

# TIENDA DE COMERCIO ELECTRÓNICO

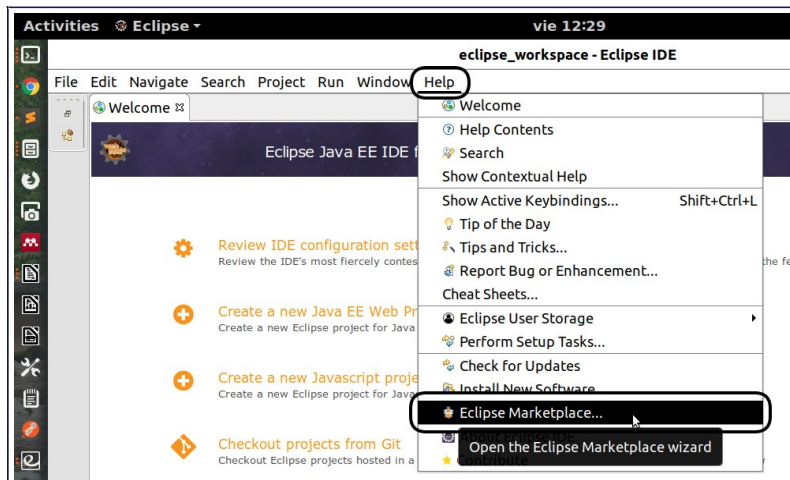
## ITERACIÓN 01 – SERVICIOS WEB

### PARTE 02 - SERVICIOS WEB RESTFULL CON WILDFLY – RESTEASY

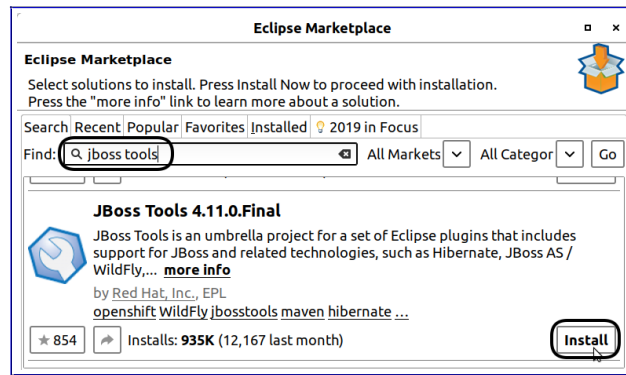
Autor: Mag. Juan Antonio Castro Silva  
Versión: 2.1 Noviembre26 de 2019 (20191126T0957)

## 1. CONFIGURACIÓN DE LAS HERRAMIENTAS DE JBOSS TOOLS

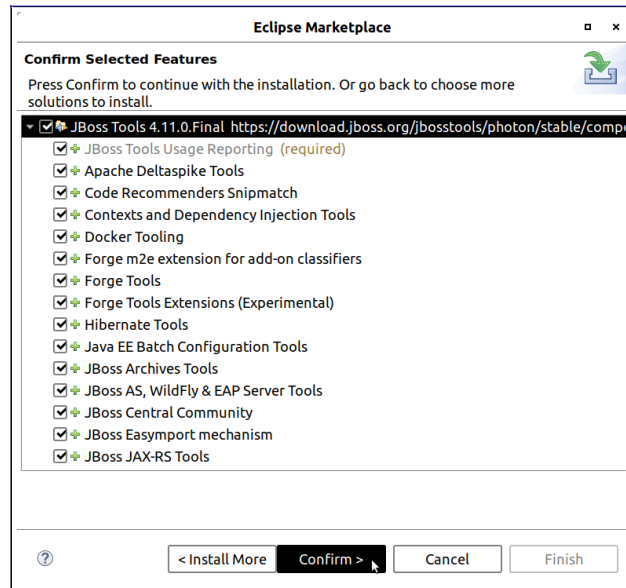
Para instalar las herramientas de jBoss [jboss tools], haga click en el menú [Help] y seleccione la opción [Eclipse Marketplace].



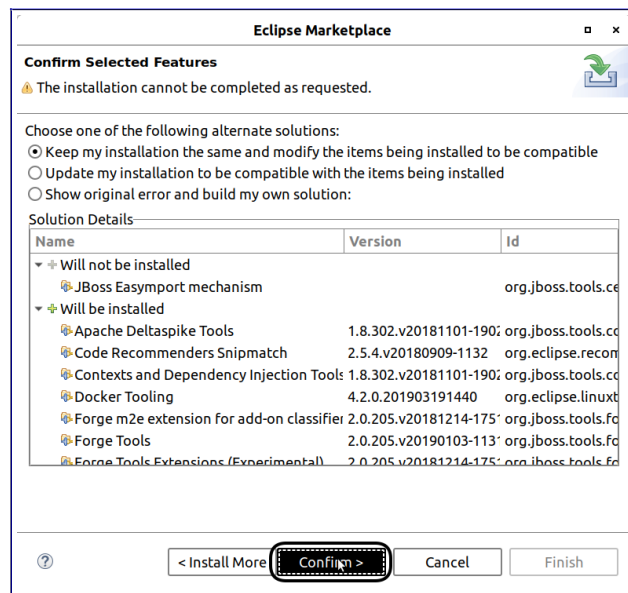
En la ventana [Eclipse Marketplace] en el cajón de texto [Find], digite jboss tools, presione la tecla [Enter], espere la respuesta de la búsqueda y en el ítem [Jboss Tools 4.11.0Final] haga click en el botón [Install].



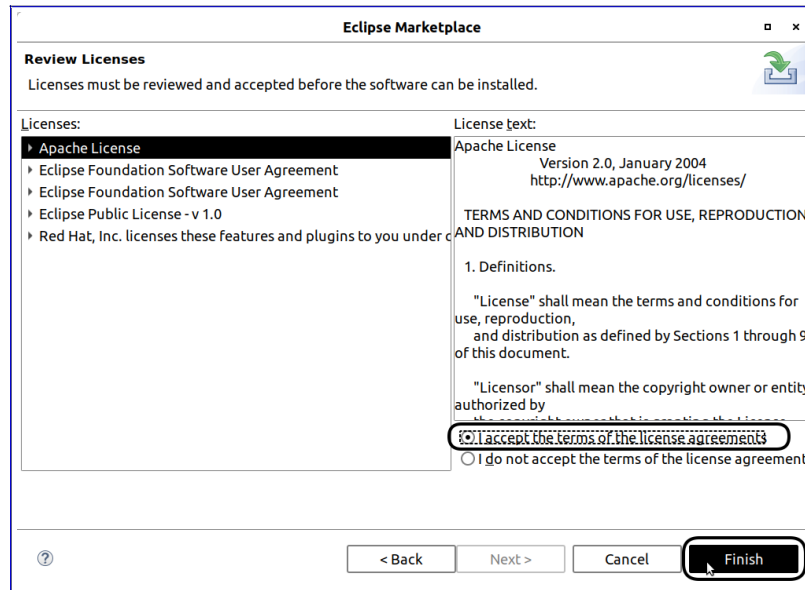
Seleccione todos los ítems (features) y haga click abajo en el botón [Confirm].



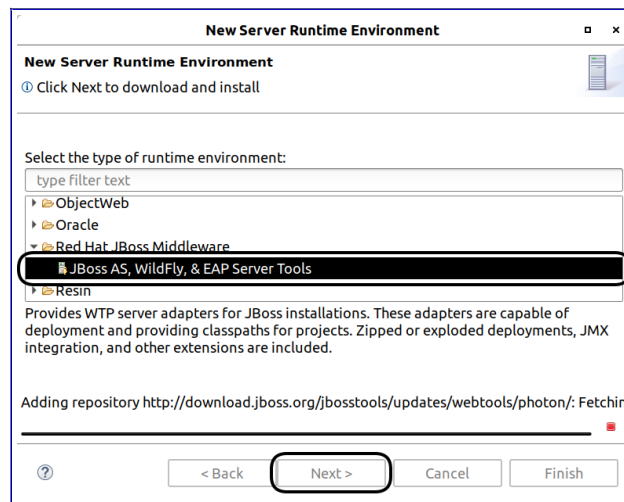
Acepte los valores por defecto y haga click abajo en el botón [Confirm].



