TECHNOLOGY

Caltech | Center for Technology & Management Education

Setup Maven Project and Class Orchestration

TECHNOLOGY

Caltech | Center for Technology & Management Education

Working on Designing Model for Admin Dashboard

You Already Know

Before we begin, let's recall what we have covered till now:

Core Java

Maven

Recap

Design and develop web pages for Admin Dashboard and End User Web App

- Web pages are designed using HTML CSS, Bootstrap, and various angular APIs.

Create Tables for the Admin Dashboard and End User Web App

- Tables are created using SQL in MySQL.

A Day in the Life of a Full Stack Developer

As a full-stack web developer, our key role is to develop both client and server software.

- Angular and Node can be used to build front end of the web page.

- Spring Boot, Java, and MySQL or MongoDB can be used to build at the back end.

A Day in the Life of a Full Stack Developer

Now, Bob needs to develop various classes for the backend. So, Bob brainstorms a bit and finds a solution.

Let me use Java and Maven to develop classes for backend.

In this lesson, we will learn Java and Maven skills to create Maven Project, develop various classes for the backend, and help Bob to complete his task effectively and quickly.

Learning Objectives

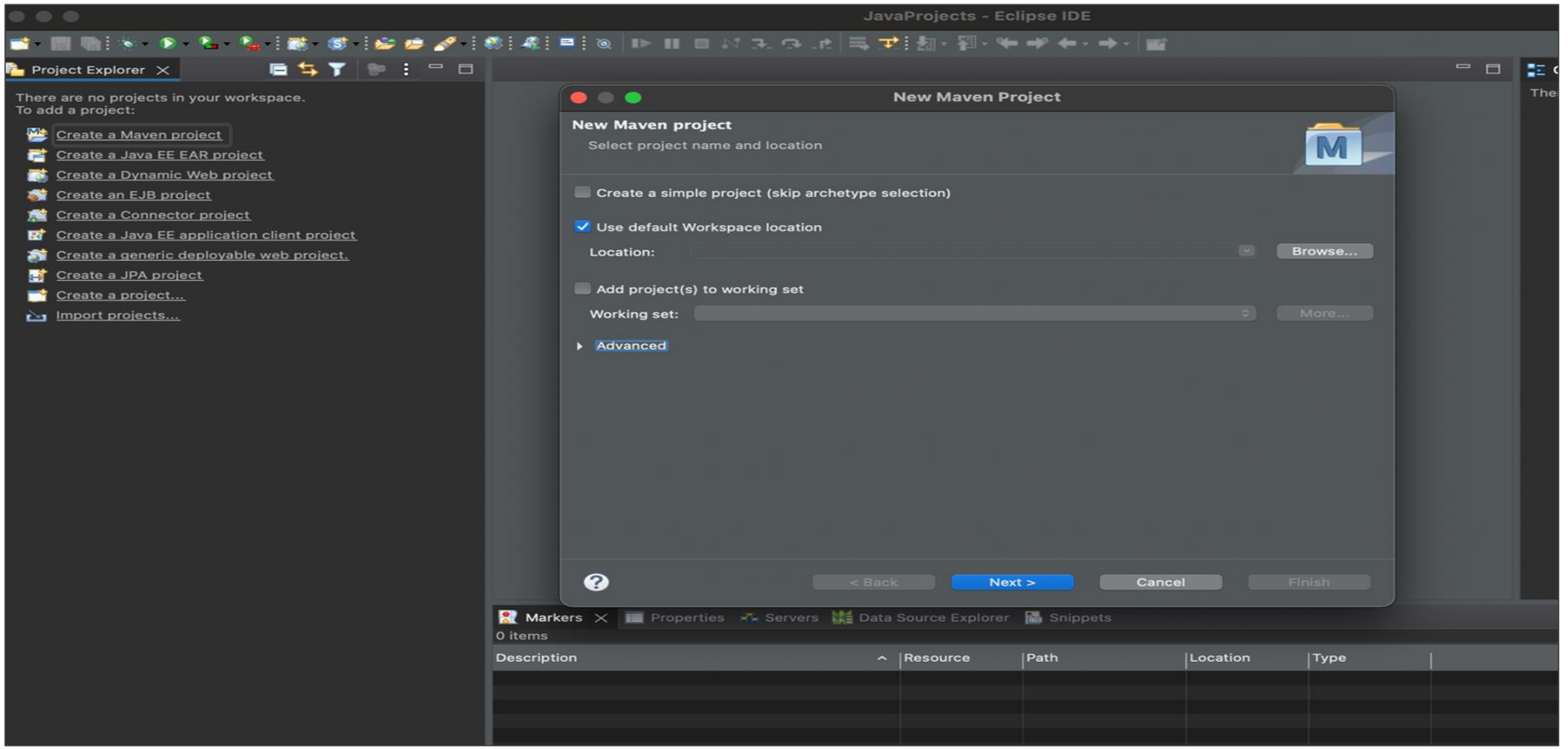
By the end of this lesson, you will be able to:

- Create Maven Project
- Develop POJO Classes for admin dashboard and end user web app
- Configure the dependencies
- Build and run the project

Create a Maven Project

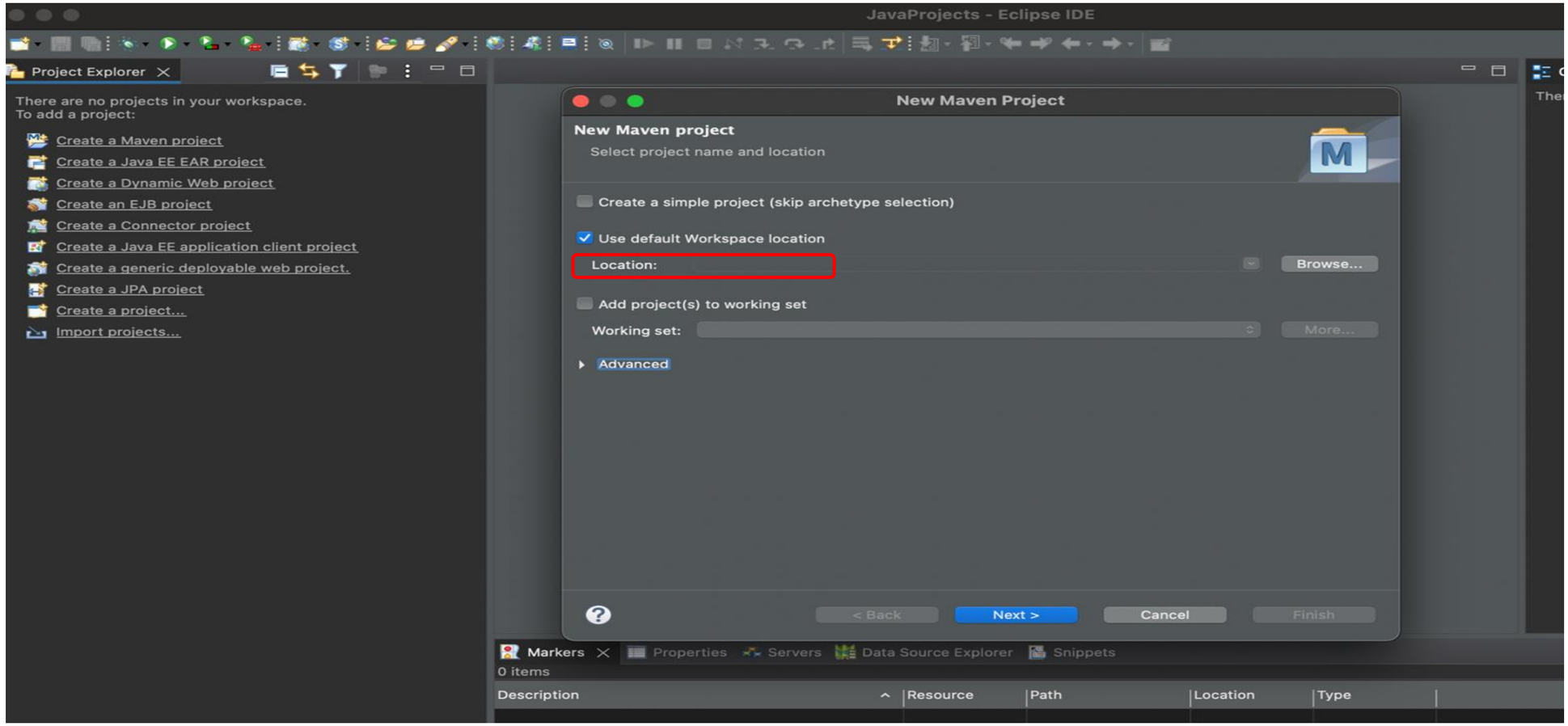
Create Maven Project

Open the Eclipse EE IDE and click to create a new Maven Project or Go to File > New and Select Maven Project.



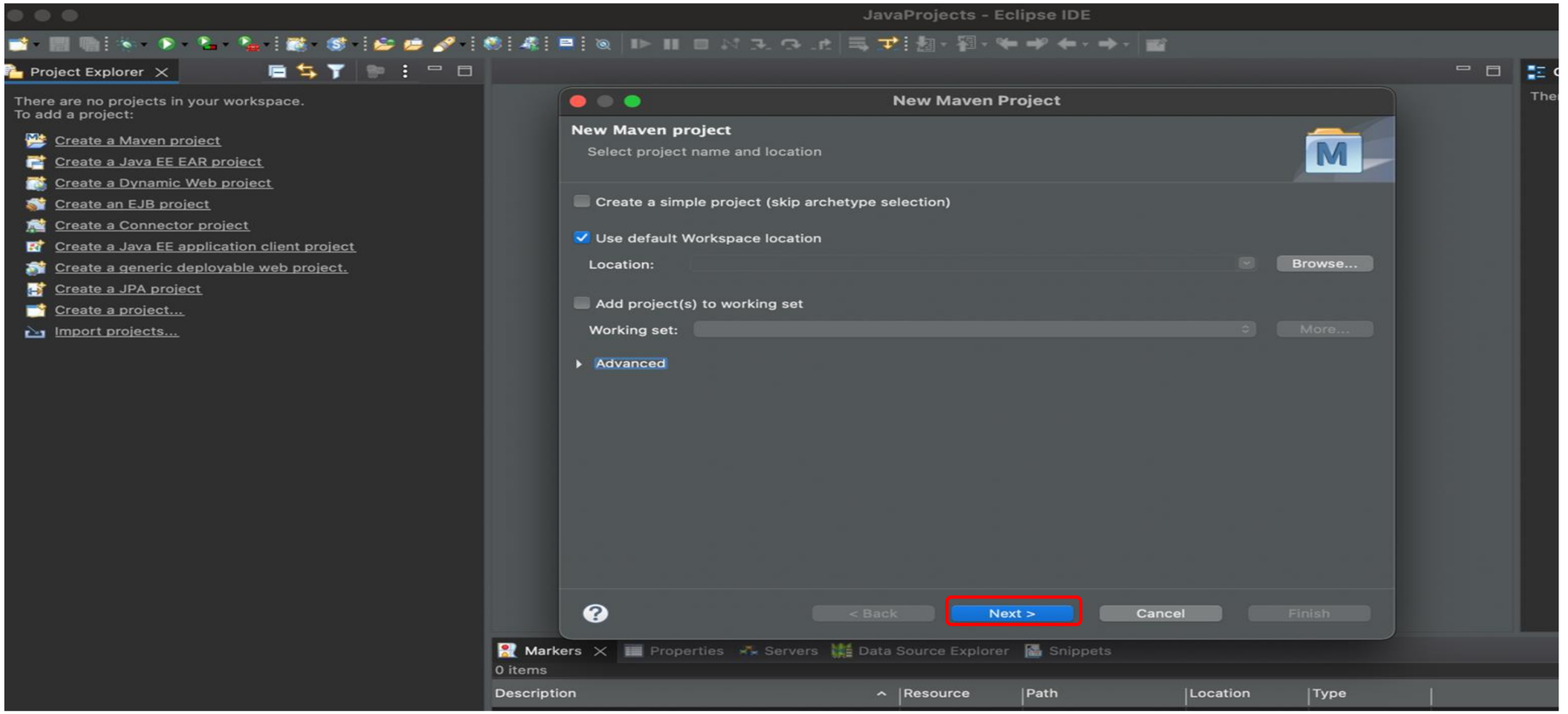
Create Maven Project

By default, workspace location will be selected and can be edited.



