ForgetPwd {
    color: #0062cc;
    font-weight: 600;
    text-decoration: none;
}
```

Listing 5.9.10: Componente: css/login.css

## 5.10. Correr la Aplicación (Tomcat-Local)

Para ejecutar la aplicación con JAAS debe adicionar un parametro en la ventana de ejecución. Haga click con el botón derecho del mouse sobre el proyecto (ecommerce5), seleccione la opción [Run As] y finalmente haga click en el ítem [Run Configuration...], como se muestra en la Figura 5.3.

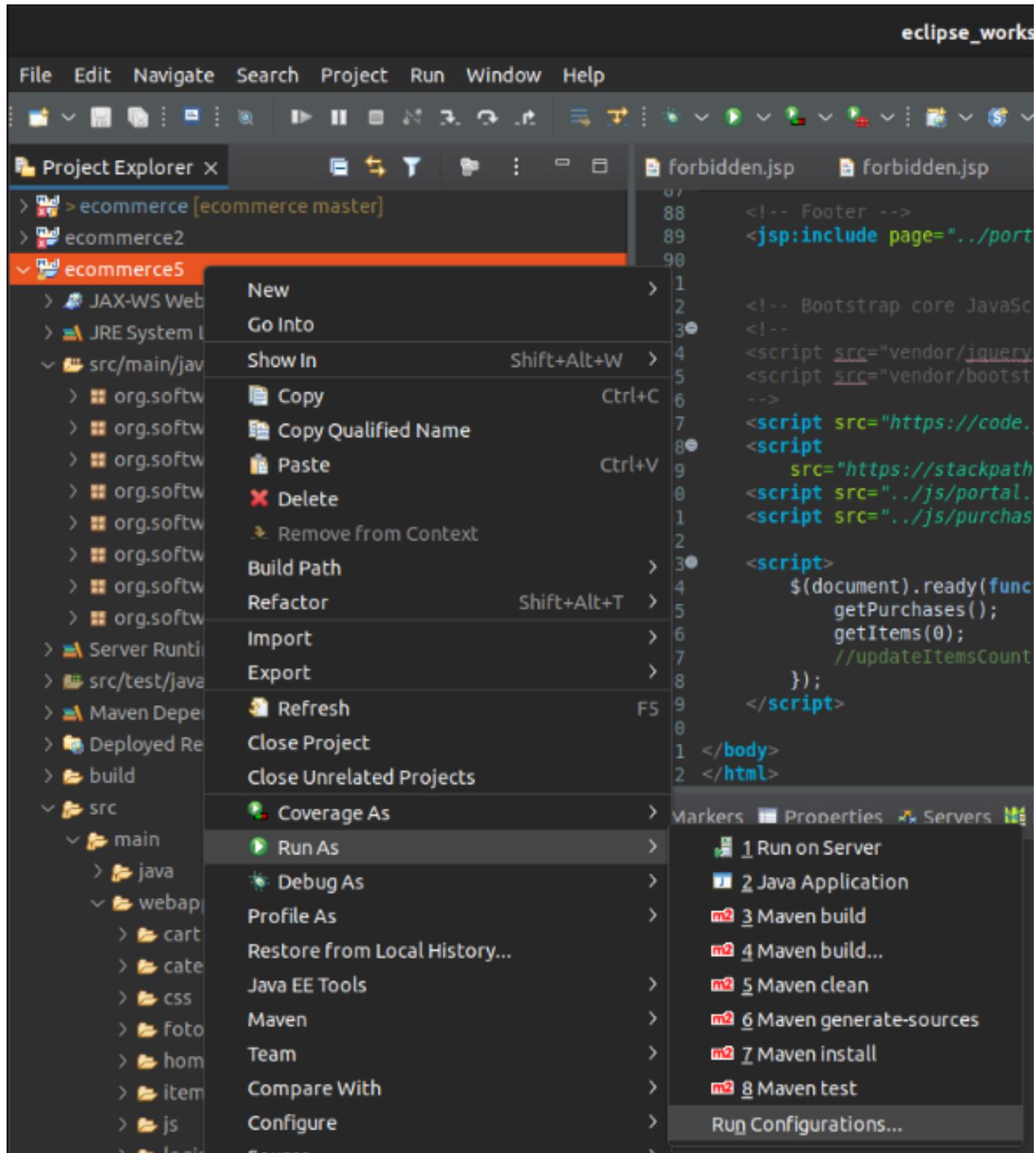


Figura 5.3: Ítem de menú de configuración de la ejecución

Adicione el siguiente parametro en la ventana de ejecución, como se muestra en la Figura 5.4.