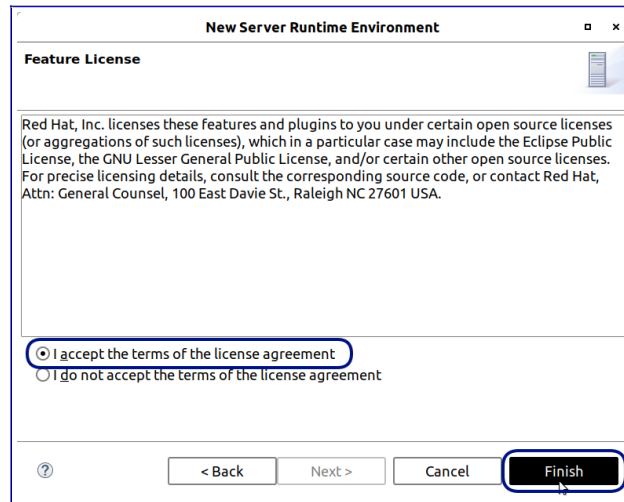
Acepte los términos de la licencia y haga click en el botón [Finish].



Seleccione [JBoss AS, Wildfly & EAP Server Tools] y haga click en el botón [Next].



Acepte los términos de la licencia y haga click en el botón [Finish].



**New Server Runtime Environment**

**Feature License**

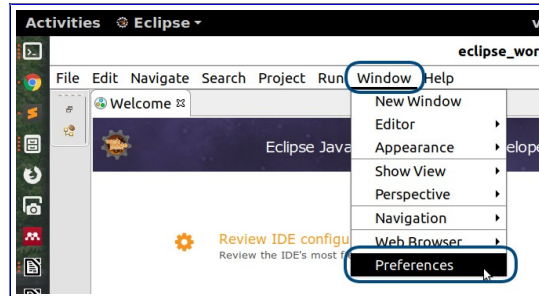
Red Hat, Inc. licenses these features and plugins to you under certain open source licenses (or aggregations of such licenses), which in a particular case may include the Eclipse Public License, the GNU Lesser General Public License, and/or certain other open source licenses. For precise licensing details, consult the corresponding source code, or contact Red Hat, Attn: General Counsel, 100 East Davie St., Raleigh NC 27601 USA.

☒ I accept the terms of the license agreement  
☐ I do not accept the terms of the license agreement

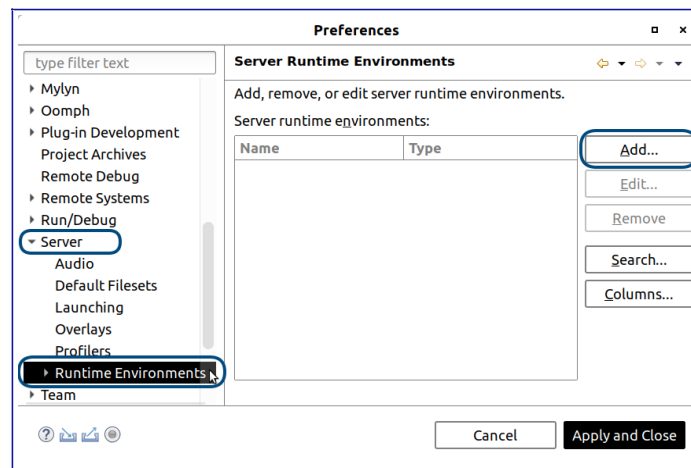
[?](#) < Back Next > Cancel **Finish**

# CONFIGURACIÓN DEL AMBIENTE DE EJECUCIÓN DEL SERVIDOR

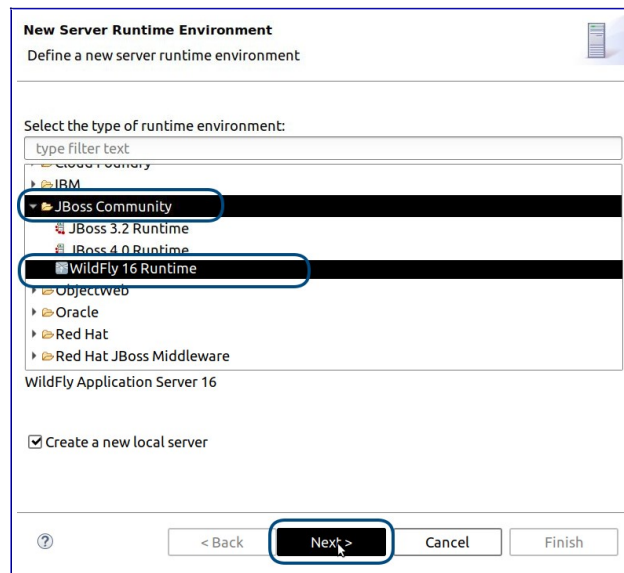
Para configurar un ambiente de ejecución de servidor (Server Runtime Environment), haga click en el menú [Window] y seleccione la opción [Preferences].



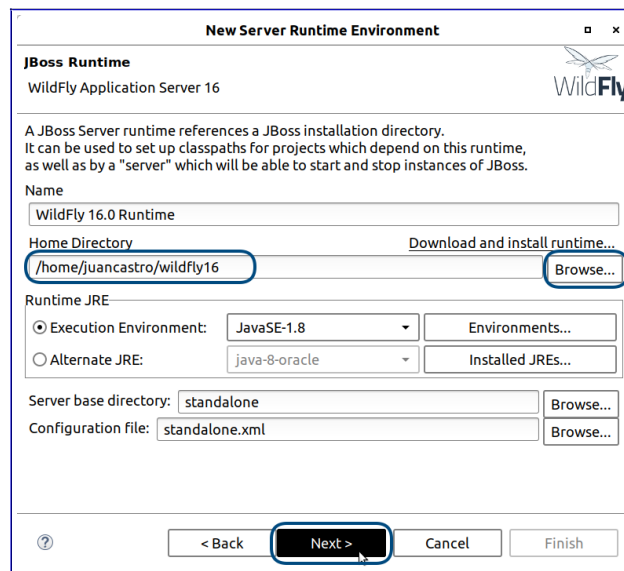
Seleccione la carpeta [Server], opción [Runtime Environment] y haga click arriba a la derecha en el botón [Add].



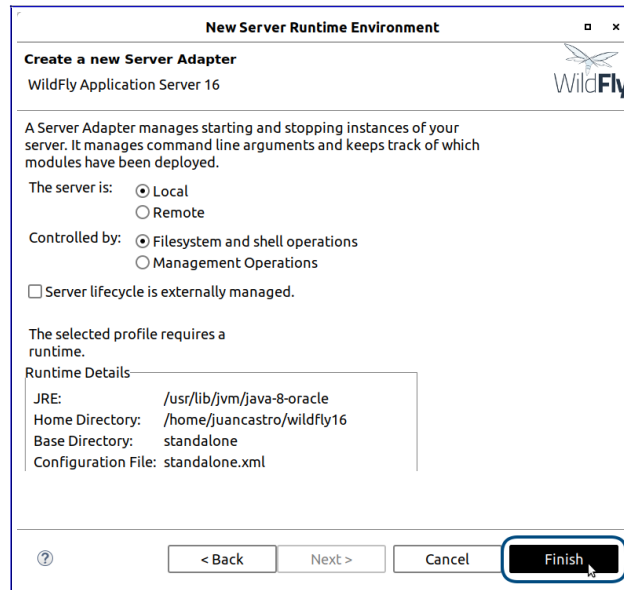
Seleccione la carpeta [JBoss Community], la opción [WildFly 16 Runtime] y haga click abajo en el botón [Next].



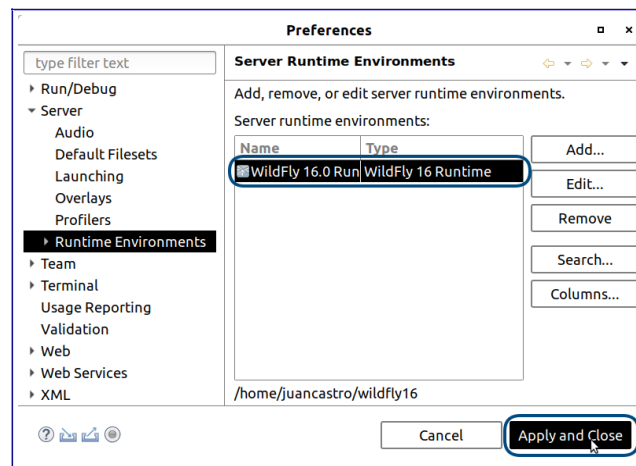
Haga click a la derecha en el botón [Browse...] y seleccione la carpeta donde tiene instalado el servidor de aplicaciones WildFly. Finalmente haga click en el botón [Next].



Acepte las opciones seleccionadas por defecto y haga click abajo en el botón [Finish].



Finalmente seleccione el ambiente de ejecución de servidor [WildFly XX.X Runtime] y haga click abajo en el botón [Apply and Close].