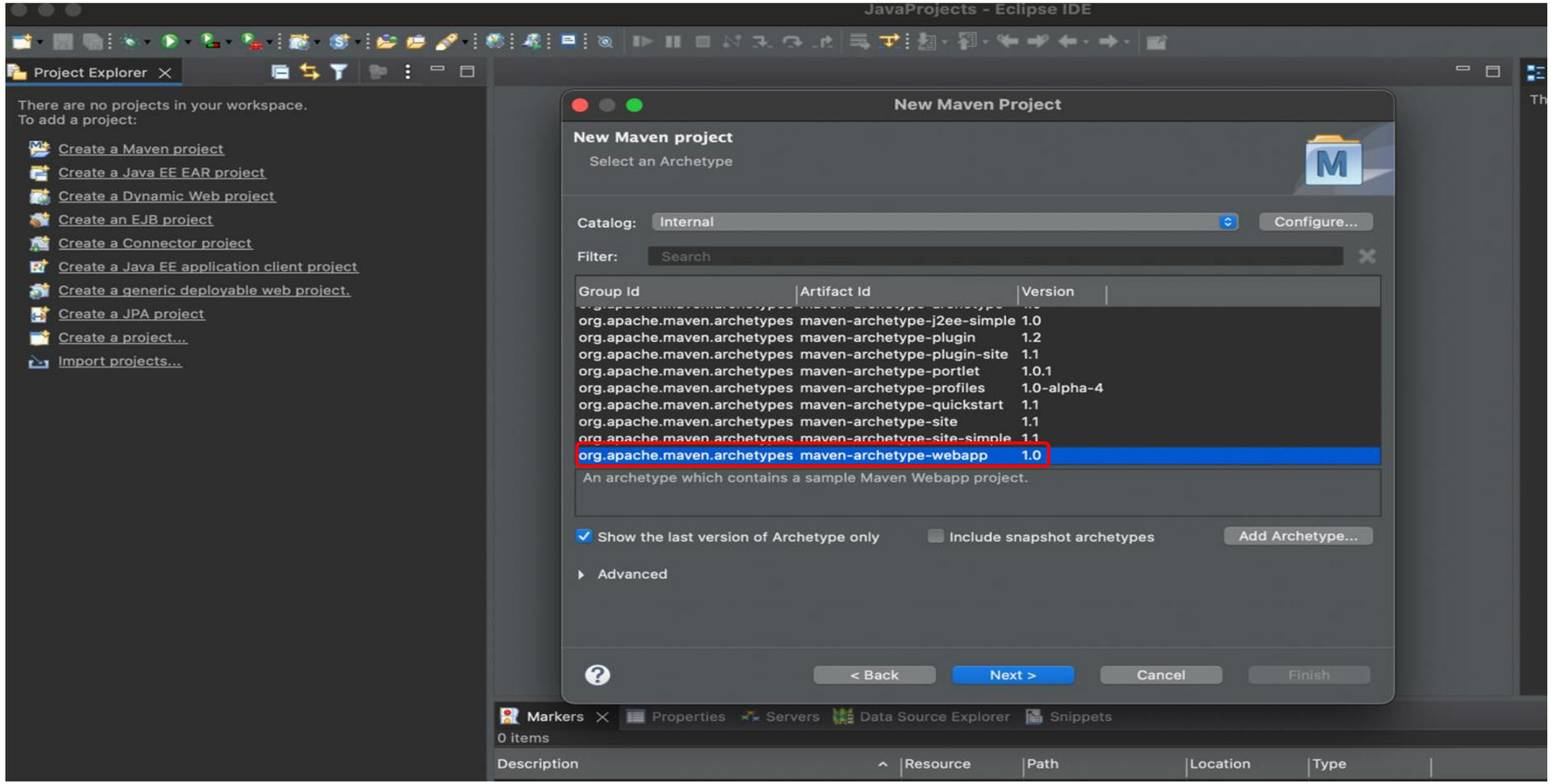
Create Maven Project

Then, click on next.



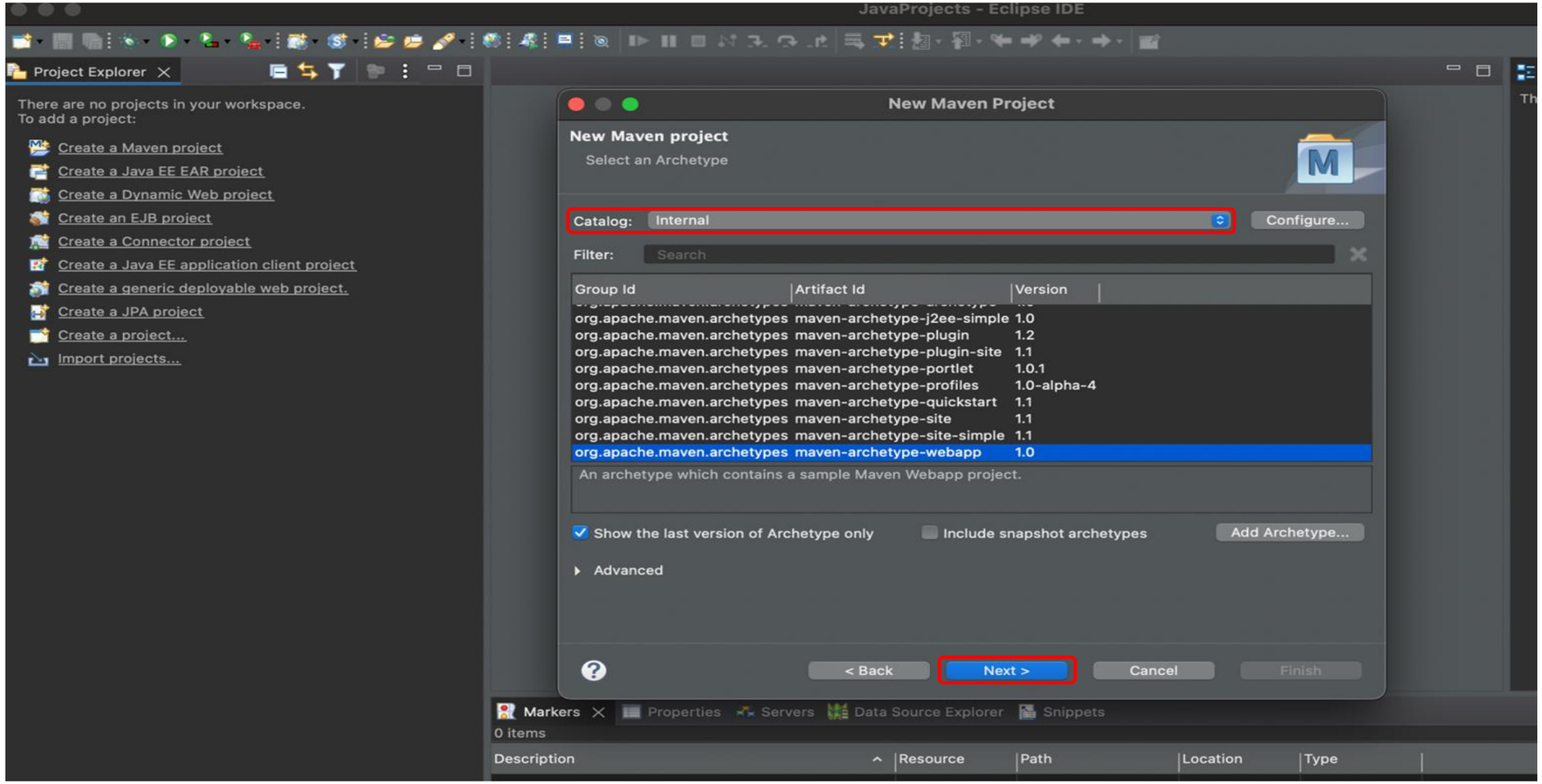
Archetype for Maven Project

Select the project archetype as web app.



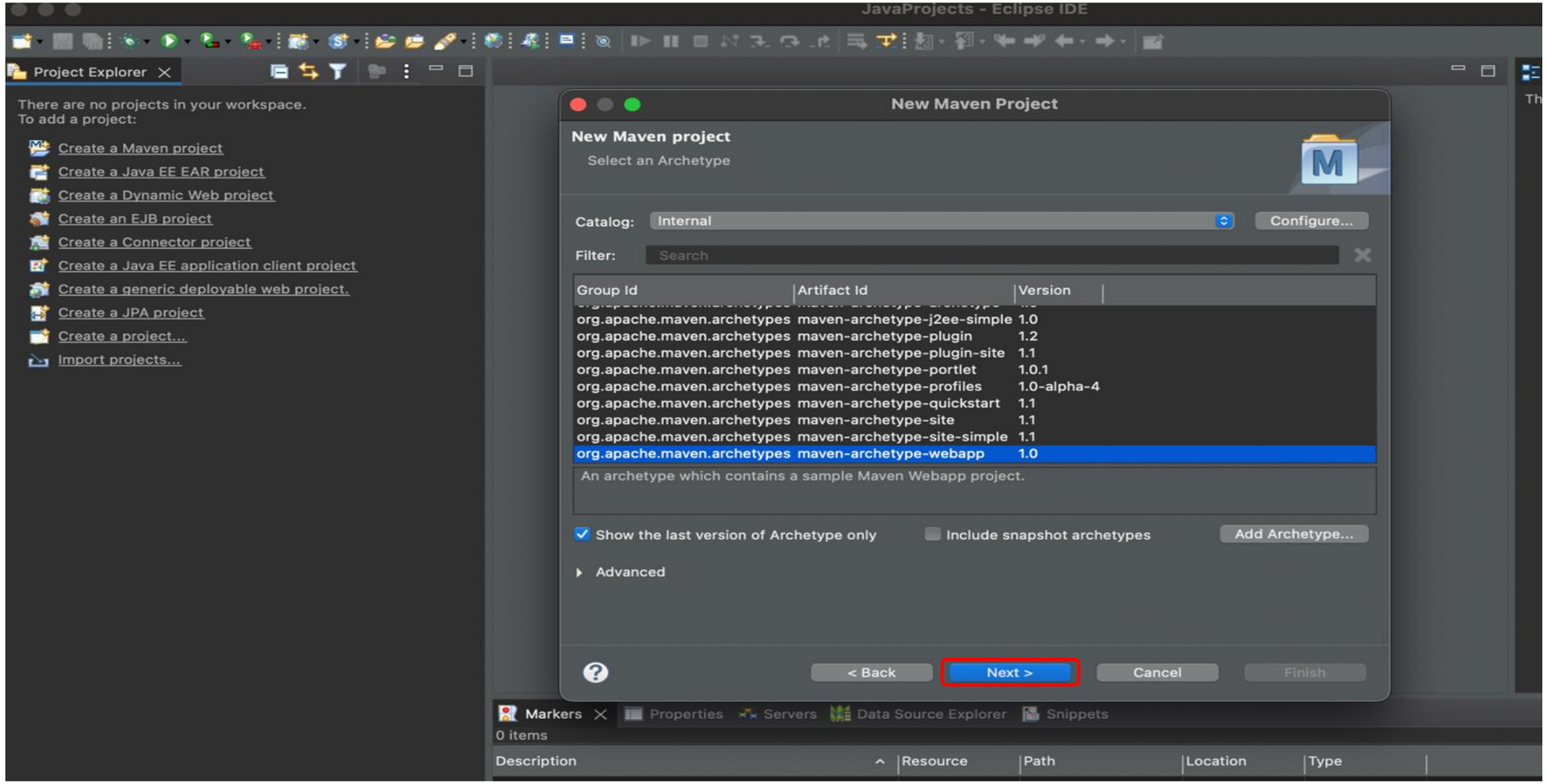
Archetype for Maven Project

Select catalog as internal to get the archetype.



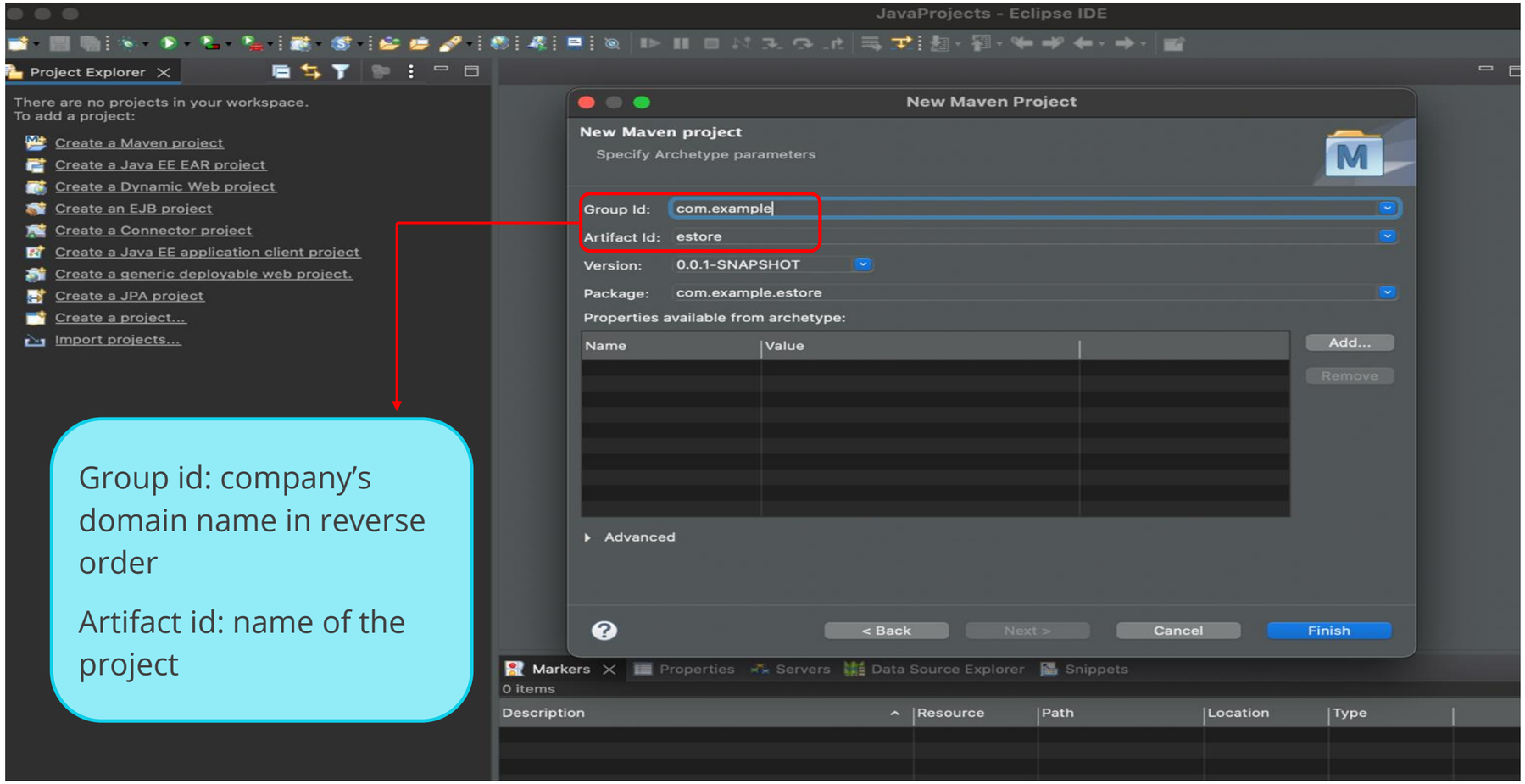
Archetype for Maven Project

Click next to proceed further.