```
-Djava.security.auth.login.config=/TOMCAT-HOME-PATH/conf/jaas.config
```

Listing 5.10.1: Parametro para cargar el mecanismo de seguridad

Reemplace TOMCAT-HOME-PATH por la ruta en donde se encuentra instalado el servidor de tomcat y finalmente haga click en el botón [Run].

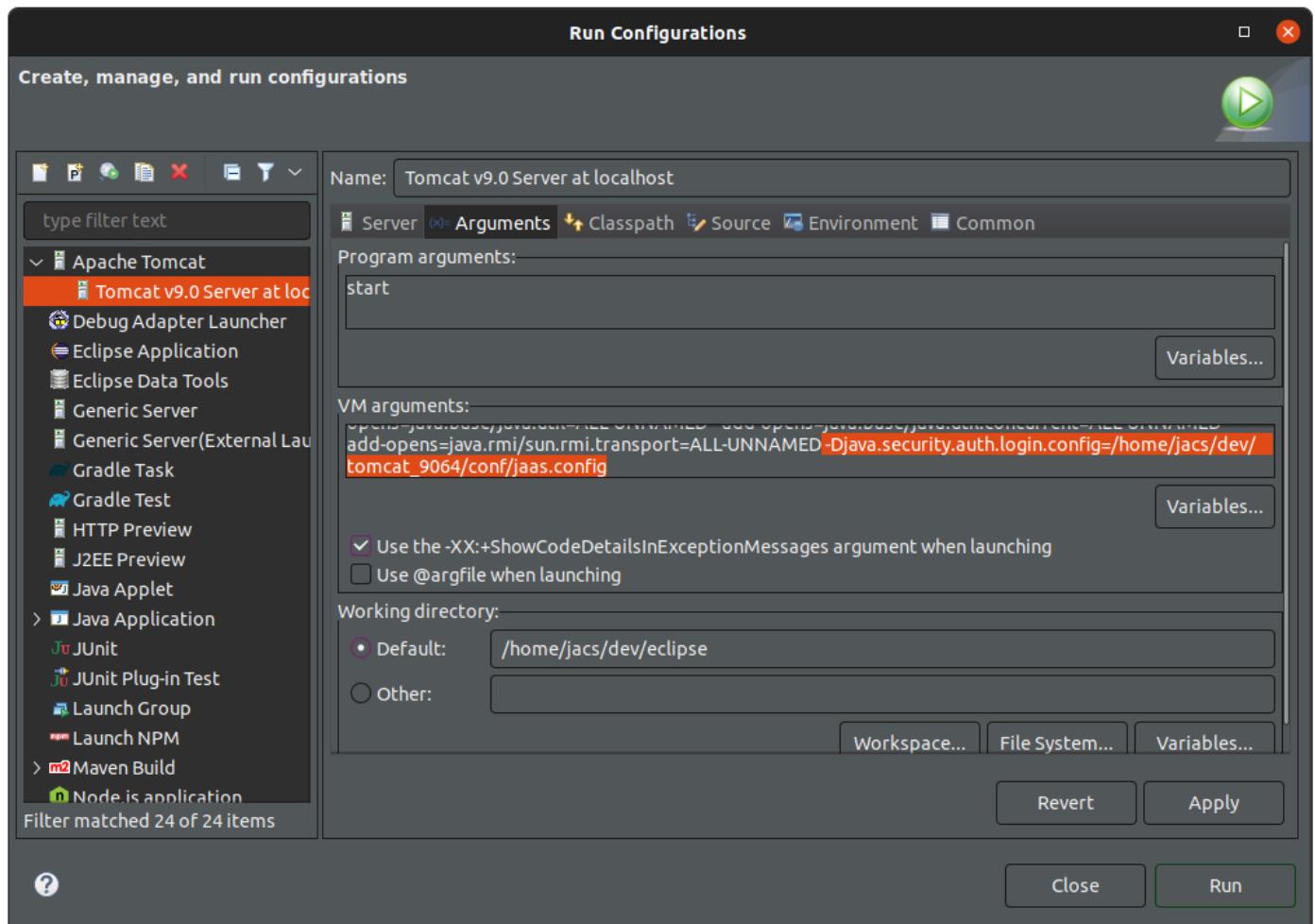


Figura 5.4: Ventana de configuración de la ejecución

## 5.11. Pruebas de Seguridad

Corra la aplicación web en el servidor, abra una ventana del navegador y haga clic arriba a la derecha en el botón [Login] Figura 5.5.