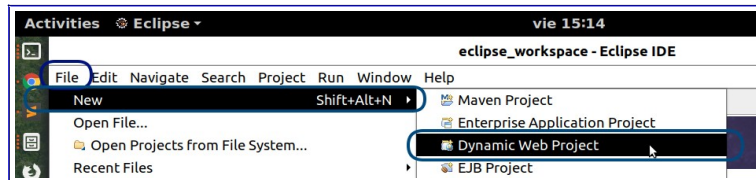


## 2. CREACIÓN DE LA APLICACIÓN WEB

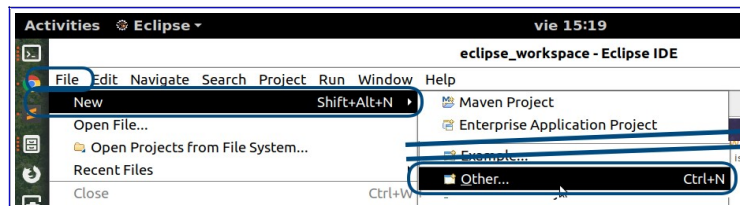
Para este proyecto se creará una aplicación basada en web.

### 2.1 CREAR PROYECTO

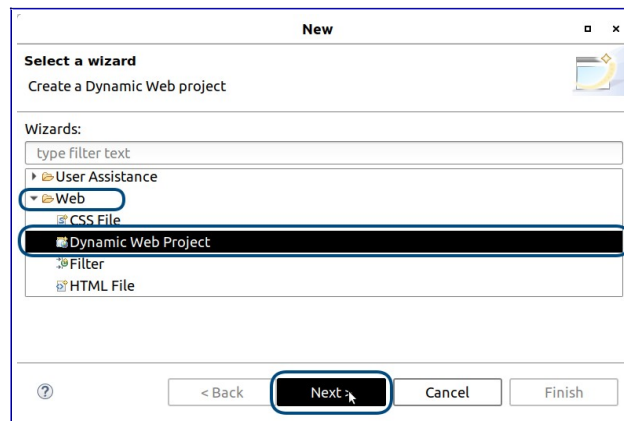
Para crear un nuevo proyecto tipo web en eclipse, haga click en el menú [File], [New] y en la opción [Dynamic Web Project].



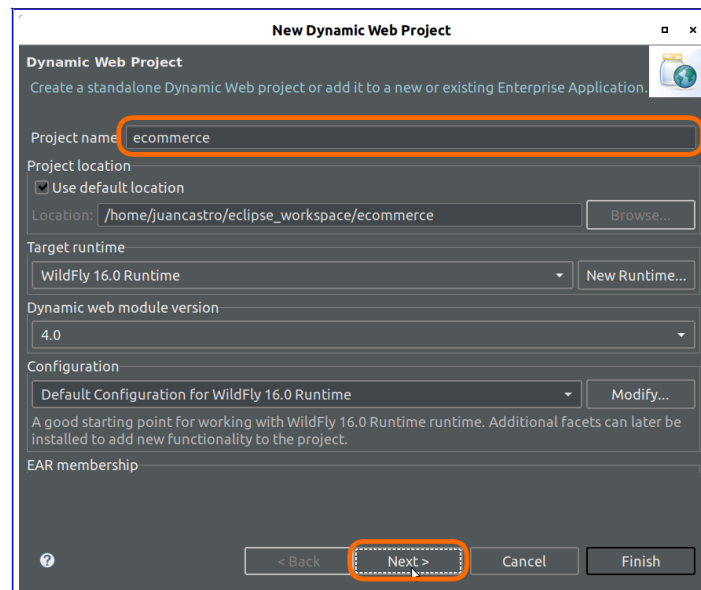
Si no encuentra la opción [Dynamic Web Project], haga click en [File], [New], [Other...].



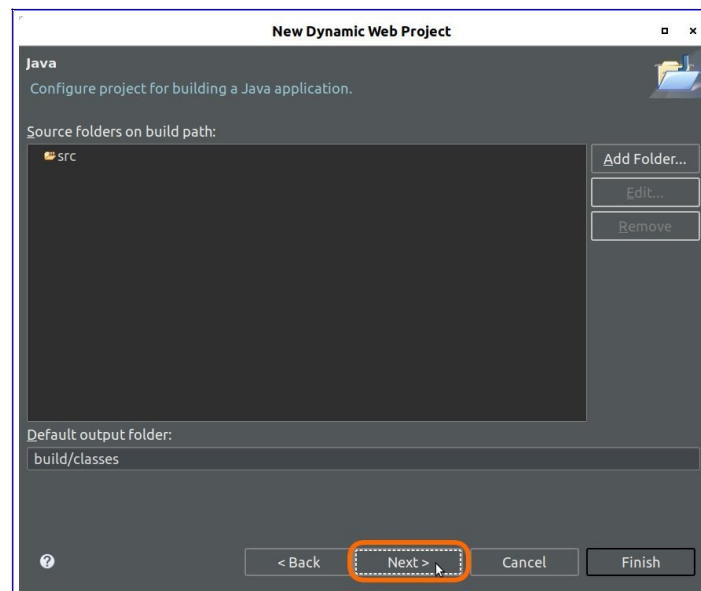
Seleccione la carpeta [Web], la opción [Dynamic Web Project] y finalmente haga click en el botón [Next].



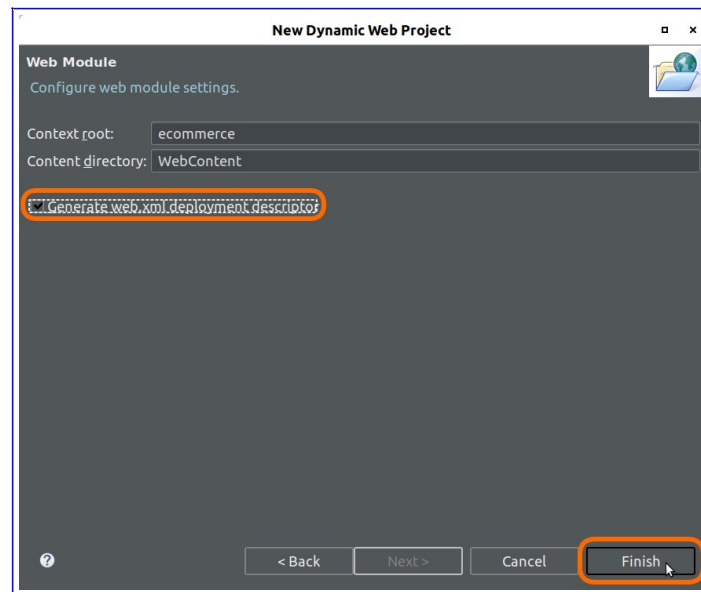
Digite el nombre del proyecto (ecommerce), verifique el ambiente de ejecución objetivo, el servidor donde quiere correr la aplicación web [Target runtime] (WildFly XX.X Runtime) y haga click abajo en el botón de [Next].



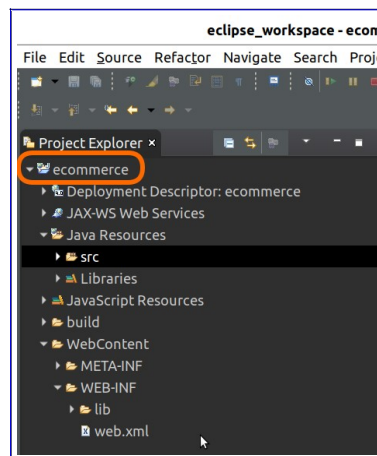
Acepte los valores por defecto y haga click abajo en el botón [Next].