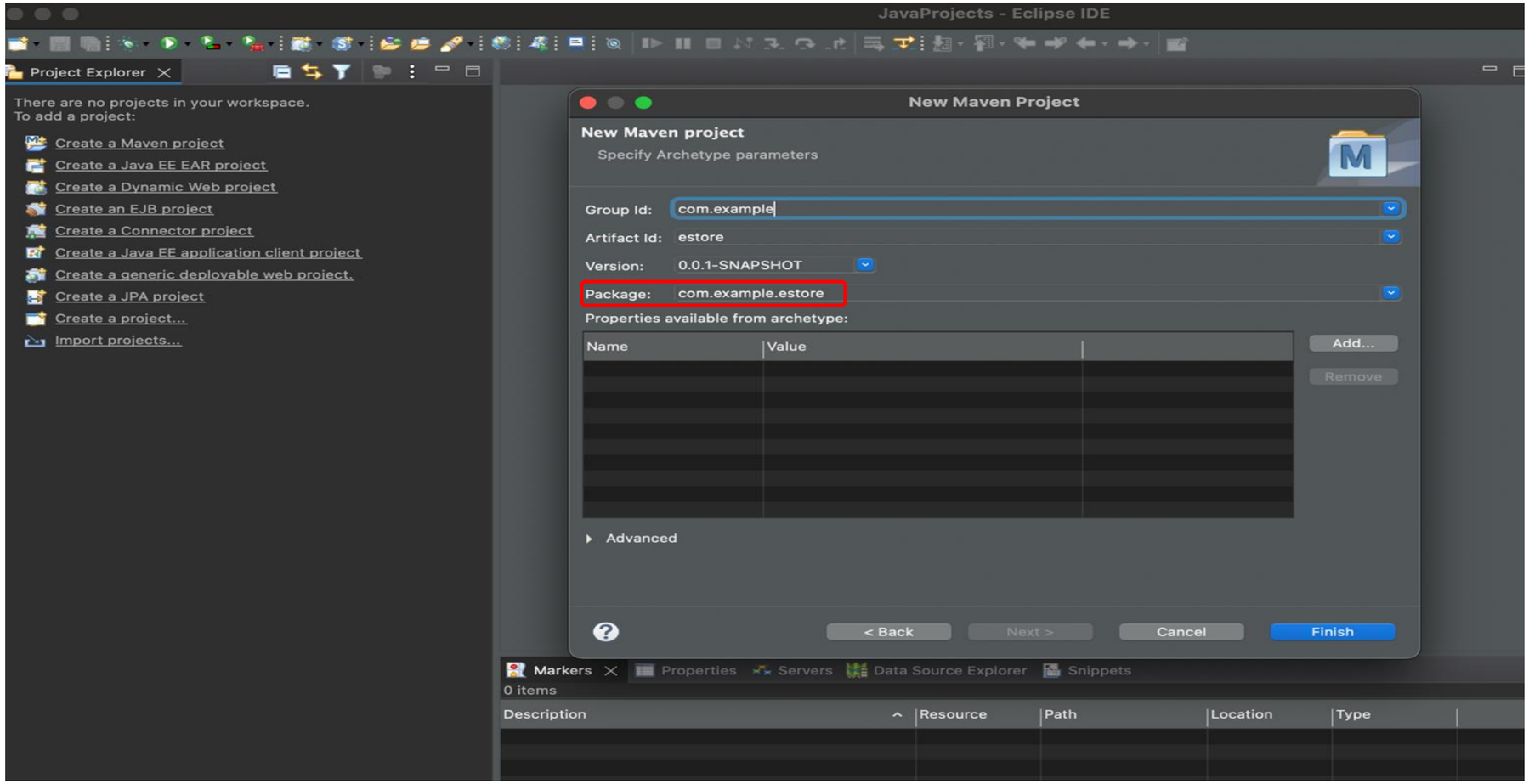
Details for Maven Project

Fill in details for the project.



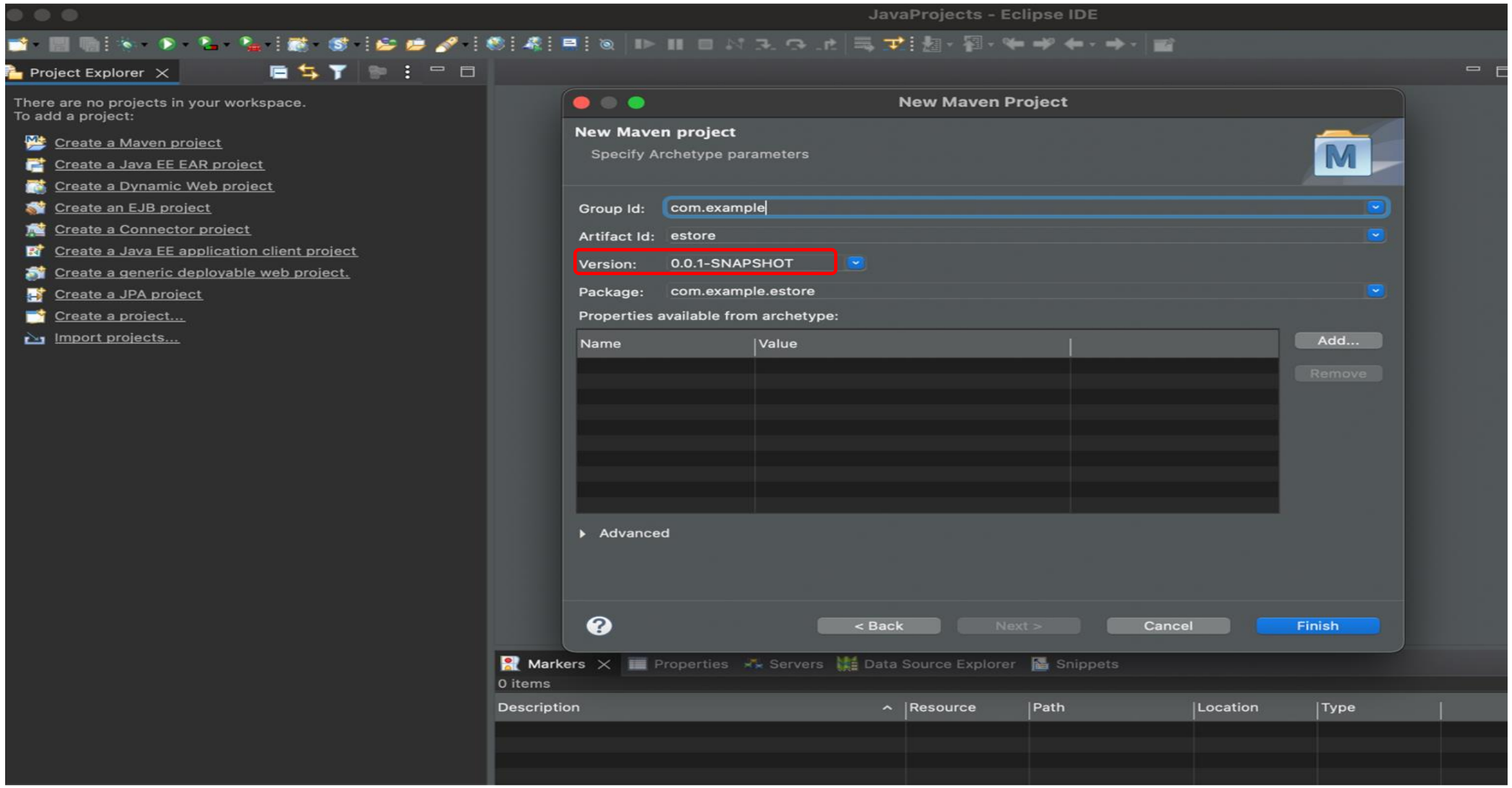
Details for Maven Project

The package name will be automatically picked up as a combination of group id and artifact id.



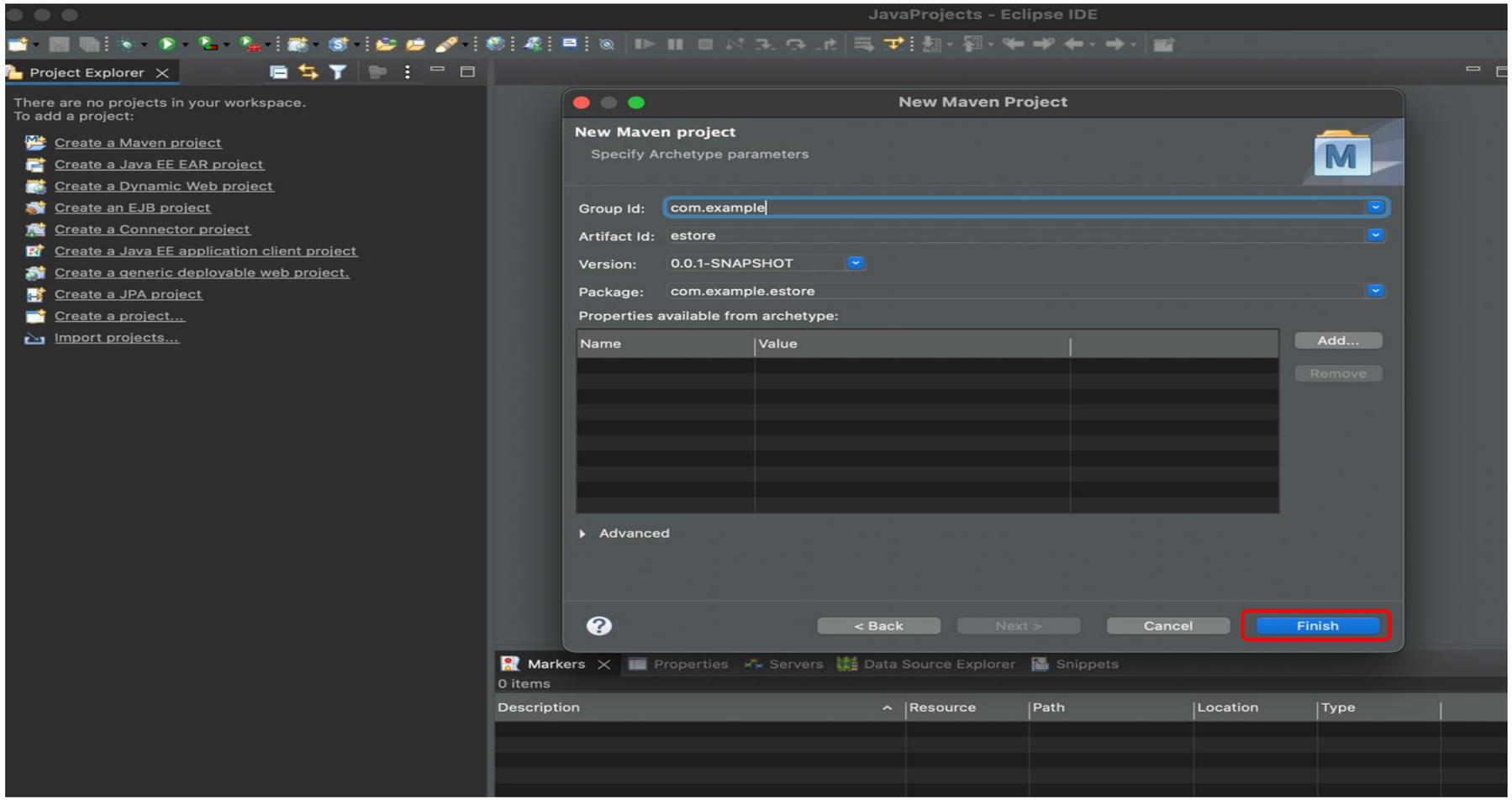
Details for Maven Project

Version specifies the initial version of the project that is editable.



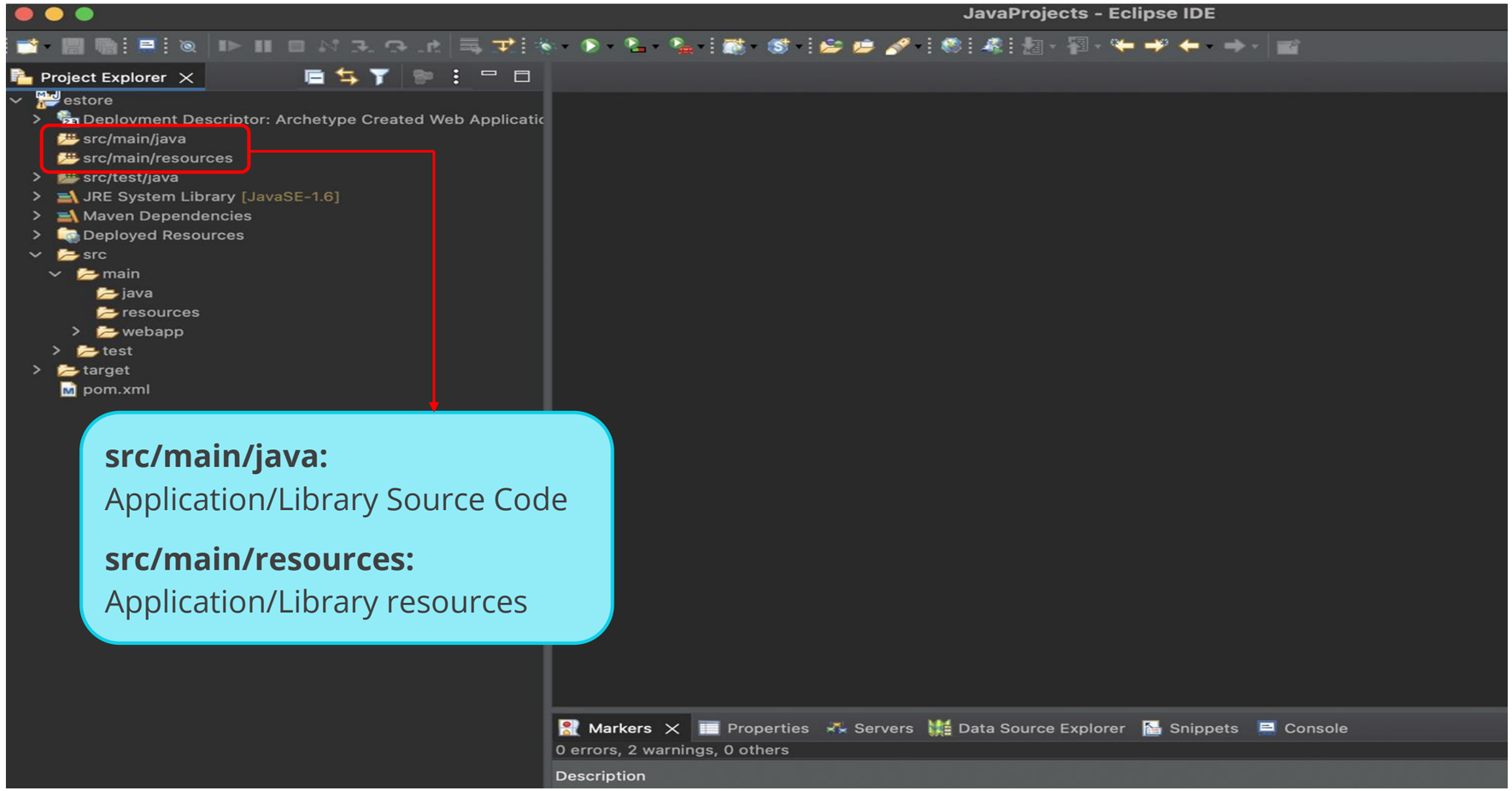
Details for Maven Project

Finally, click on Finish.