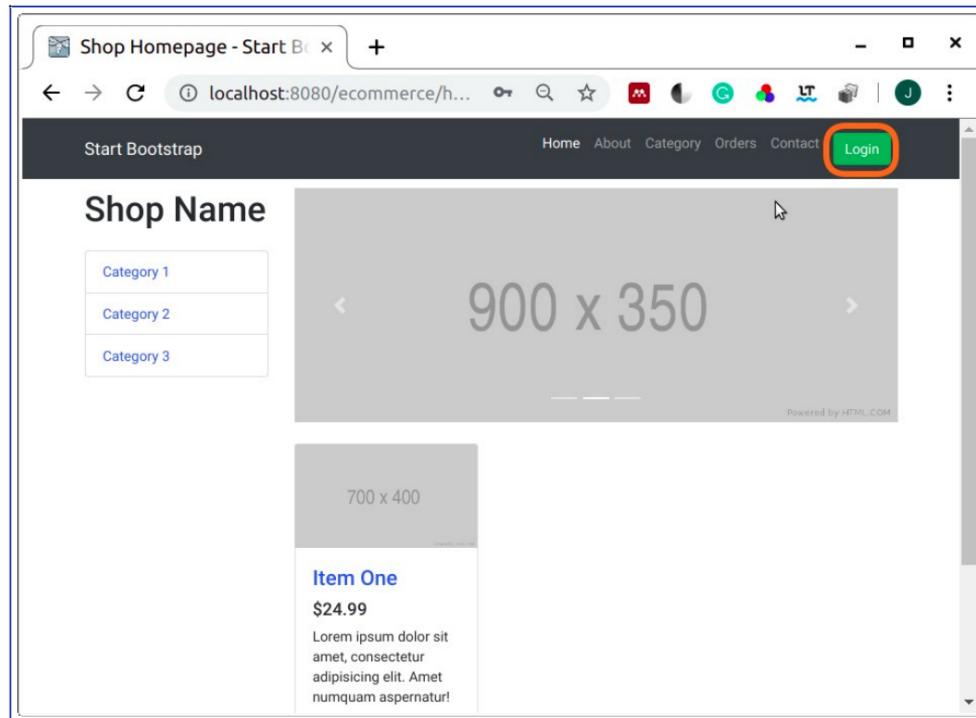


Figura 5.5: Botón de Login

El sistema mostrará el formulario de login Figura 5.6.

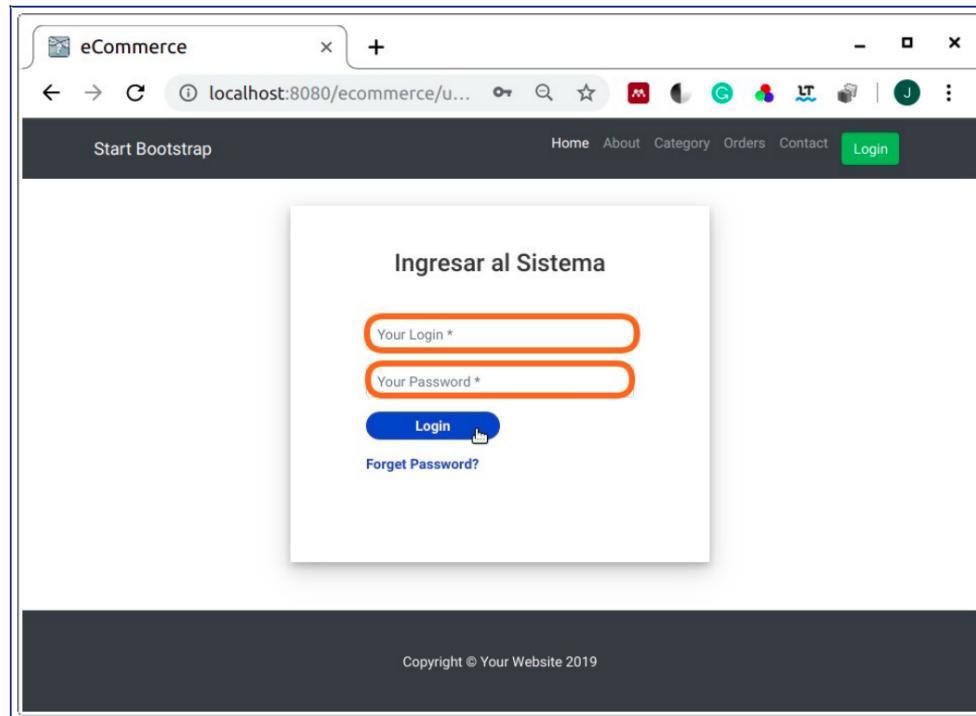


Figura 5.6: Formulario de Login

### 5.11.1. Prueba de login

Digite los siguientes usuarios y claves:

- usuario incorrecto y clave errada,
- usuario correcto y clave errada,
- usuario incorrecto y clave correcta,
- usuario correcto y clave correcta.