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



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

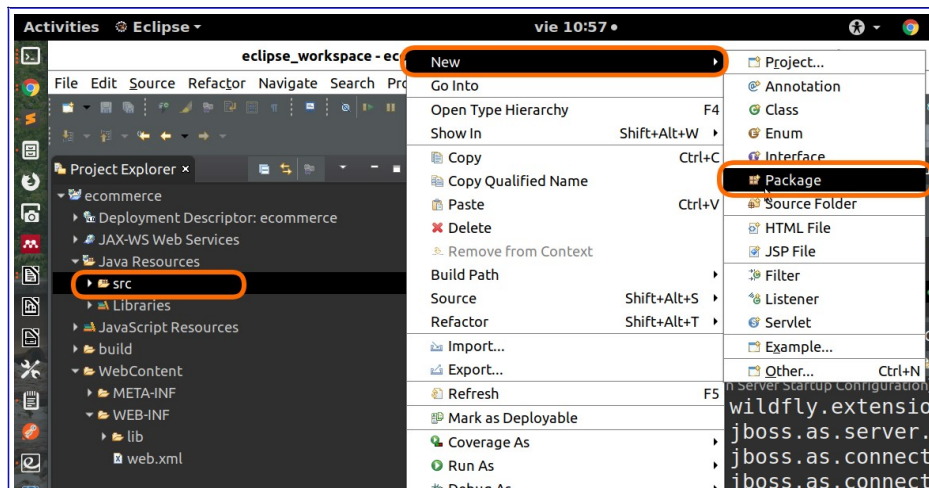


## 2.2 CREAR PAQUETES

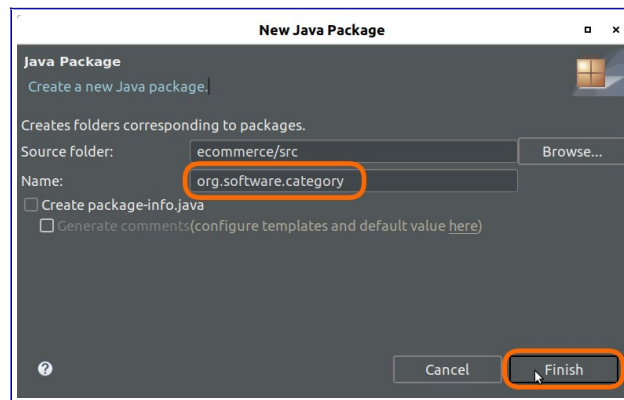
La creación de las clases y paquetes es una implementación del Diagrama de Clases. Para esta primera iteración el diagrama de clases comprende los siguientes paquetes y clases.

- org.software.category
  - Category
  - CategoryList
  - CategoryService
- org.software.product
  - Product
  - ProductList
  - ProductService
- org.software.util
  - DataBase
  - WebConfig

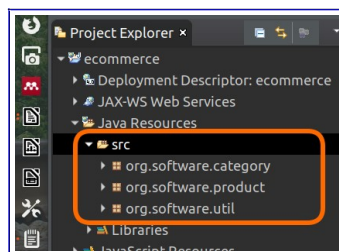
Para crear un paquete haga click con el botón derecho en la carpeta [src], seleccione la opción [New] y finalmente en haga click en [Package].



Digite el nombre del paquete en el cajón de texto de nombre [Name] (org.software.category) y haga click en el botón de [Finish].



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

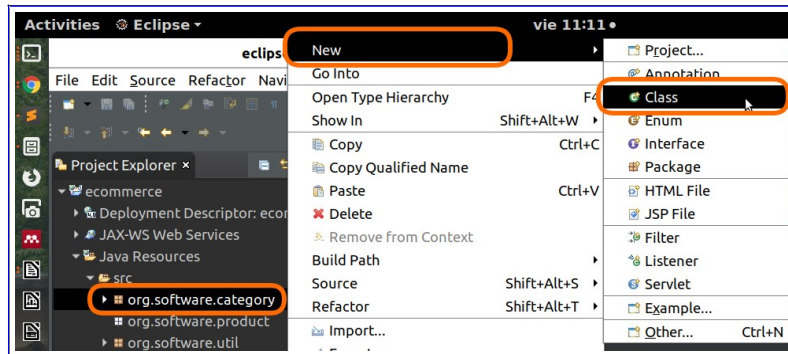


## 2.3 CREACIÓN DE LAS CLASES

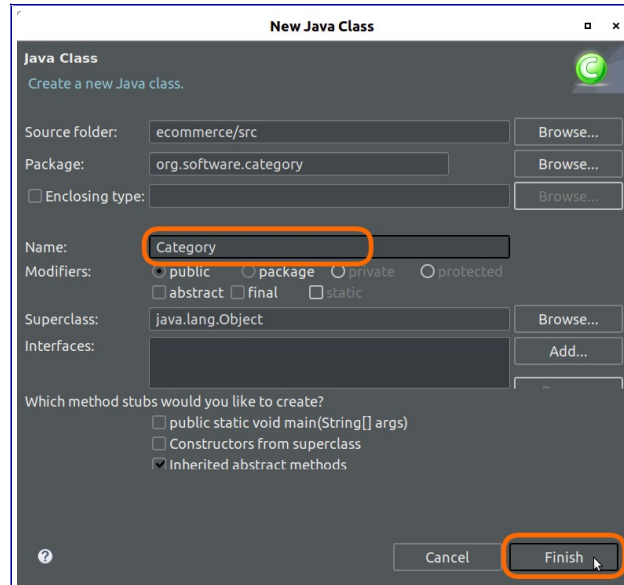
Dentro del paquete `org.software.category` debe crear tres clases:

- `Category`
- `CategoryList`
- `CategoryService`

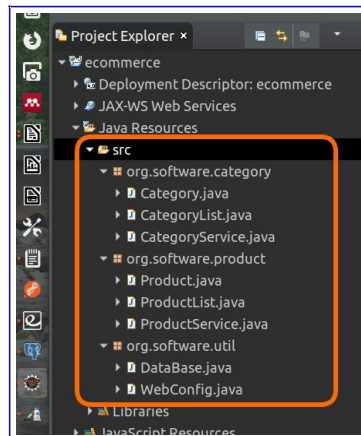
Para crear una clase haga click con el botón derecho del mouse en el paquete `[org.software.category]`, seleccione `[New]` y finalmente haga click en `[Class]`.



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]`.



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



El código fuente de las clases se muestra a continuación:

#### Clase WebConfig:

```
001 package org.software.util;
002
003 import javax.ws.rs.ApplicationPath;
004 import javax.ws.rs.core.Application;
005
006 @ApplicationPath("/ws")
007 public class WebConfig extends Application {
008
009 }
```

#### Clase DataBase:

```
001 package org.software.util;
002
003 import java.sql.Connection;
004 import java.sql.DriverManager;
005 import java.sql.PreparedStatement;
006 import java.sql.ResultSet;
007 import java.sql.SQLException;
008 import java.sql.Statement;
009
010 public class DataBase {
011
012     public Connection getConnection(String profile) {
013         Connection connection = null;
014         String driver = "org.postgresql.Driver";
015         String url = "jdbc:postgresql://localhost:5432/ecommerce";
016
017         String user = "";
018         String password = "";
```

```

019
020     if (profile.equals("admin")){
021         user = "ecommerce_admin";
022         password = "234567";
023     }
024     if (profile.equals("client")){
025         user = "ecommerce_client";
026         password = "345678";
027     }
028     if (profile.equals("guest")){
029         user = "ecommerce_guest";
030         password = "456789";
031     }
032
033     try {
034         Class.forName(driver);
035         connection = DriverManager.getConnection(url, user, password);
036     } catch (Exception e) {
037         System.out.println("Error: " + e.toString());
038     }
039     return connection;
040 }
041
042 public void closeObject(Connection connection) {
043     try {
044         connection.close();
045     } catch (SQLException e) {
046         e.printStackTrace();
047     }
048 }
049
050 public void closeObject(PreparedStatement preparedStatement) {
051     try {
052         preparedStatement.close();
053     } catch (SQLException e) {
054         e.printStackTrace();
055     }
056 }
057
058 public void closeObject(Statement statement) {
059     try {
060         statement.close();
061     } catch (SQLException e) {
062         e.printStackTrace();
063     }
064 }
065
066 public void closeObject(ResultSet resultSet) {
067     try {
068         resultSet.close();
069     } catch (SQLException e) {
070         e.printStackTrace();
071     }
072 }
073 }

```



## Clase Category

```
001 package org.software.category;
002
003 import java.sql.Statement;
004
005 public class Category {
006     private long id;
007     private int published;
008     private String name;
009     private String icon;
010
011     public Category() {
012         super();
013     }
014
015     public Category(long id, int published, String name, String icon) {
016         super();
017         this.id = id;
018         this.published = published;
019         this.name = name;
020         this.icon = icon;
021     }
022
023     public long getId() {
024         return id;
025     }
026
027     public void setId(long id) {
028         this.id = id;
029     }
030
031     public int getPublished() {
032         return published;
033     }
034
035     public void setPublished(int published) {
036         this.published = published;
037     }
038
039     public String getName() {
040         return name;
041     }
042
043     public void setName(String name) {
044         this.name = name;
045     }
046
047     public String getIcon() {
048         return icon;
049     }
050
051     public void setIcon(String icon) {
052         this.icon = icon;
053     }
054
055 }
056 }
```

## Clase CategoryList:

```
001 package org.software.category;
002
003 import java.util.List;
004
005 import javax.xml.bind.annotation.XmlElement;
006 import javax.xml.bind.annotation.XmlRootElement;
007
008 @XmlRootElement(name="categories")
009 public class CategoryList {
010
011     private List<Category> items;
012
013     public CategoryList() {
014     }
015
016     public CategoryList(List<Category> items) {
017         this.items = items;
018     }
019
020     @XmlElement(name = "data")
021     public List<Category> getItems() {
022         return items;
023     }
024
025 }
```