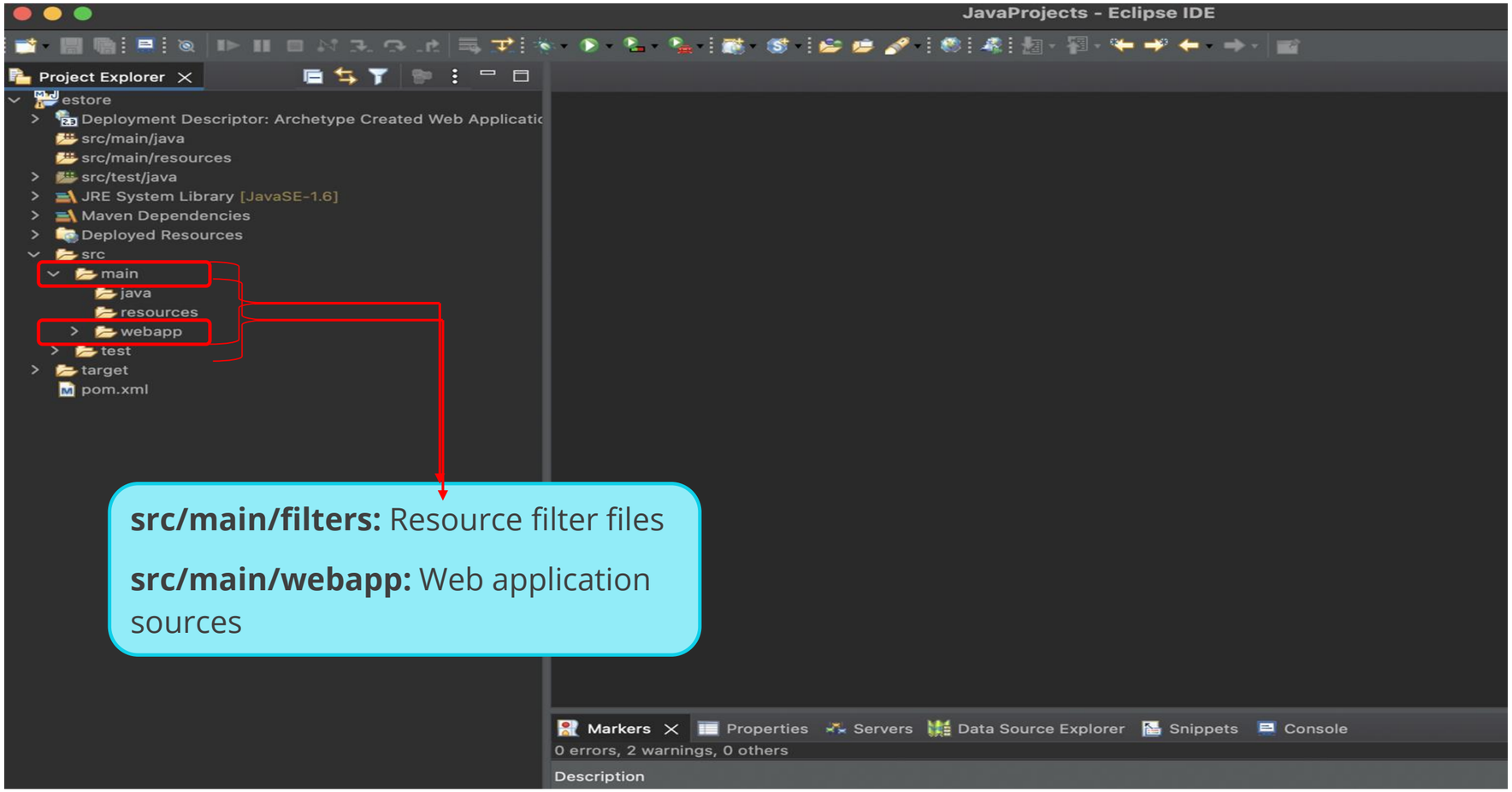
Maven Project Structure

The Maven Project Structure contains various files and directories such as:



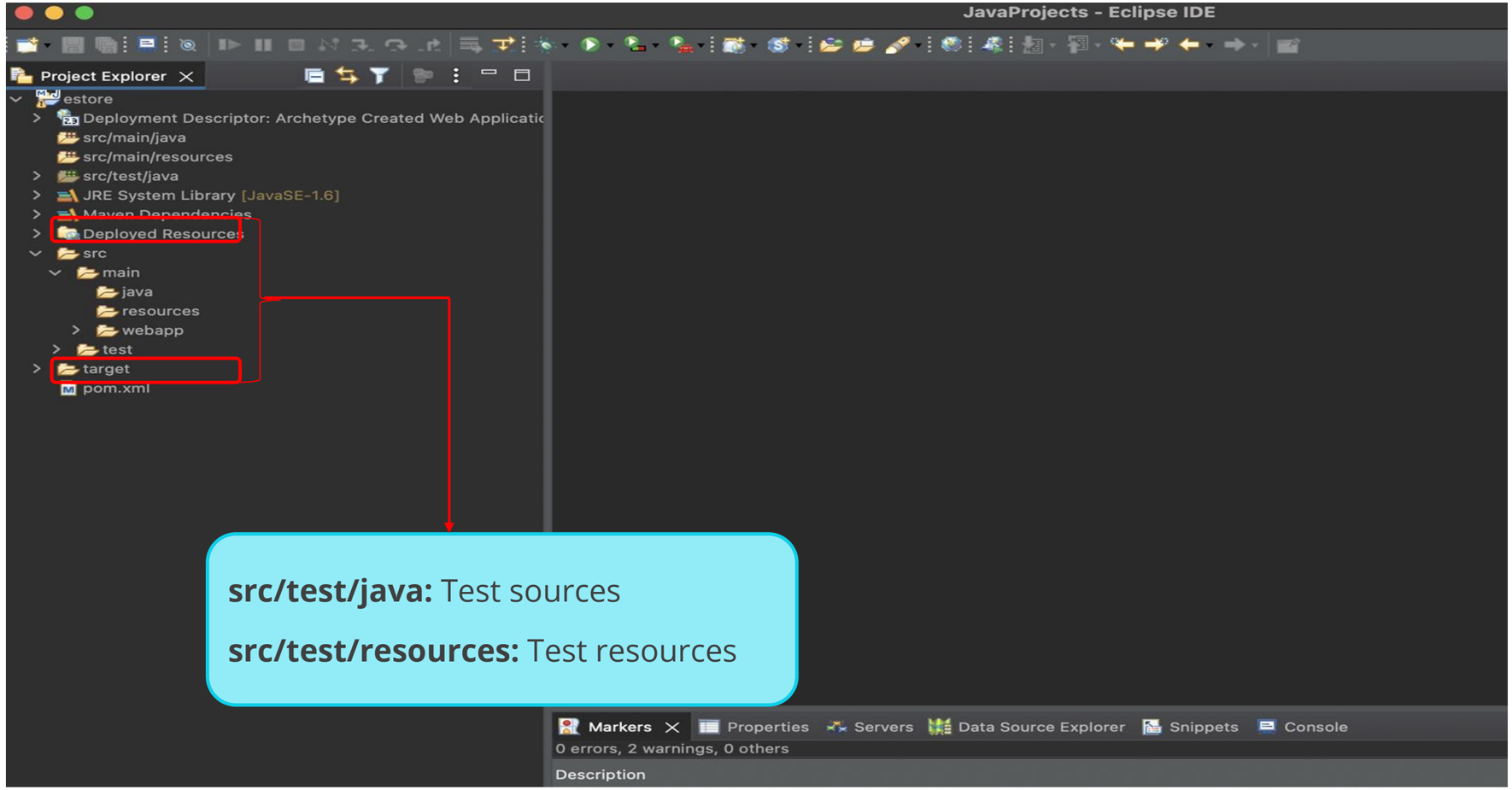
Maven Project Structure

The Maven Project Structure contains various files and directories such as:



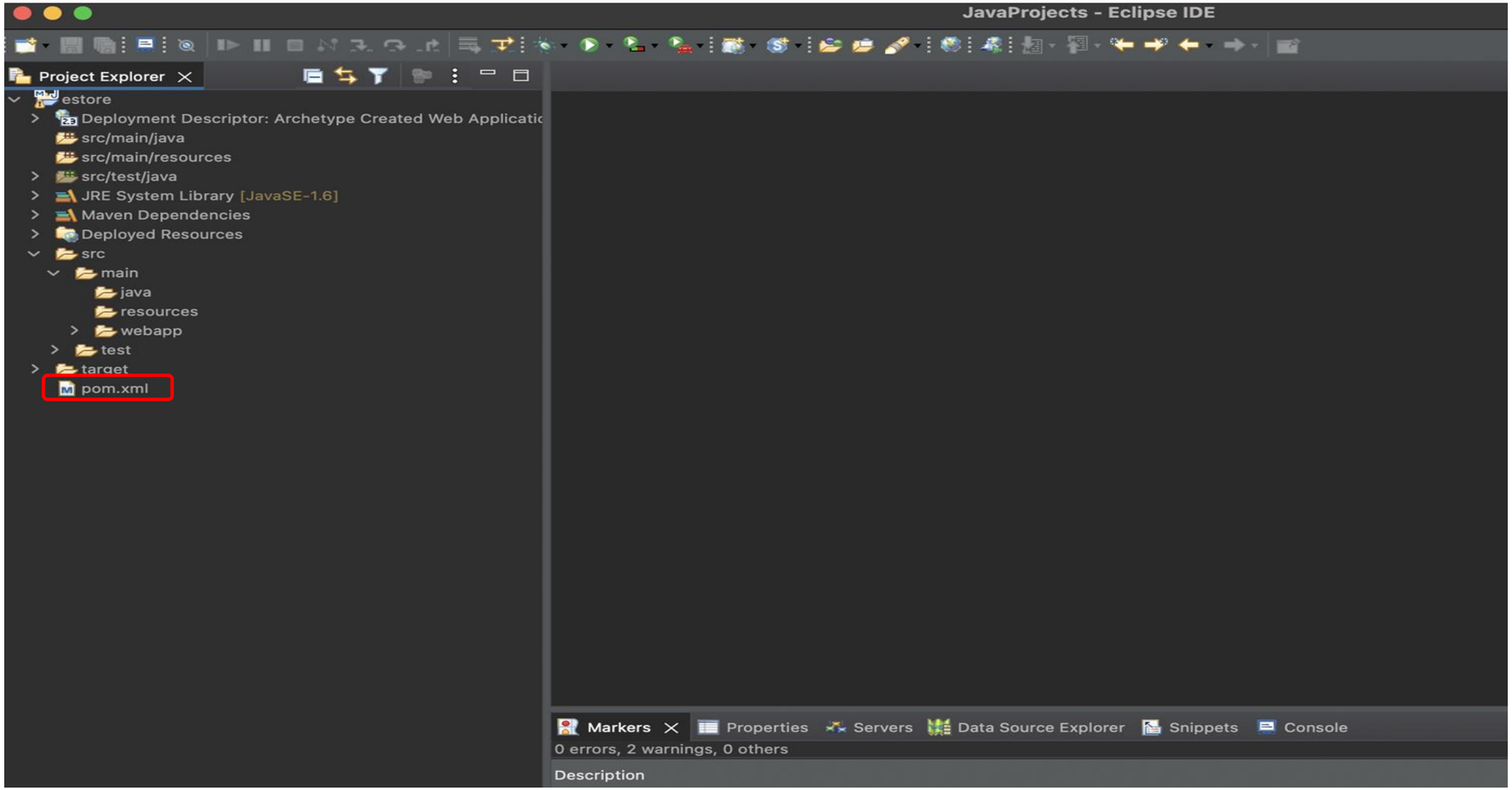
Maven Project Structure

The Maven Project Structure contains various files and directories such as:



Maven Project Structure

pom.xml: A Project Object Model or POM is the fundamental unit of work in Maven, which is an xml file.