En caso de error al digitar las credenciales, el sistema debe mostrar la página login/error.jsp Figura 5.7.

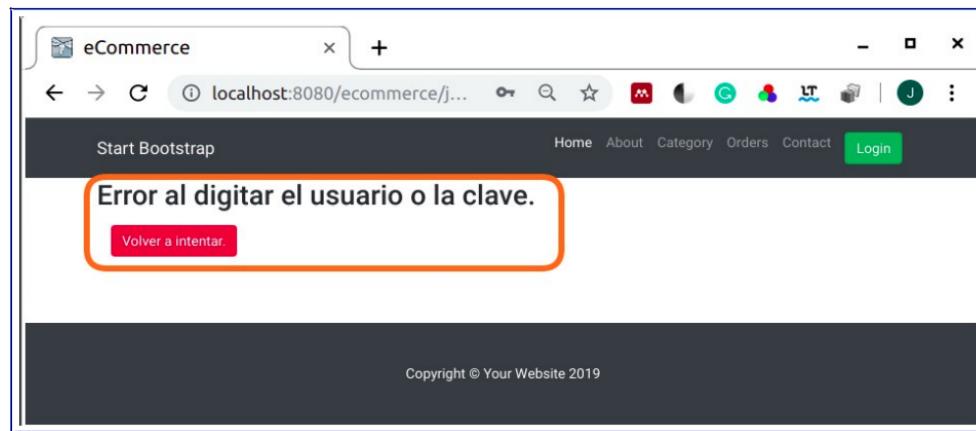


Figura 5.7: Prueba de error al iniciar sesión

### 5.11.2. Prueba del perfil Cliente

Haga clic arriba a la derecha en el botón de [Login] y digite las credenciales de un usuario con el perfil Cliente. (user=maria, password=mquintero2018). El sistema determina el usuario que ha iniciado sesión (maria) y presenta el panel de control con las opciones (funcionalidad) a la que tiene permisos. A la derecha se presenta un menú con los ítems [Panel de Control] y [Logout] Figura 5.8.

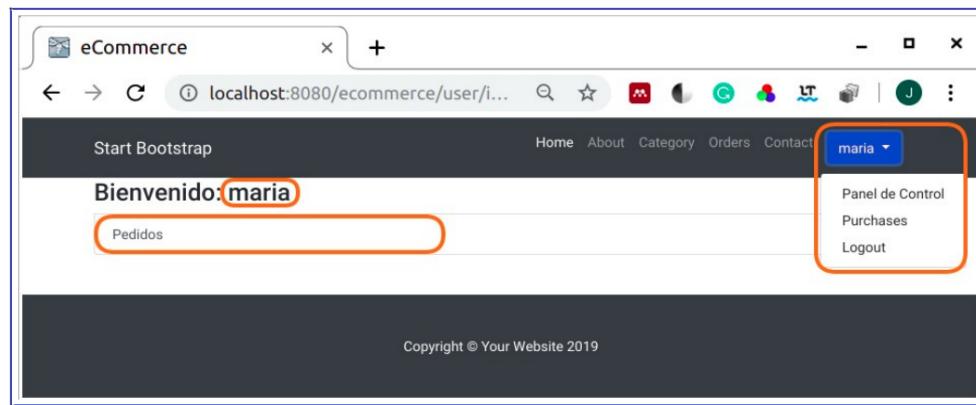


Figura 5.8: Prueba del perfil Cliente

Al seleccionar la opción de pedidos (orders), se mostrará la página [order/index.jsp]. Esta funcionalidad se implementará en una iteración posterior Figura 5.9.