## Clase CategoryService:

```
001 package org.software.category;
002
003 import java.sql.Connection;
004 import java.sql.PreparedStatement;
005 import java.sql.ResultSet;
006 import java.sql.Statement;
007 import java.util.ArrayList;
008
009 import javax.ws.rs.Consumes;
010 import javax.ws.rs.DELETE;
011 import javax.ws.rs.GET;
012 import javax.ws.rs.POST;
013 import javax.ws.rs.PUT;
014 import javax.ws.rs.Path;
015 import javax.ws.rs.PathParam;
016 import javax.ws.rs.Produces;
017 import javax.ws.rs.core.Response;
018
019 import org.software.util.DataBase;
020
021 @Path("/category")
022 public class CategoryService {
023
024     @POST
025     @Path("/")
026     @Consumes({ "application/json" })
027     @Produces("application/json")
028     public Response add(Category category) {
029         DataBase database = new DataBase();
030         Connection connection1 = null;
031         PreparedStatement preparedStatement1 = null;
032         String sql = "";
033         String mensaje = "";
034         int inserted = 0;
035         try {
036             connection1 = database.getConnection("admin");
037
038             sql = "INSERT INTO categories(name, icon)";
039             sql += " VALUES (?, ?)";
040
041             preparedStatement1 = connection1.prepareStatement(sql);
042             preparedStatement1.setString(1, category.getName());
043             preparedStatement1.setString(2, category.getIcon());
044             inserted = preparedStatement1.executeUpdate();
045         } catch (Exception e) {
046             System.out.println("Error: " + e.toString());
047         } finally {
048             database.closeObject(preparedStatement1);
049             database.closeObject(connection1);
050         }
051
052         if (inserted > 0) {
053             mensaje = "{\"mensaje\":\"Adicionar OK\"}";
054             return Response.status(200).entity(mensaje).build();
055         } else {
056             mensaje = "{\"mensaje\":\"Error al adicionar\"}";
057             return Response.status(400).entity(mensaje).build();
058         }
059     }
}
```

```

060 @GET
061 @Path("/")
062 @Produces("application/json")
063 // @Produces("application/xml")
064 public CategoryList getAll() {
065     ArrayList<Category> categoryList = new ArrayList<Category>();
066     DataBase database = new DataBase();
067     Connection connection1 = null;
068     Statement statement1 = null;
069     ResultSet rs1 = null;
070     String sql = "";
071
072     try {
073         connection1 = database.getConnection("guest");
074         statement1 = connection1.createStatement();
075
076         sql = "select * from categories";
077
078         rs1 = statement1.executeQuery(sql);
079         while (rs1.next()) {
080             int id = rs1.getInt("id");
081             int published = rs1.getInt("published");
082             String name = rs1.getString("name");
083             String icon = rs1.getString("icon");
084
085             Category category = new Category();
086             category.setId(id);
087             category.setPublished(published);
088             category.setName(name);
089             category.setIcon(icon);
090
091             categoryList.add(category);
092         }
093     } catch (Exception e) {
094         System.out.println("Error: " + e.toString());
095     } finally {
096         database.closeObject(rs1);
097         database.closeObject(statement1);
098         database.closeObject(connection1);
099     }
100     return new CategoryList(categoryList);
101 }
102
103 @PUT
104 @Path("/{id}")
105 @Consumes({ "application/json" })
106 @Produces("application/json")
107 public Response update(Category category, @PathParam(value = "id") int id) {
108     DataBase database = new DataBase();
109     Connection connection1 = null;
110     PreparedStatement preparedStatement1 = null;
111     String sql = "";
112     String mensaje = "";
113     int updates = 0;
114     try {
115         connection1 = database.getConnection("admin");
116
117         sql = "UPDATE categories SET published=?, name=?, icon=?";
118         sql += " WHERE id=?";
119
120         preparedStatement1 = connection1.prepareStatement(sql);

```

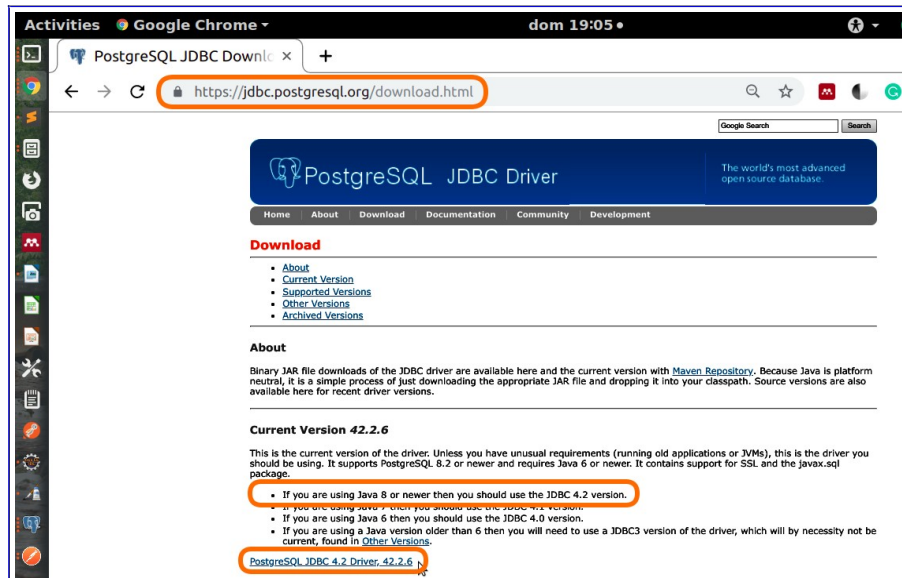
```

121         preparedStatement1.setInt(1, category.getPublished());
122         preparedStatement1.setString(2, category.getName());
123         preparedStatement1.setString(3, category.getIcon());
124         preparedStatement1.setInt(4, id);
125         updates = preparedStatement1.executeUpdate();
126     } catch (Exception e) {
127         System.out.println("Error: " + e.toString());
128     } finally {
129         database.closeObject(preparedStatement1);
130         database.closeObject(connection1);
131     }
132
133     if (updates > 0) {
134         mensaje = "{\"mensaje\":\"Modificar OK\"}";
135         return Response.status(200).entity(mensaje).build();
136     } else {
137         mensaje = "{\"mensaje\":\"Error al modificar\"}";
138         return Response.status(400).entity(mensaje).build();
139     }
140 }
141
142 @DELETE
143 @Path("/{id}")
144 @Consumes({ "application/json" })
145 @Produces("application/json")
146 public Response adicionar(@PathParam(value = "id") int id) {
147     DataBase database = new DataBase();
148     Connection connection1 = null;
149     PreparedStatement preparedStatement1 = null;
150     String sql = "";
151     String mensaje = "";
152     int deleteds = 0;
153     try {
154         connection1 = database.getConnection("admin");
155
156         sql = "DELETE FROM categories WHERE id=?";
157
158         preparedStatement1 = connection1.prepareStatement(sql);
159         preparedStatement1.setInt(1, id);
160         deleteds = preparedStatement1.executeUpdate();
161     } catch (Exception e) {
162         System.out.println("Error: " + e.toString());
163     } finally {
164         database.closeObject(preparedStatement1);
165         database.closeObject(connection1);
166     }
167
168     if (deleteds > 0) {
169         mensaje = "{\"mensaje\":\"Eliminar OK\"}";
170         return Response.status(200).entity(mensaje).build();
171     } else {
172         mensaje = "{\"mensaje\":\"Error al eliminar\"}";
173         return Response.status(400).entity(mensaje).build();
174     }
175 }
176
177 }

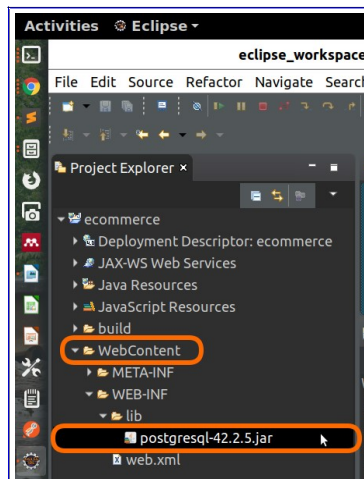
```

## 2.4 INSTALACIÓN DEL DRIVER JDBC

Para acceder a la base de datos (postgresql) debe instalar el driver JDBC, el driver lo puede descargar del sitio web (<https://jdbc.postgresql.org/download.html>). En esta página web se advierte que si se utiliza el JDK 1.8 se debe utilizar el Driver versión JDBC42.