Maven Project Structure

The minimum requirement for a POM are the following:

01 Project root

02 Model version – Should be 4.0.0.

03 Group id – Id of the projects group

04 Artifact id – Id of the artifact (project)

05 Version – Artifact version under the specified group

06 Dependencies – List of external dependency to be added for the project in xml.

Configure pom.xml File

pom.xml

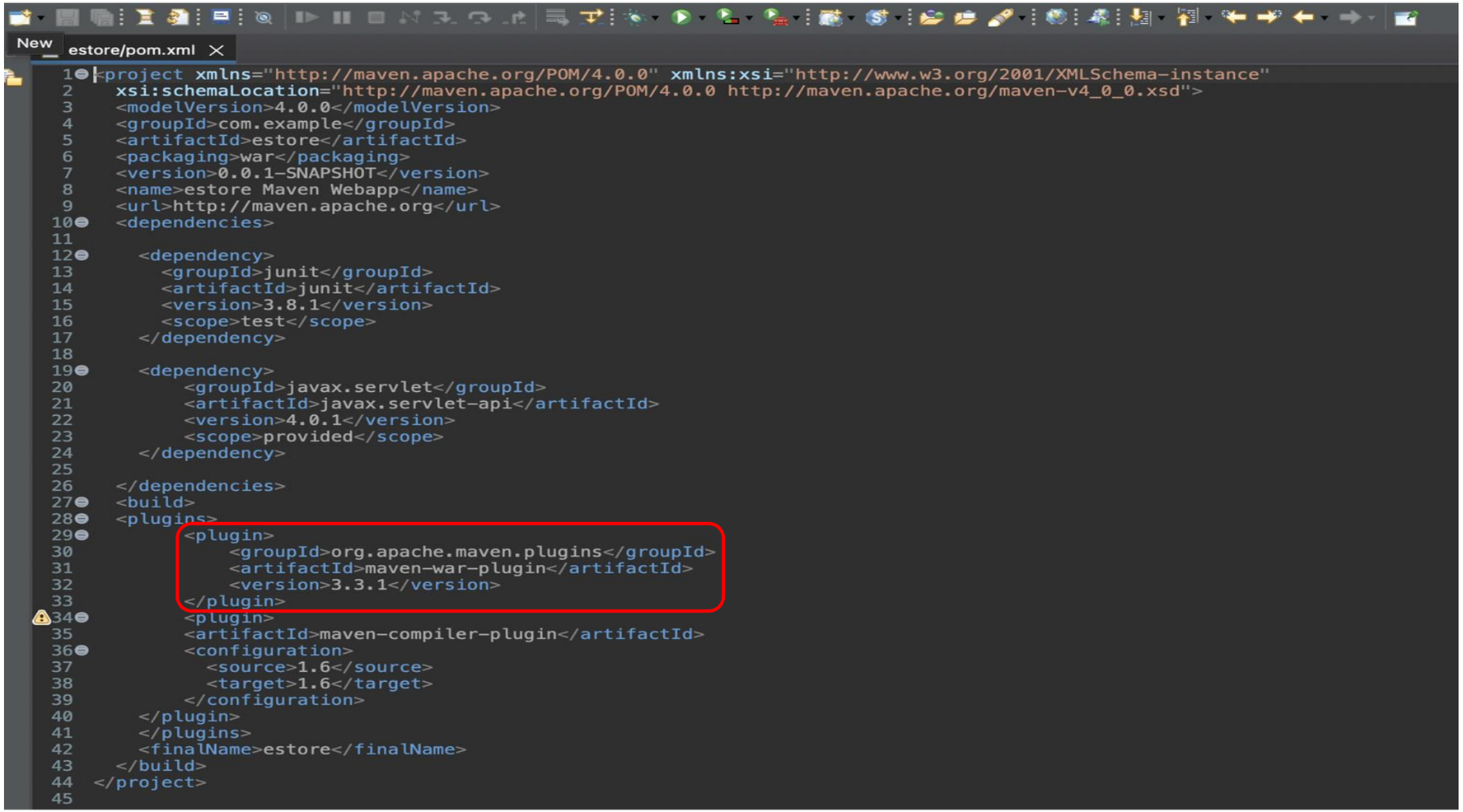
Add dependency for Servlet.

New estore/pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>estore</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>estore Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>

    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>4.0.1</version>
        <scope>provided</scope>
    </dependency>

  </dependencies>
  <build>
  <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>3.3.1</version>
        </plugin>
        <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
    </plugin>
    </plugins>
    <finalName>estore</finalName>
  </build>
</project>
```

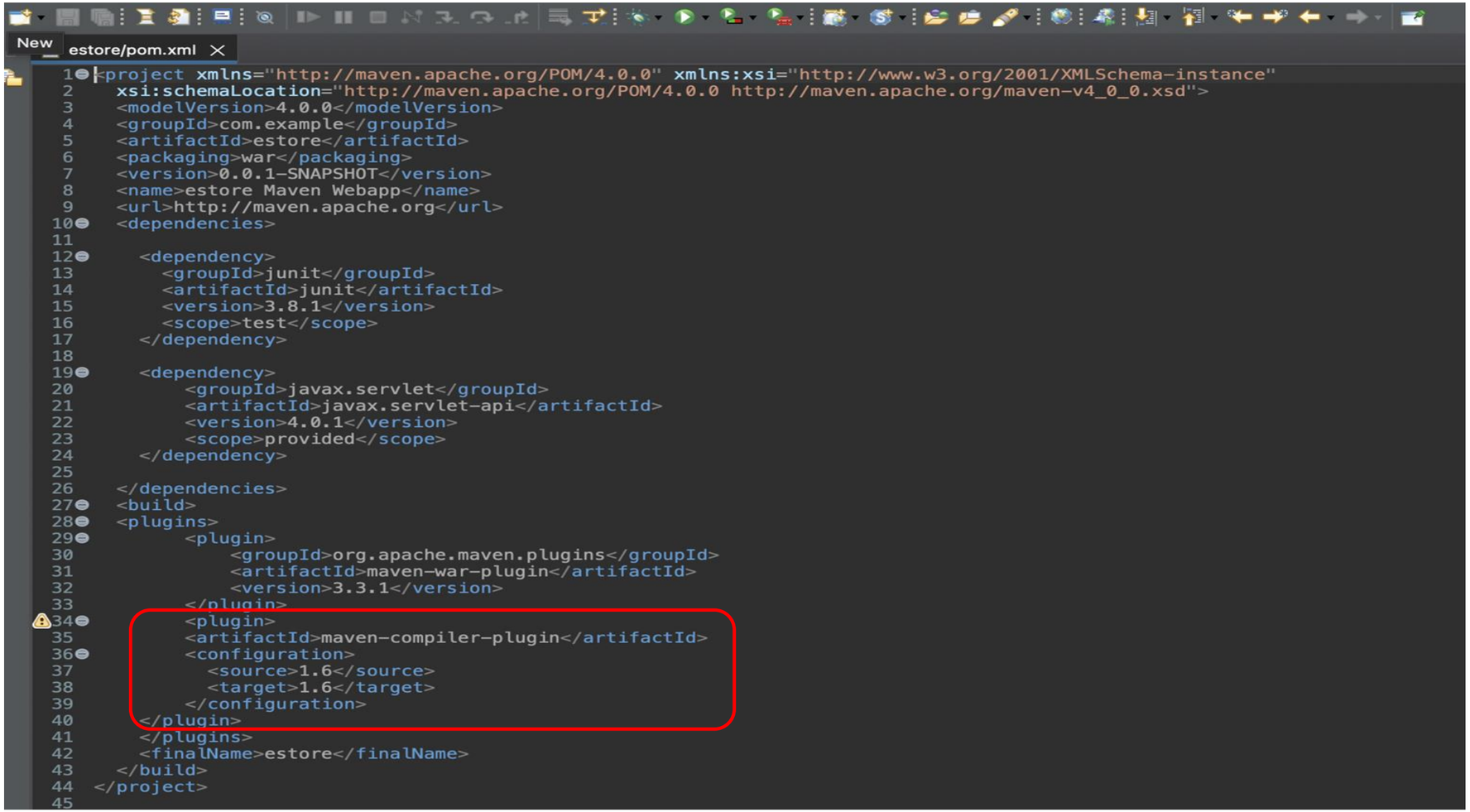
pom.xml

Add Plugin for Maven war.



pom.xml

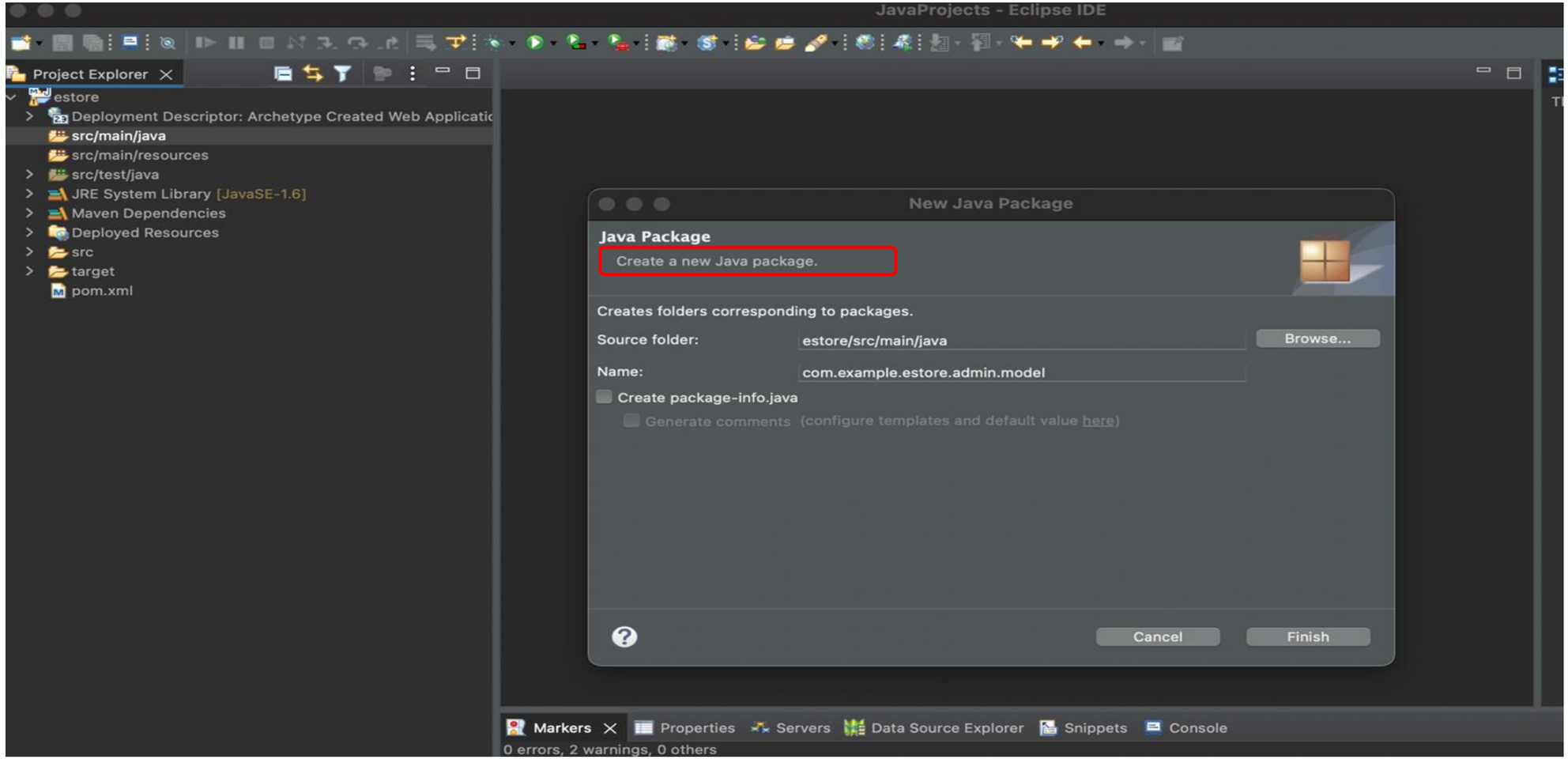
Add Plugin for Maven compiler.



Create Model for Admin Dashboard

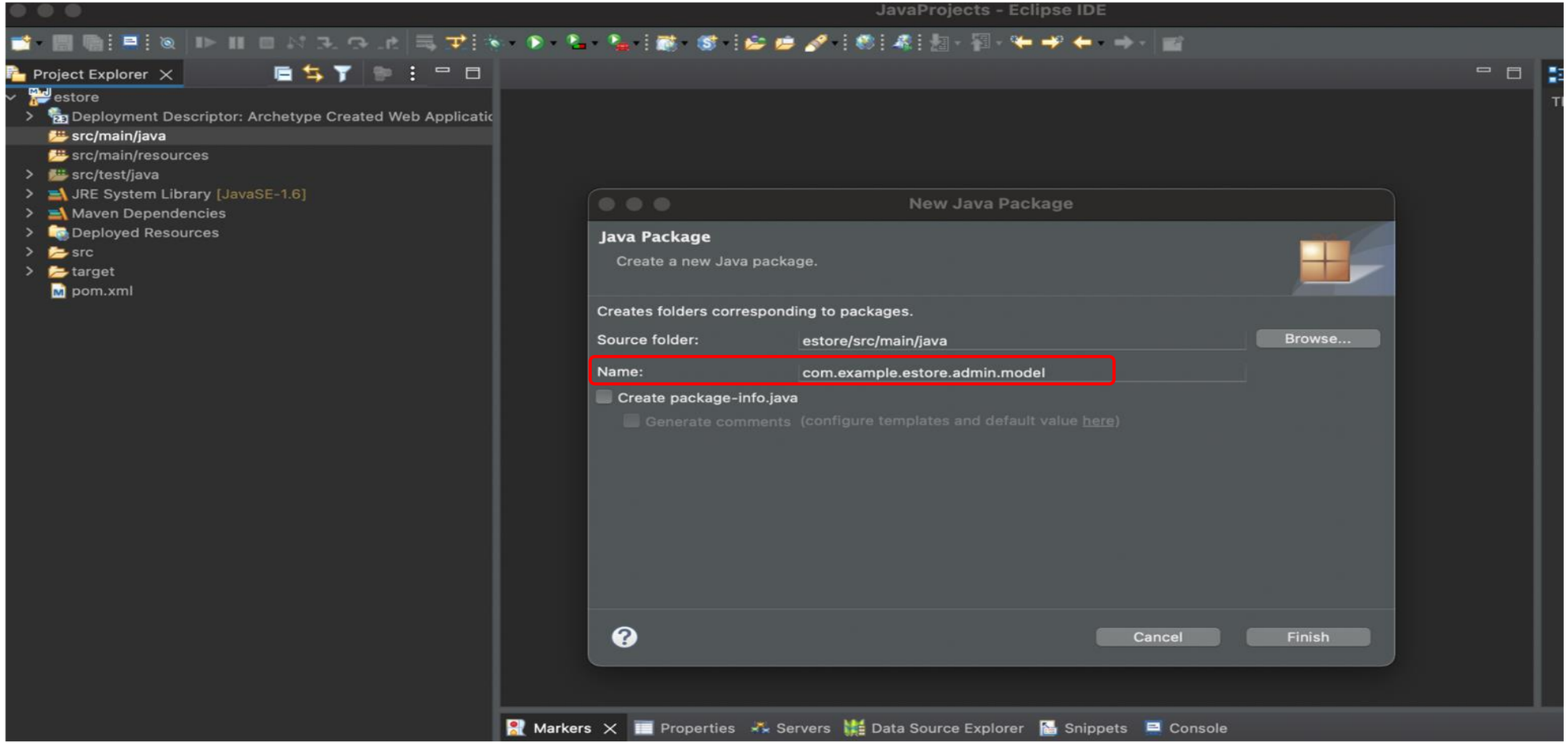
Create Package for Model

Right-click on project structure and create a new package for model.



Create Package for Model

Create a package by named **com.example.estore.admin.model** and create POJO classes for the admin e-store.



POJO Class Structure

POJO Class Structure

POJO is a plain old java object, which contains attributes for an object and along with it, check constructors (default and parameterized), a toString method.

Attributes are private or needs to be accessed individually you can create getter and setters inside the class.

POJO Class Structure

Review the structure of the POJO class.

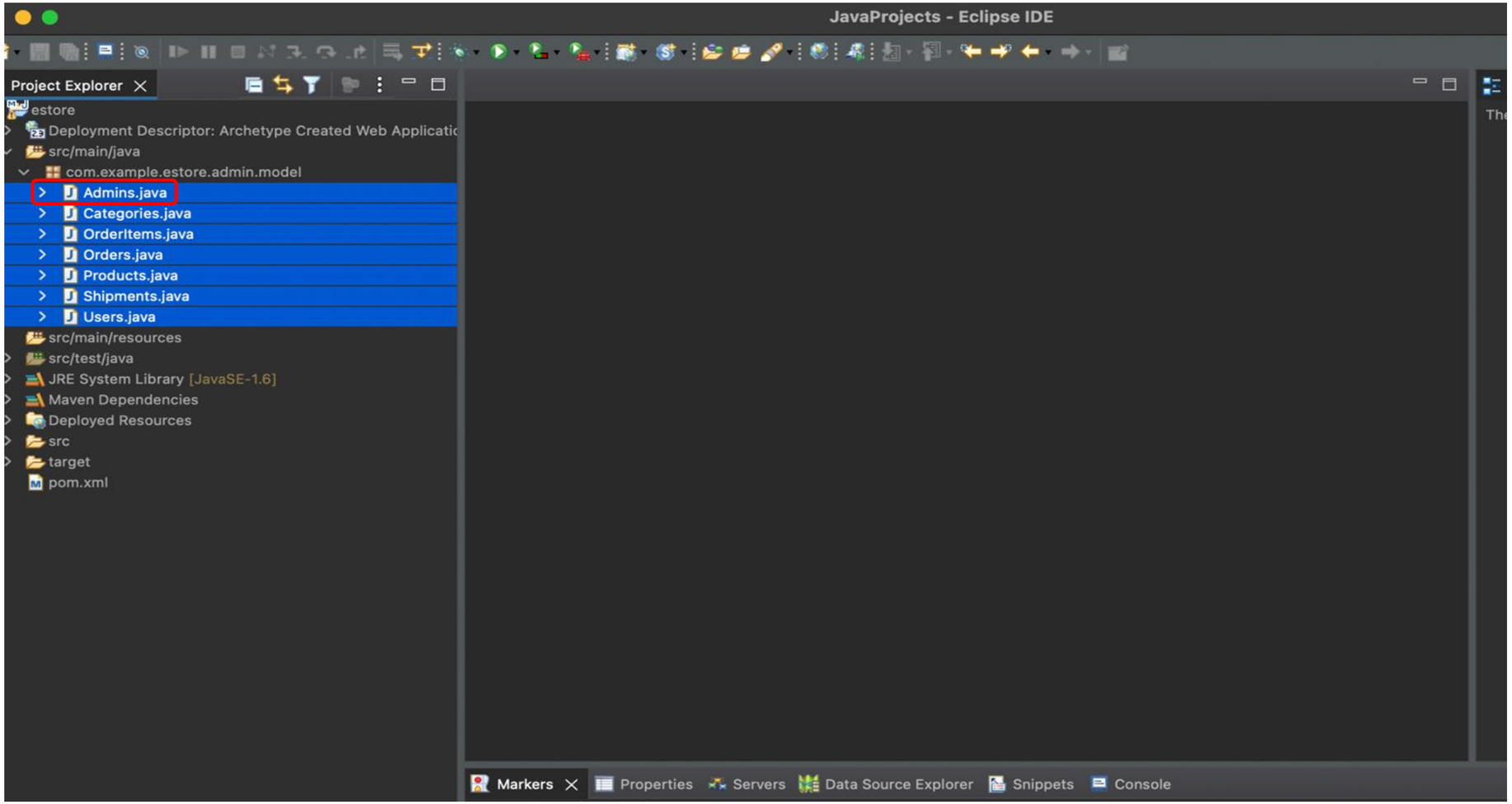
```
public class Users {