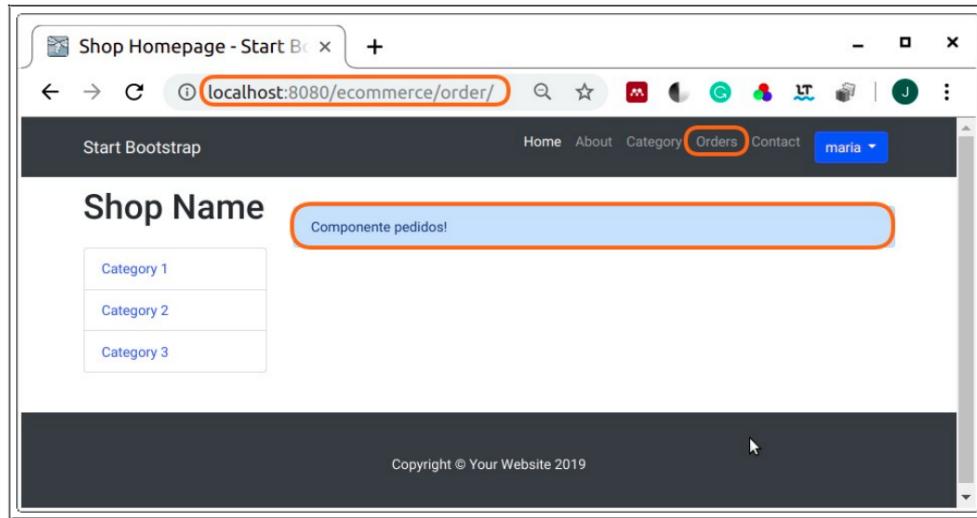


Figura 5.9: Pedidos

### 5.11.3. Prueba acceso no autorizado

Si el usuario con perfil Cliente (maria) intenta acceder a un recurso al cual no tiene permiso (category), se producirá un error [403 Forbidden] y el sistema mostrará una página con el mensaje apropiado, de acuerdo a lo que se parametrizó en el archivo de configuración de la aplicación web [web.xml] Figura 5.10.

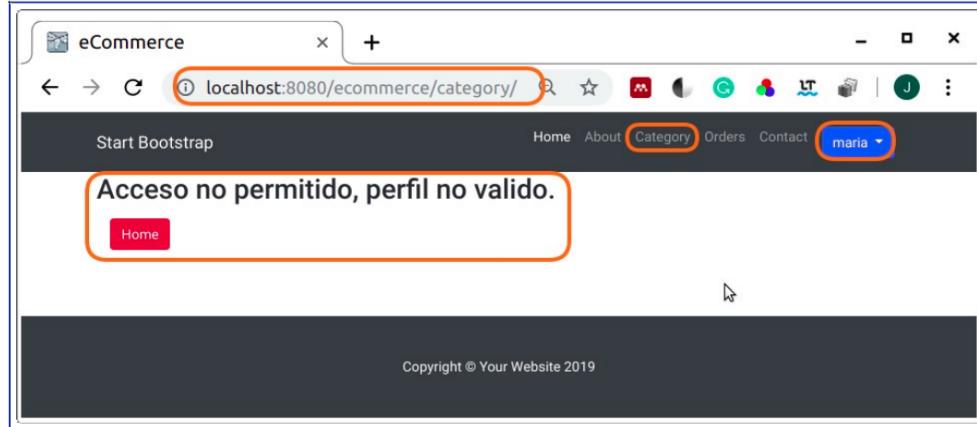


Figura 5.10: Acceso no autorizado (Forbidden)

### 5.11.4. Prueba Logout

Haga clic arriba a la derecha en el menú del usuario y seleccione la opción [Logout] Figura 5.11.