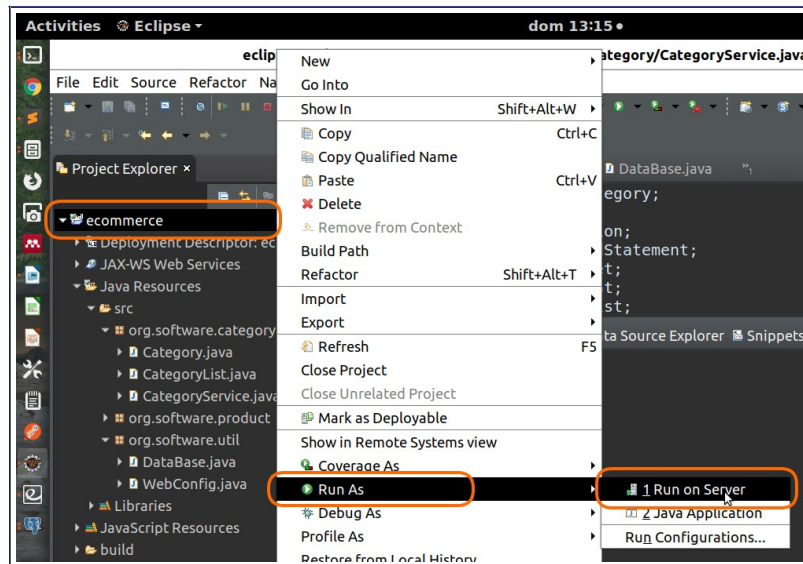


Copie el Driver JDBC en la carpeta WebContent/WEB-INF/lib.

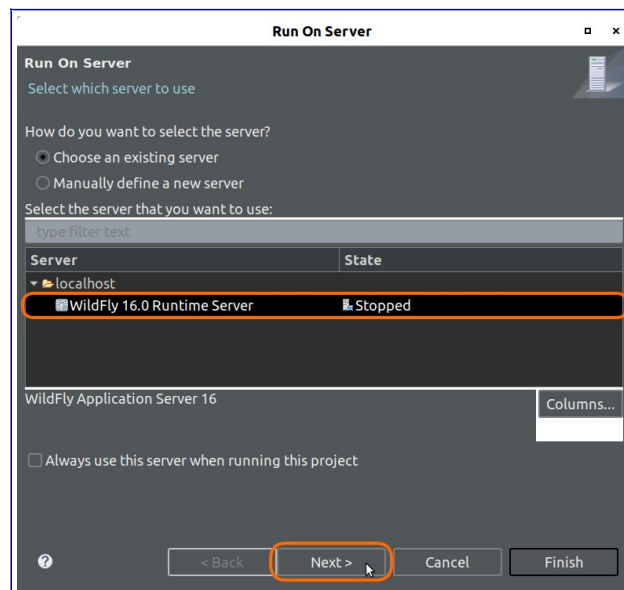


### 3. CORRER EL PROYECTO

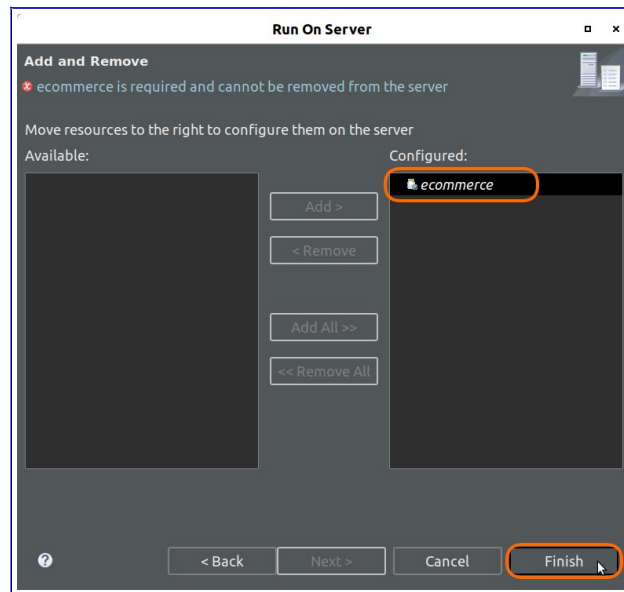
Para correr el proyecto, haga click con el botón derecho del mouse sobre el proyecto (ecommerce), menú [Run As] y seleccione la opción [Run on Server].



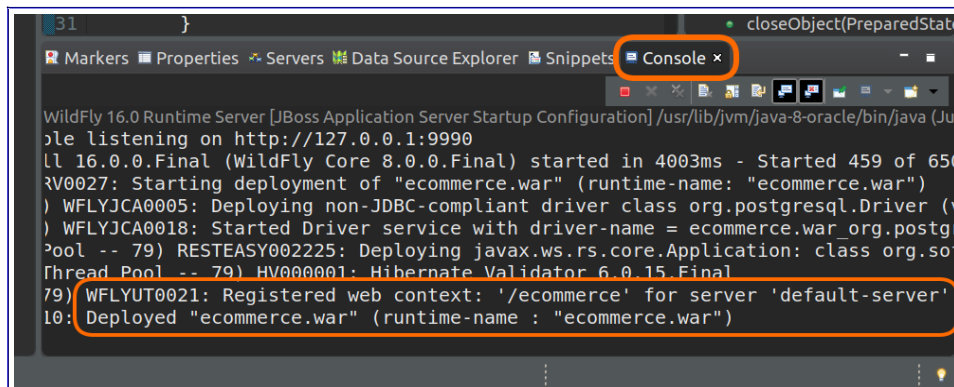
Verifique el servidor de aplicaciones donde desea correr la aplicación web [Runtime Server] y haga click abajo en el botón [Next].



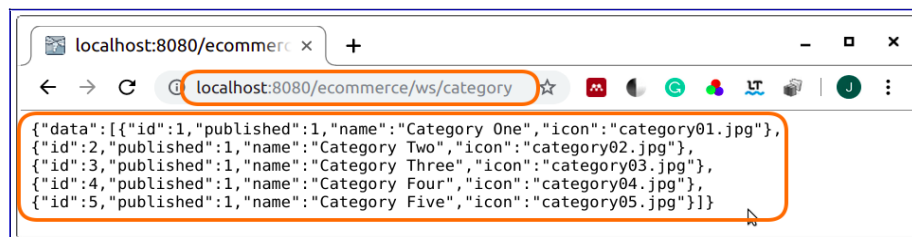
Verifique que el proyecto que desea correr esta configurado y haga click abajo en el botón [Finish].



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.

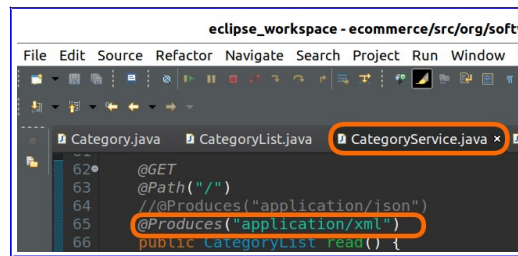


Para probar la operación de lectura del CRUD (Read - método GET), en un navegador abra la url (<http://localhost:8080/ecommerce/ws/category>). Podrá ver el listado de todos los registros de la tabla categories en formato JSON.



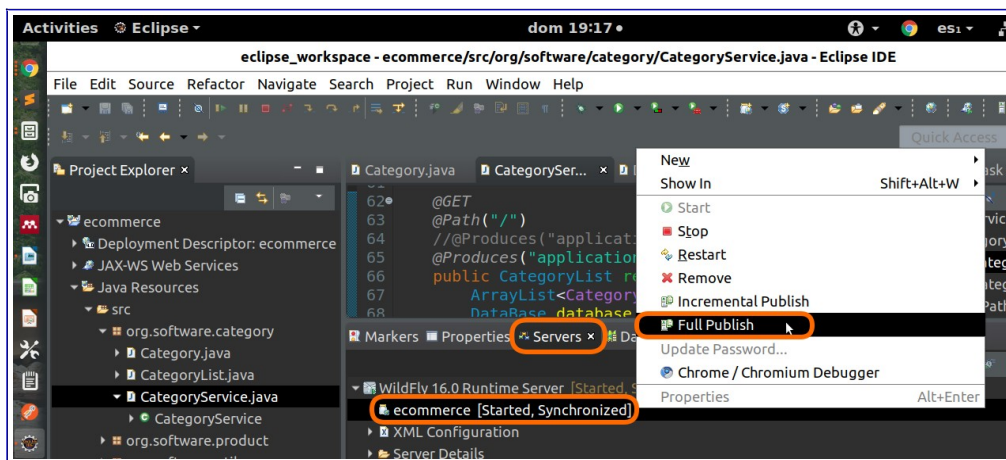


Si desea obtener los datos en formato XML haga los cambios en la clase [CategoryService].