      String street;
      String city;
      String state;
      String country;
      Integer pincode;

      public Users() {
      }
```

POJO Class Structure

```
	public Users(String street, String city, String state, String
country, Integer pincode) {
		this.street = street;
		this.city = city;
		this.state = state;
		this.country = country;
		this.pincode = pincode;
	}
	@Override
	public String toString() {
		return "Users [street=" + street + ", city=" + city + ",
state=" + state + ", country=" + country + ", pincode=‟ + pincode +
"]";
	}

}
```

Create POJO Classes for the Admin Dashboard

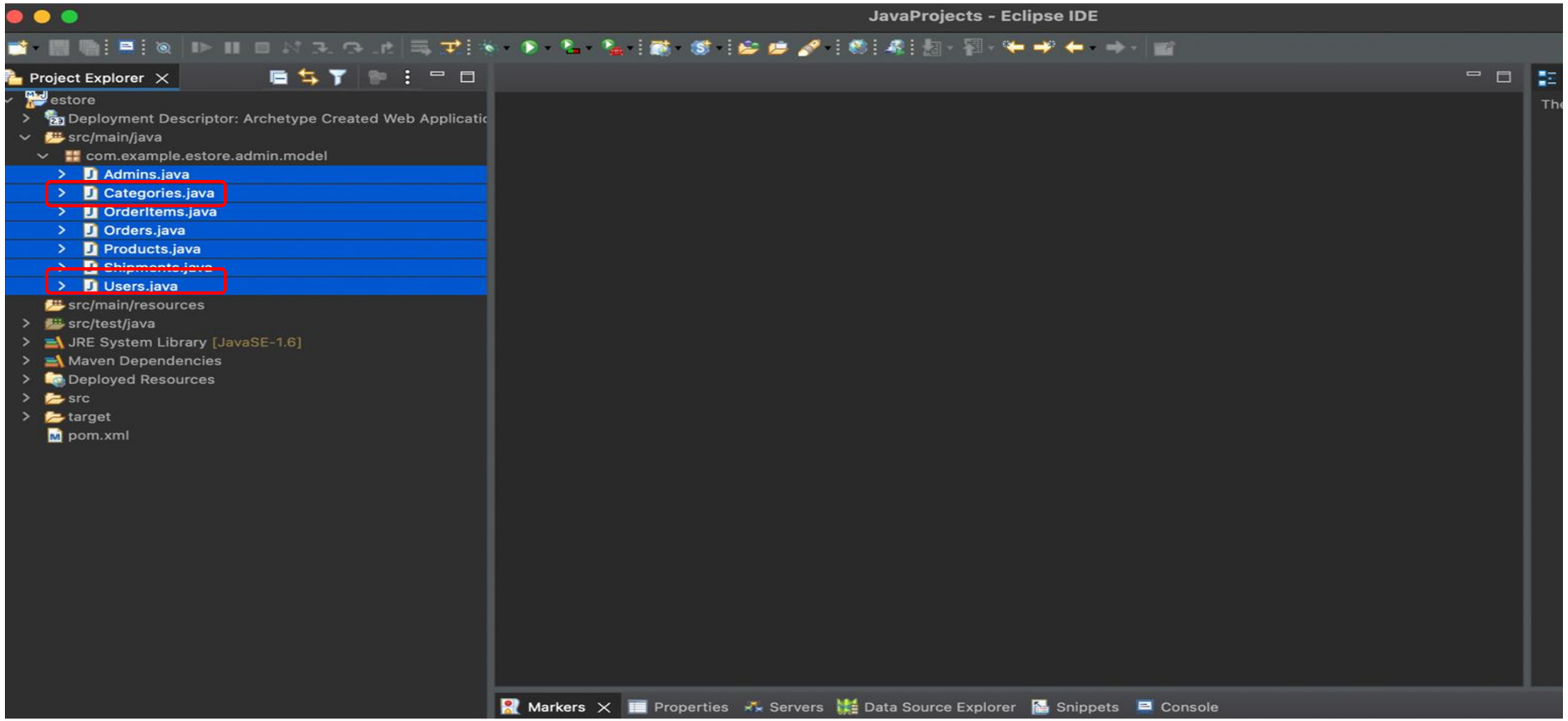
Admins.java : For the Admin Authentication i.e. the Admin User



Create POJO Classes for the Admin Dashboard

Users.java : For the users registered on the Web Backend

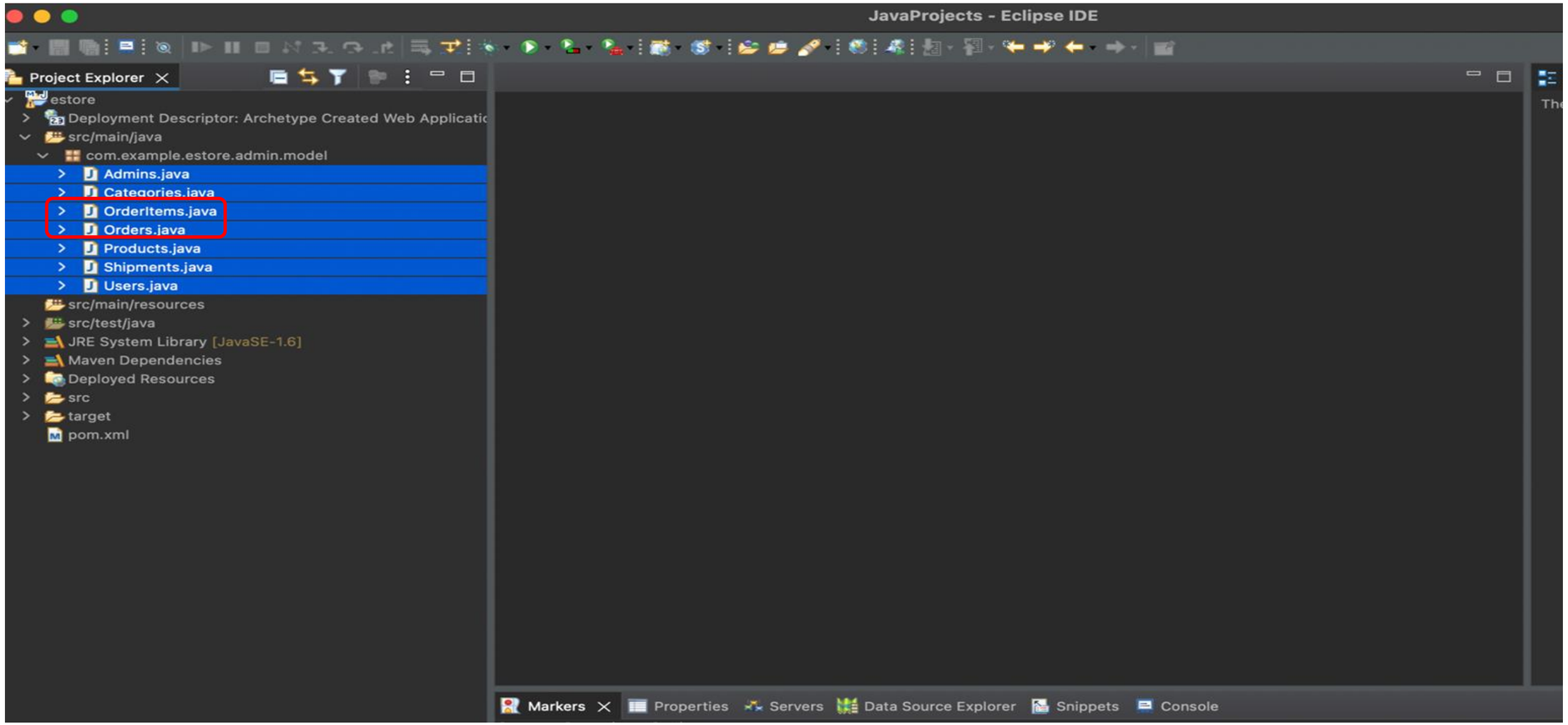
Categories.java : For the Product Categories



Create POJO Classes for the Admin Dashboard

Orders.java : For the orders placed by the Users

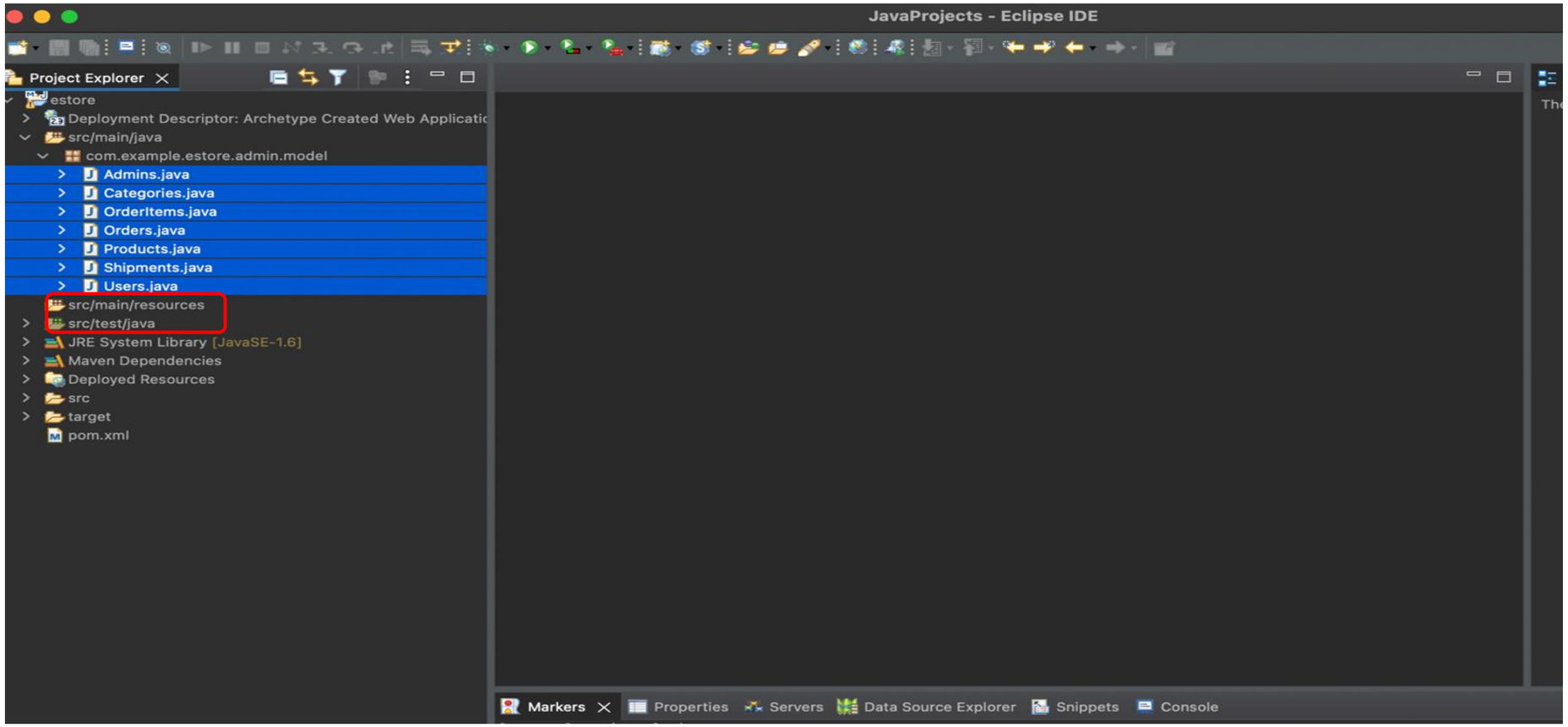
Orderitems.Java : For the products available in an order placed by users



Create POJO Classes for the Admin Dashboard

Products.Java : For the products along with category to be added by admin for end users

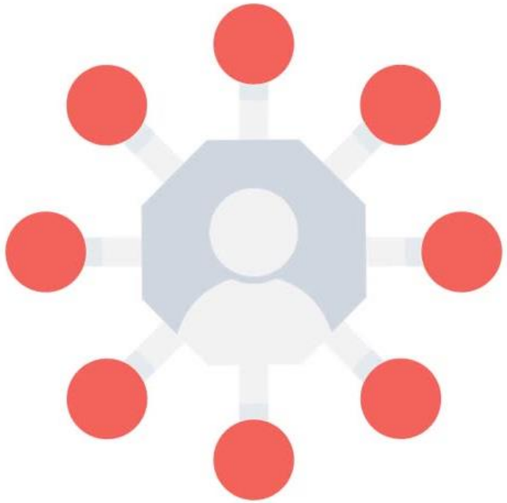
Shipments.Java : For the placed order shipped for a user



Admins.java

Admin Model will have attributes for authentication of the admin users.