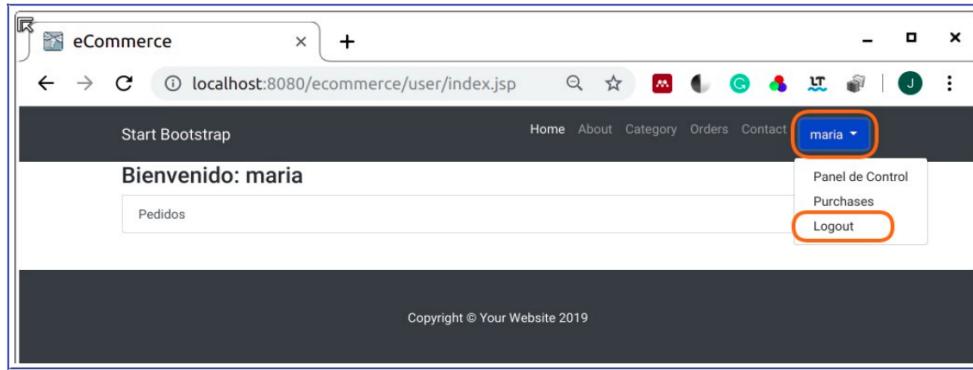


Figura 5.11: Logout

El sistema debe cerrar la sesión de usuario y presentar nuevamente el menú [Login], lo cual indica que el usuario no ha iniciado sesión Figura 5.12.

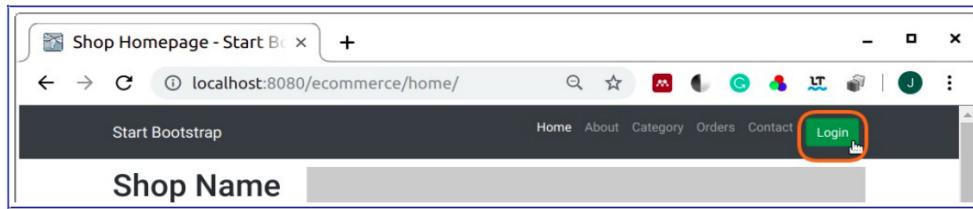


Figura 5.12: Prueba de cierre de sesión

### 5.11.5. Prueba de recurso protegido – asegurado

Si un usuario que no ha iniciado sesión intenta acceder a un recurso protegido, el sistema lanzará la ventana de login. Sin haber iniciado sesión, haga clic en la opción de menú [Category] o digite la url [http://localhost:8080/ecommerce/category] Figura 5.13.

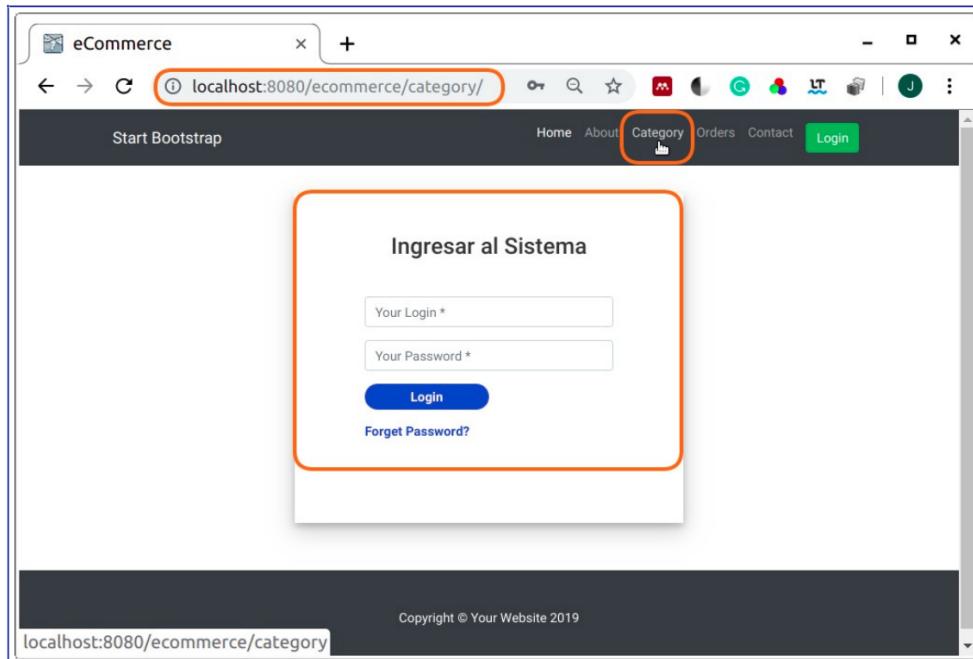


Figura 5.13: Prueba de recurso protegido

### 5.11.6. Prueba de los Servicios Web

Si el usuario no ha iniciado sesión e intenta acceder al servicio web category

`http://localhost:8080/ecommerce/ws/category`

, el sistema protege el recurso y lanza la ventana de login Figura 5.14.