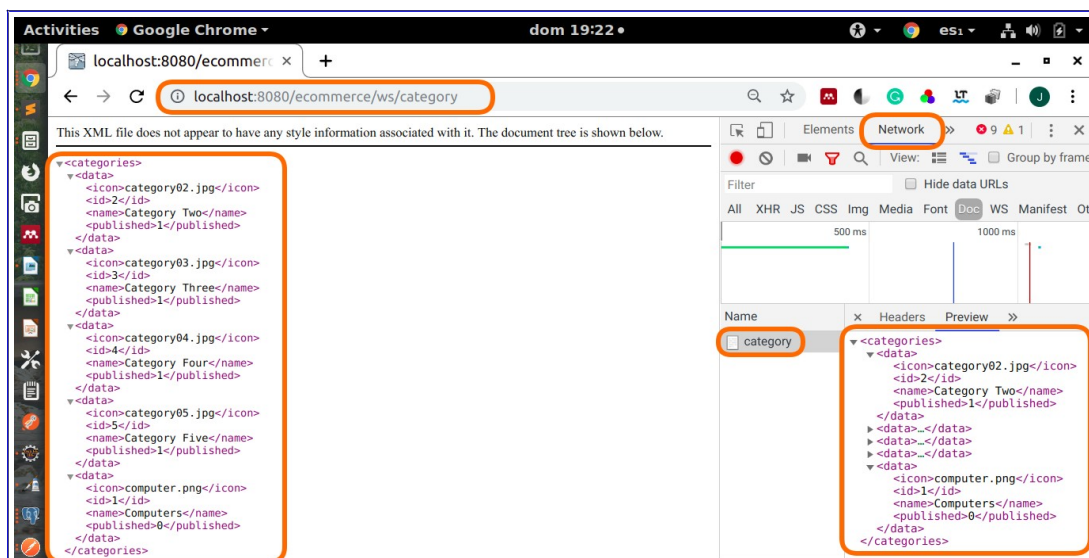


```
62 * @GET
63 * @Path("/")
64 * // @Produces("application/json")
65 * @Produces("application/xml")
66 * public CategoryList read() {
```

Para desplegar la aplicación con los nuevos cambios, abajo en la ficha [Servers], haga click con el botón derecho del mouse sobre la aplicación web [ecommerce] y seleccione la opción [Full Publish].

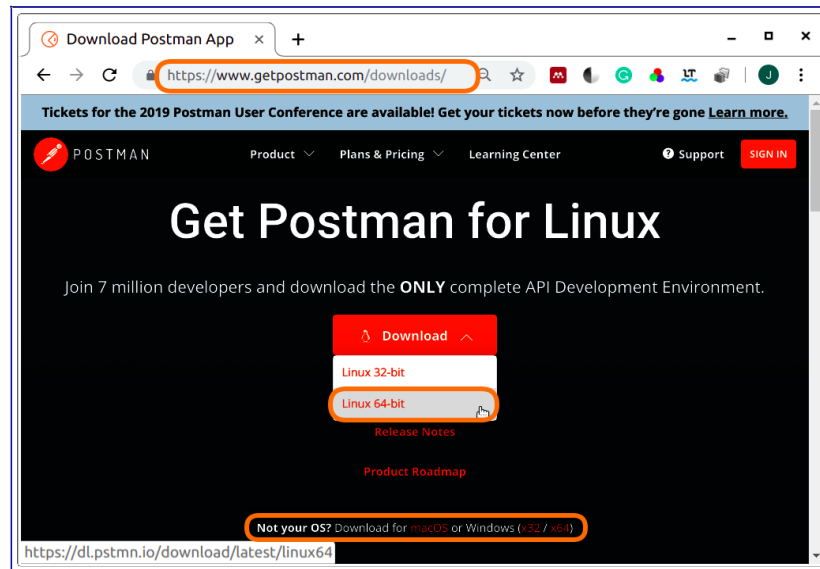


En un navegador abra la url (<http://localhost:8080/ecommerce/ws/category>) y podrá ver los resultados en formato XML.

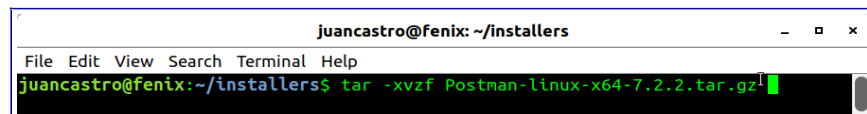


## 4. PROBAR LOS SERVICIOS WEB

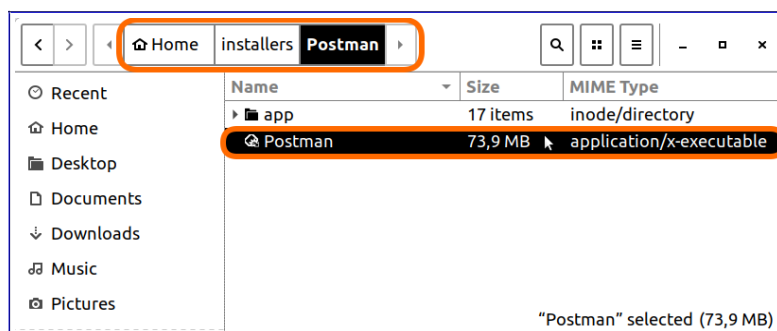
La prueba de los servicios web la podemos hacer de diversas formas, para este proyecto se utilizará la aplicación Postman. Descargue el instalador de Postman desde la página web oficial (<https://www.getpostman.com/downloads/>). Existen instaladores para diversos sistemas operativos.



Para instalar Postman abra una ventana de Terminal – Shell, ingrese a la carpeta donde lo descargo y ejecute el comando (tar – xvzf) para descomprimirlo.



Para ejecutar Postman, ingrese al carpeta donde lo instalo y haga click en el archivo ejecutable [Postman].



Para utilizar Postman debe crear una cuenta de usuario.

Postman

File Edit View Help

Enterprise user? [Sign in here](#)

### WHY SIGN UP?

- Organize all your API development within Postman Workspaces
- Sync your Postman data across devices
- Backup your data to the Postman cloud
- It's free!

### Create Account

[Sign in instead?](#)

Email  
email@server.com

Username  
username

Password  
\*\*\*\*\* [SHOW](#)

[Create free account](#)

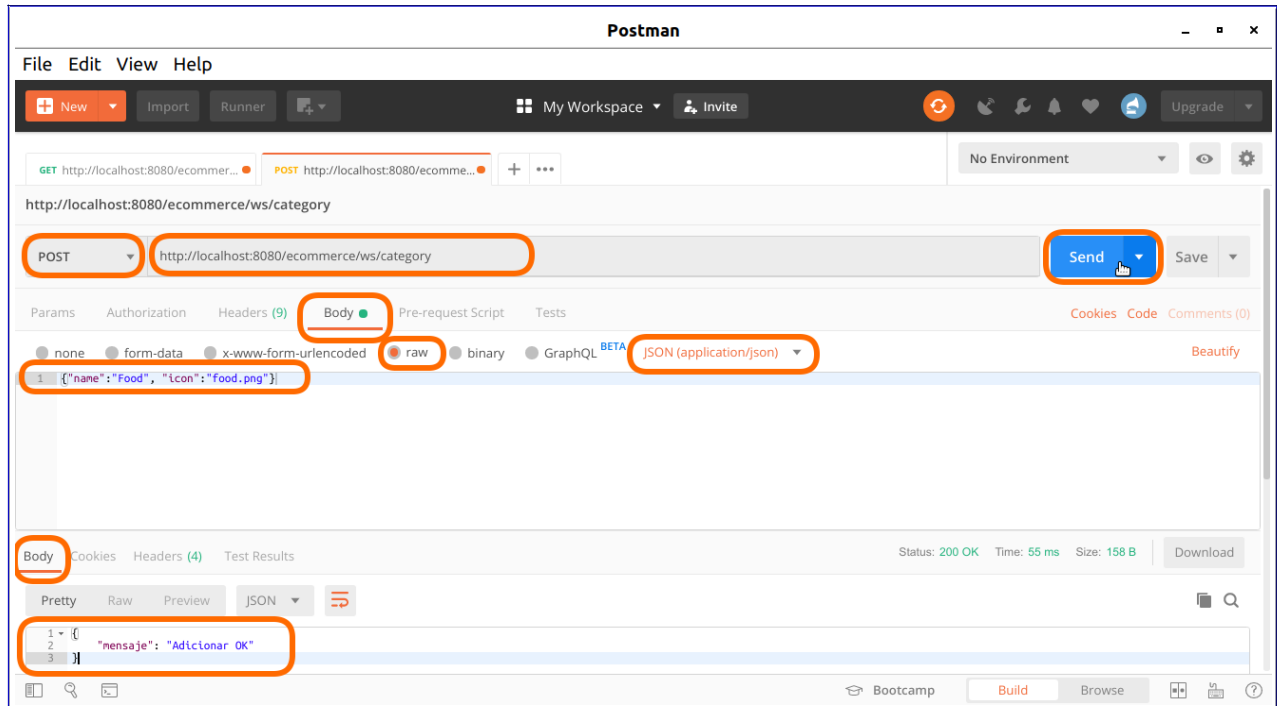
or

[Skip signing in and take me straight to the app](#)

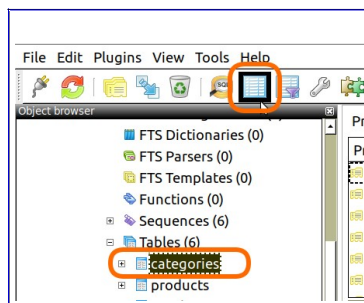
Digite los datos requeridos y haga click en el botón [Create free account].