```
public class Admins {
      Integer adminId;
      String email;
      String password;
      String fullName;
      Integer loginType;
      Date addedOn;
}
```



Categories.java

Category is used to classify the products.

Categories model will have attributes to define the category of the product.

```
public class Categories {
	Integer categoryId;
	String categoryName;
	String categoryDescription;
	String catgeoryImageUrl;
	Integer active;
	Date addedOn;
}
```

Products.java

E-store web app will display the list of products to the end user.

The admin must add the products in the database from the admin dashboard.

```
public class Products {
      Integer productId;
      String productTitle;
      String productDescription;
      String productCode;
      List<String> images;
      Integer thumnailImage;
      Integer price;
      Date addedOn;
      Integer rating;
}
```

Products.java

Product model will have attributes that defines the product object for the ecommerce store.

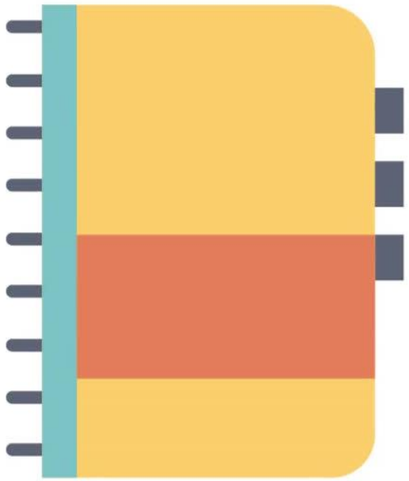
```
public class Products {
      Integer productId;
      String productTitle;
      String productDescription;
      String productCode;
      List<String> images;
      Integer thumnailImage;
      Integer price;
      Date addedOn;
      Integer rating;
}
```

Users.java

To list the users, a user model is used.

Users model will have the details for the user who will register on the end user web app.

```
public class Users {
      String street;
      String city;
      String state;
      String country;
      Integer pincode;
}
```



Orders.java

A user can place an order which must have data associated with the user and the products within it.

Order model will hold the attributes for the order placed by user.

```
public class Orders {
	Integer orderId;
	Date orderDate;
	String orderStatus;
	List<OrderItems> products;
	Integer totalItems;
	Double itemsSubTotal;
	Double shipmentCharges;
	Double totalAmount;
	Integer paymentStatus;
	String paymentStatusTitle;
	Integer paymentMethod;
	String paymentMethodTitle;
	Integer userId;
}
```

OrderItems.java

Order will have multiple products inside it as placed by the user.

OrderItems model will have attributes that define the various products associated to an order.

```
public class OrderItems {
	Integer orderItemId;
	Integer orderId;
	Integer productId;
	String productTitle;
	String productDescription;
	String productCode;
	String productImg;
	String productCategory;
	Integer price;
	Integer quantity;
	Integer totalPrice;
}
```

Shipments.java

The order placed by the user is shipped by the Admin. Shipment Model. holds the information for the order being shipped.