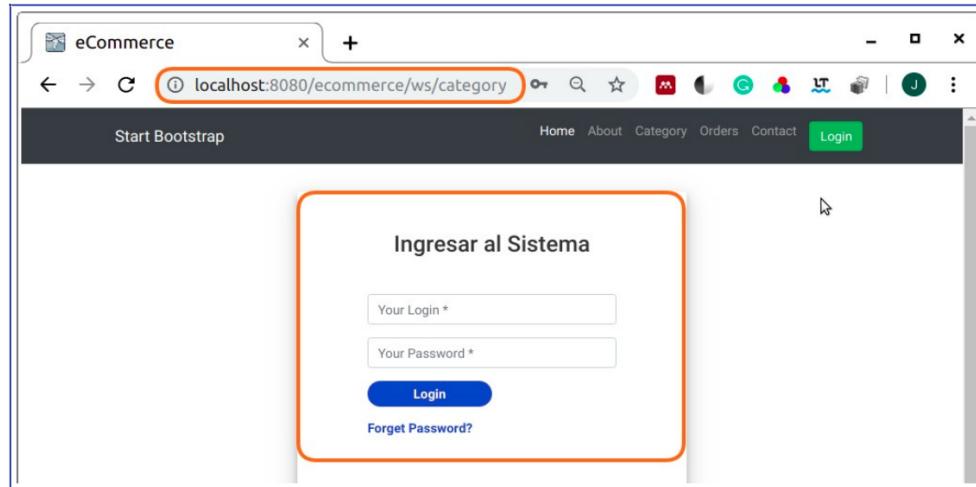


Figura 5.14: Prueba de los Servicios Web

Las pruebas de los servicios web en Postman, también obligan al usuario a iniciar sesión, protegiendo – asegurando los recursos Figura 5.15.

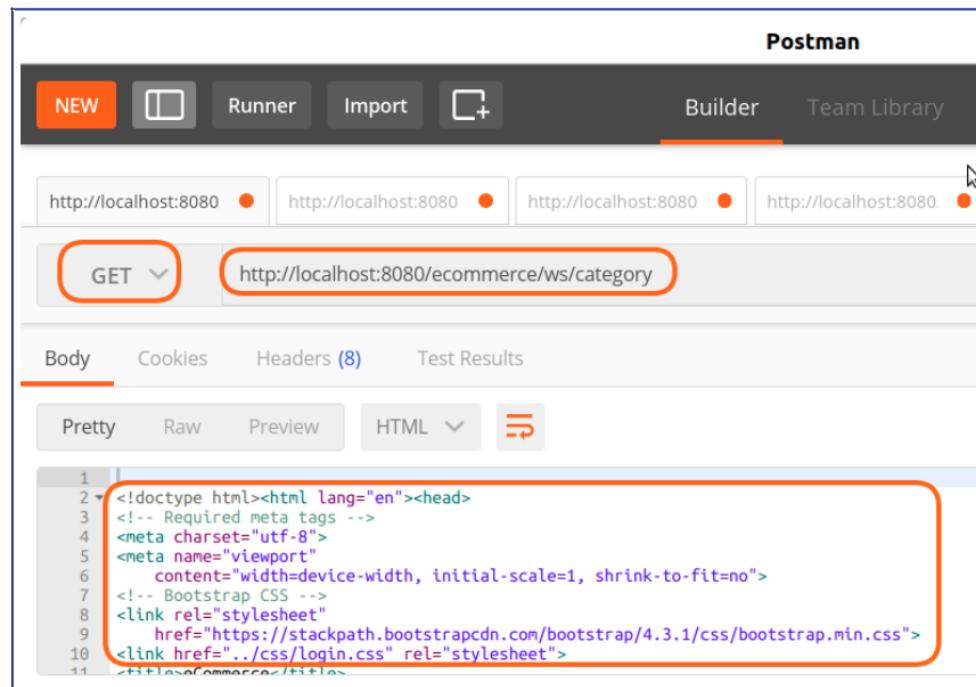


Figura 5.15: Pruebas de Servicios Web con Postman