## 4.1 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 click en el botón [Send]. Abajo en la ficha [Body] podrá ver la respuesta del servidor en formato JSON.



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].

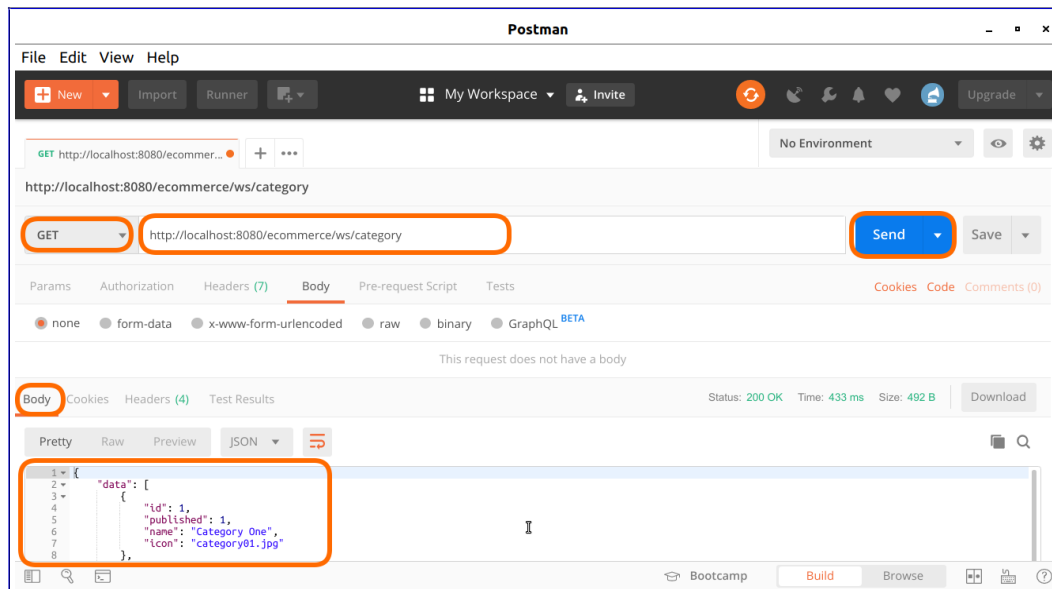


Se puede ver que el ultimo registro creado tiene el name: Food y el icon: food.png.

id	published	name	icon	created_at	updated_at
1	1	Category One	category01.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
2	1	Category Two	category02.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
3	1	Category Three	category03.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
4	1	Category Four	category04.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
5	1	Category Five	category05.jpg	2018-11-19 12:02:19.49203	2018-11-19 12:02:19.49203
6	0	Food	food.png	2019-07-14 16:11:51.972782	2019-07-14 16:11:51.972782

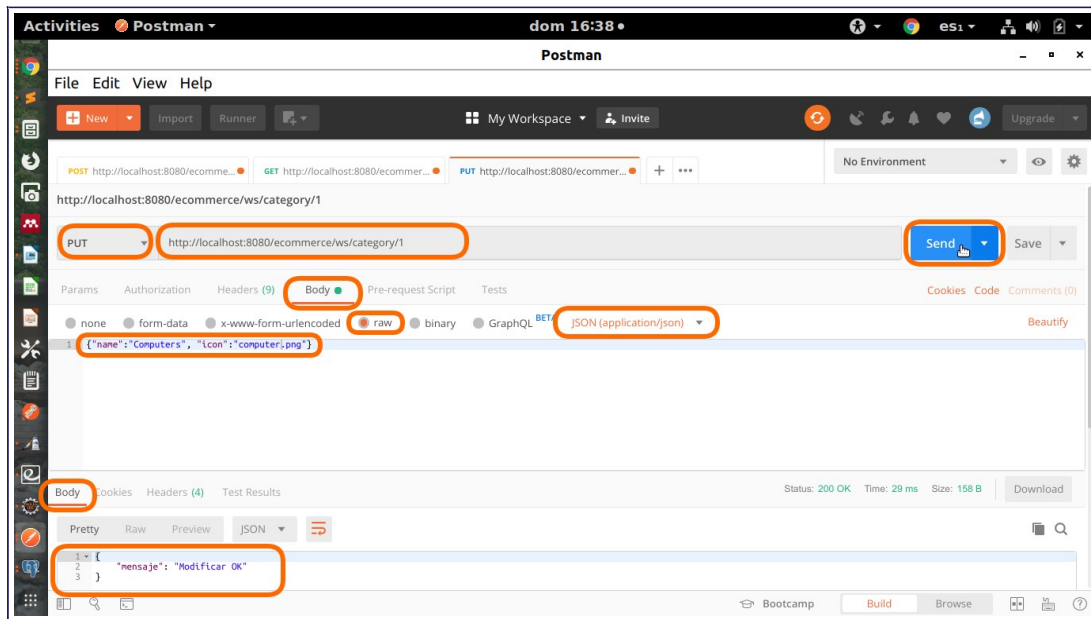
## 4.2 Read:

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.



## 4.3 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.

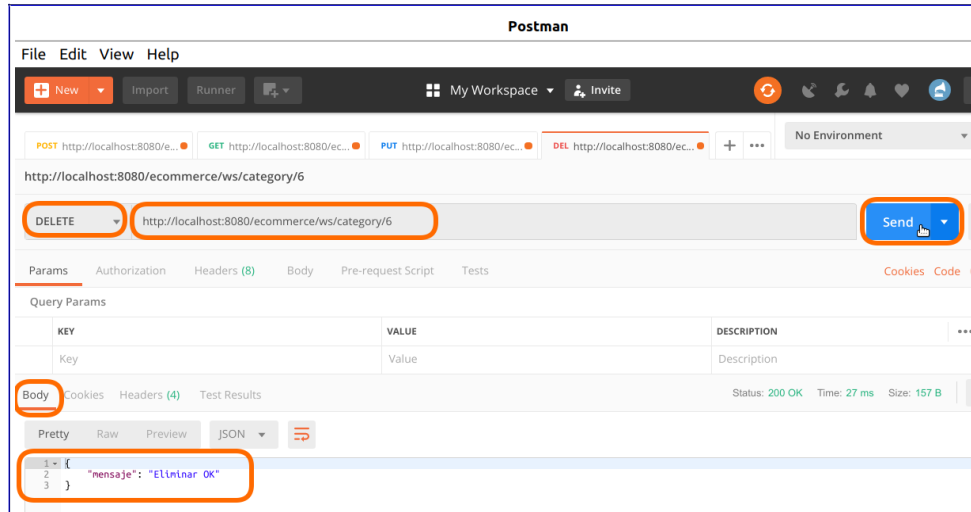


Verifique en la base de datos (ecommerce) que las modificaciones se estn realizando en la tabla [categories]. Se puede ver que el registro con id: 1 fue modificado.

	id	published	name	icon	created_at	updated_at
	PK1 bigint	integer	character varying(255)	character varying(255)	timestamp without time zone	timestamp
1	1	0	Computers	computer.png	2018-11-19 12:02:19.49203	2018-11-1
2	2	1	Category Two	category02.jpg	2018-11-19 12:02:19.49203	2018-11-1
3	3	1	Category Three	category03.jpg	2018-11-19 12:02:19.49203	2018-11-1

## 4.4 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.



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

The screenshot shows a database table named 'categories' with the following data:

id	published	name	icon
1	0	Computers	computer.png
2	1	Category Two	category02.jpg
3	1	Category Three	category03.jpg
4	1	Category Four	category04.jpg
5	1	Category Five	category05.jpg