```
public class Shipments {
      Integer shipmentId;
      Integer shipmentStatus;
      String shipmentTitle;
      Date shipmentDate;
      String shipmentMethod;
      String shipmentCompany;
}
```

Cloud Computing

Caltech | Center for Technology & Management Education

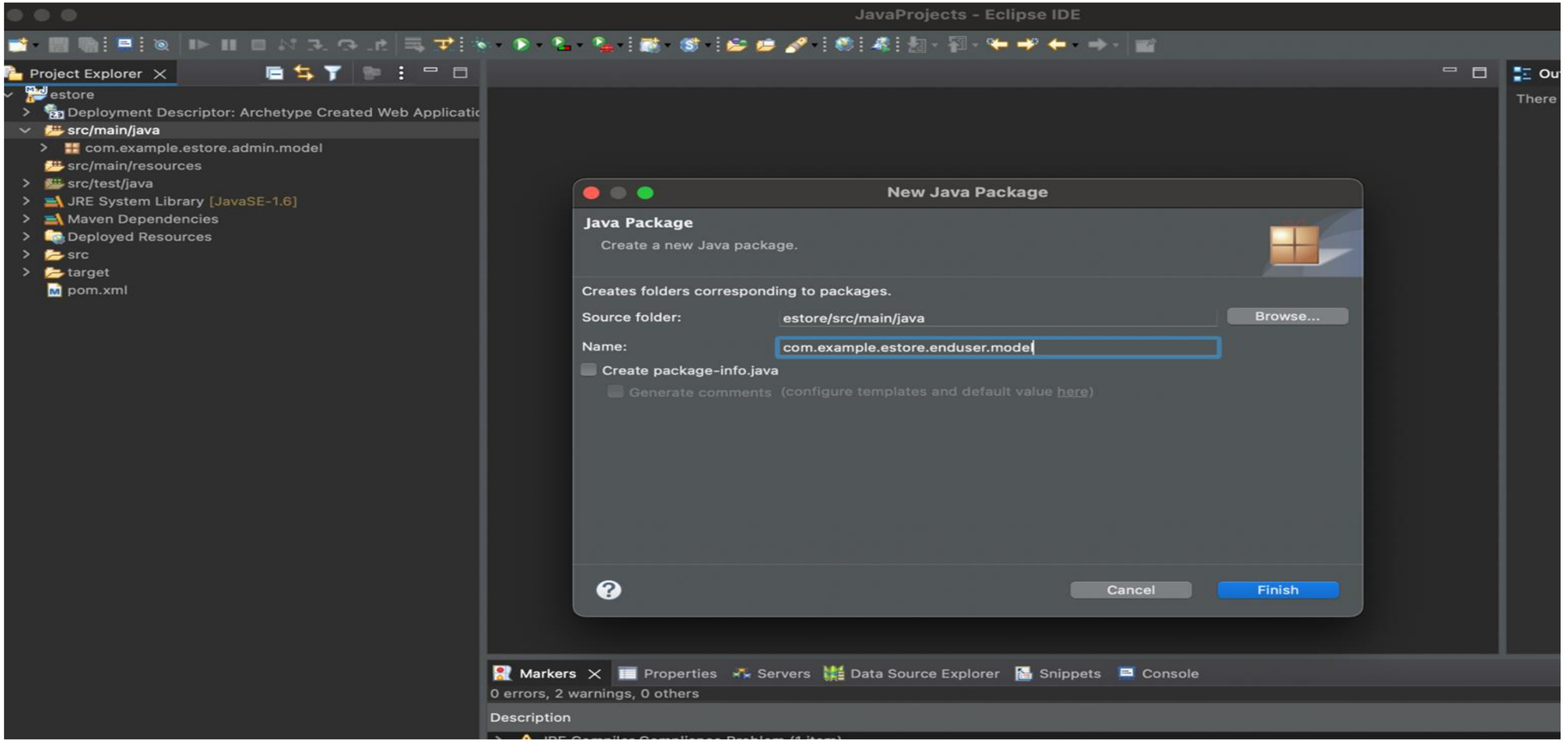
Working on Designing Model for End User Web App

Powerd by simplilearn

Create Model for End User Web App

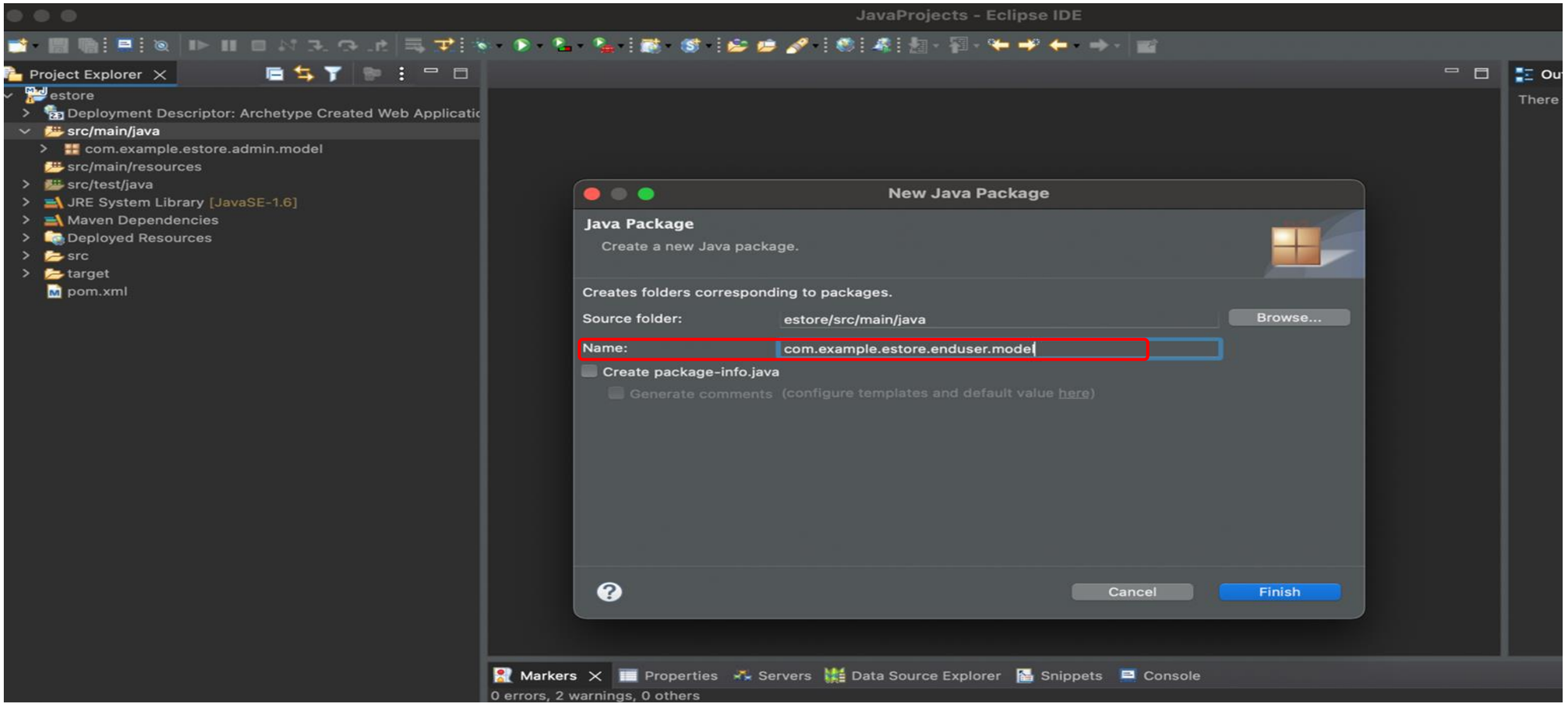
Create Package for Model

Right-click on the project structure and create a new package for the model.



Create Package for Model

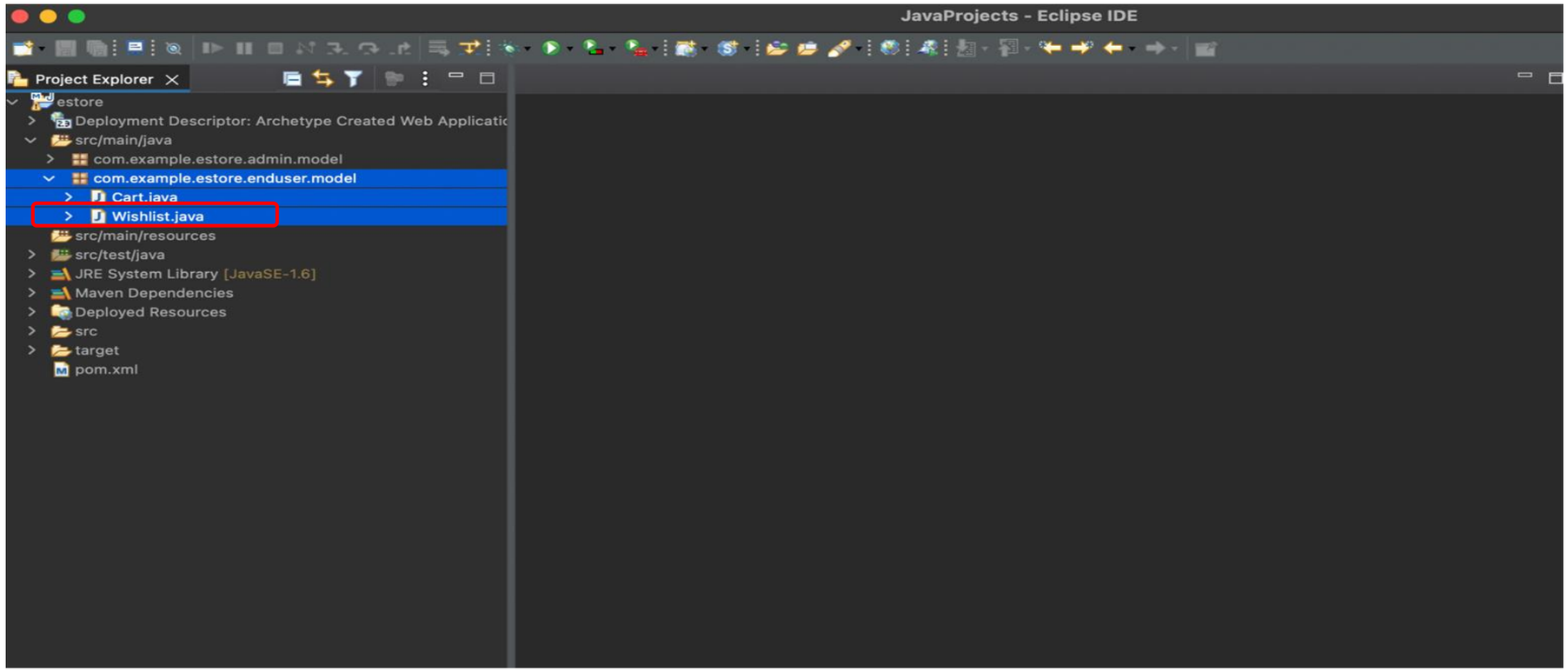
Create a package by name **com.example.estore.enduser.model** and create POJO classes for the end user web app of e-store.



Create POJO Classes for the Admin Dashboard

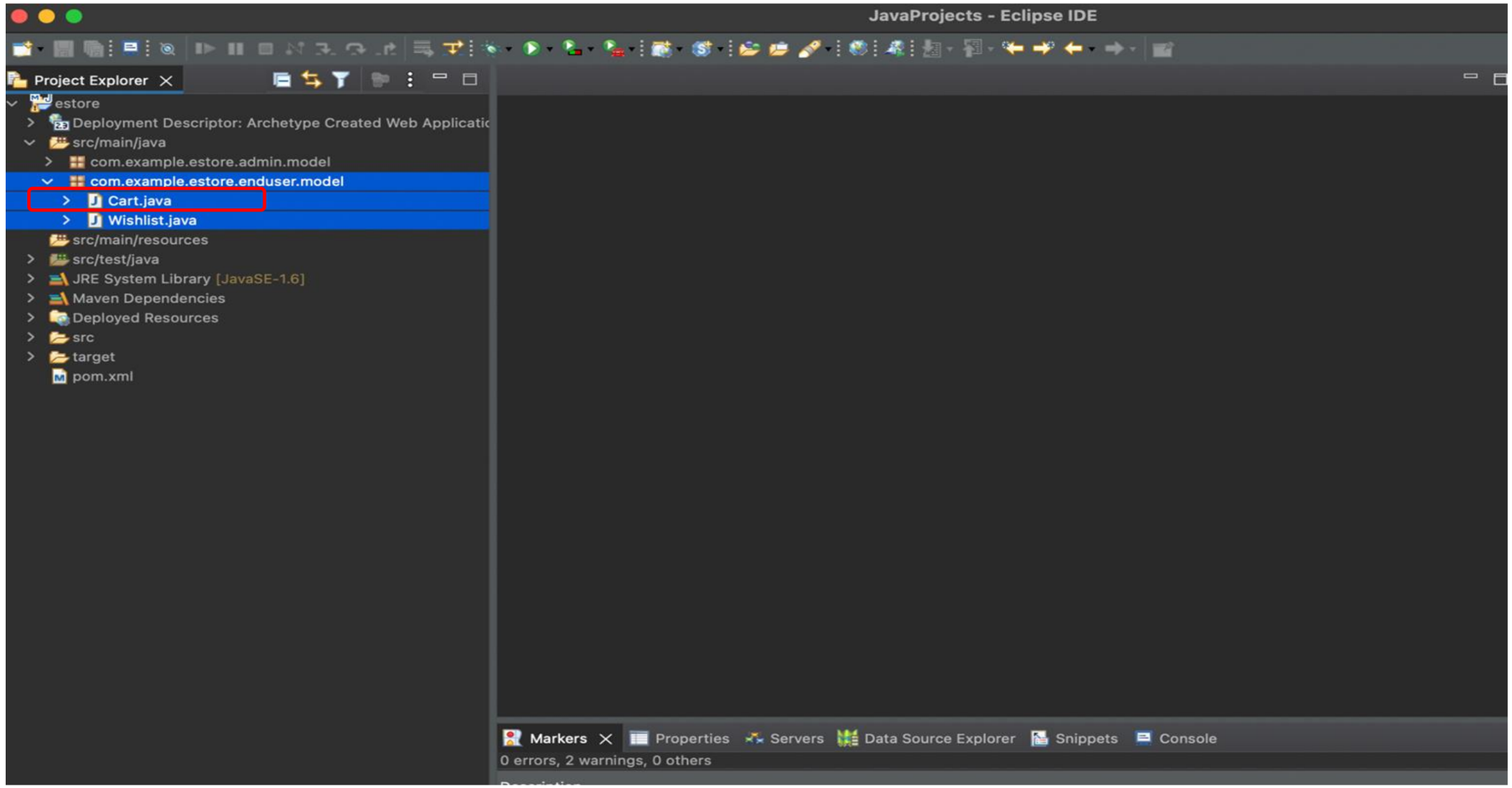
Create various POJO classes for the end user web app.

Wishlist.java : The user can wishlist the product. The details of the same will be managed by this class.



Create POJO Classes for the Admin Dashboard

Cart.Java : Cart which shall persist the order items of the user. Cart as class will hold the product and user details.



Create POJO Classes for the Admin Dashboard

Existing Classes to be used from Admin:

Users.java

For the users, registered on the web backend, the profile shall be managed using the same model from admin.

Orders.java

For the user to place an order, use the same model from admin.

OrderItems.java

For the products available in an order placed by user, use the same model from admin.

Create POJO Classes for the Admin Dashboard

Existing Classes to be used from Admin:

Products.java

For displace the products to the end user, we will use the same model from admin.

Shipments.java

For a user the placed order shipped, use the same model from admin.

Wishlist.java

A user can select some of the products for which purchases can be made later.

Wishlist model holds the details of the product for a particular user.

```
public class Wishlist {
   Integer wishListId;
      Integer productId;
      Integer userId;
}
```

Cart.java

A user moves the products to the cart to place an order.

Cart model shall contain the product and user details as attributes.

```
public class Cart {
   Integer cartId;
      Integer productId;
      Integer userId;
}
```

Design Patterns

DAO and Service

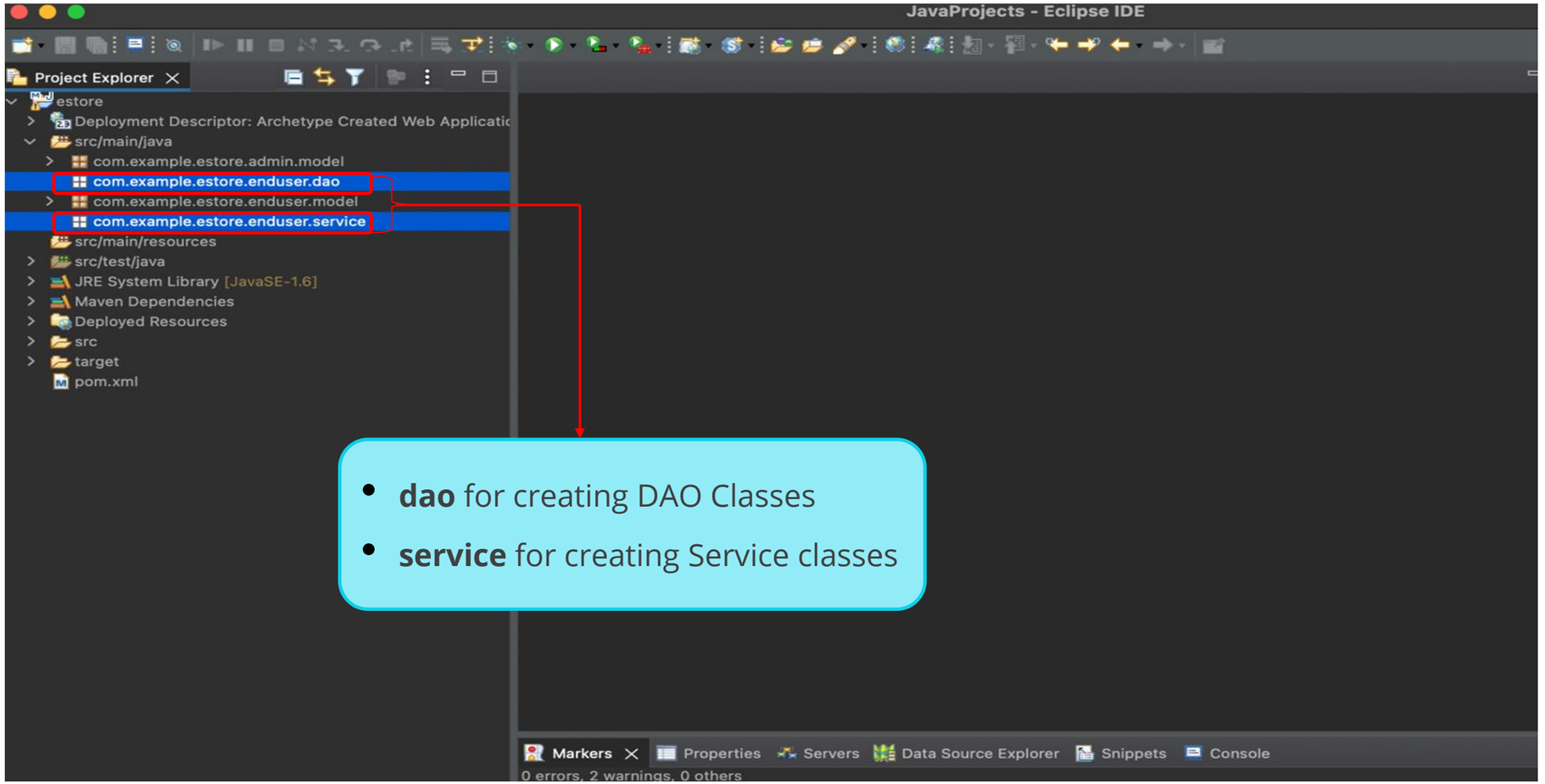
DAO and Service

DAO is Data Access Object and is used to separate the data persistence logic in a separate. layer.

The service will use the DAO and implement principle of Separation of Logic.

Creating Packages for Design Pattern Implementation

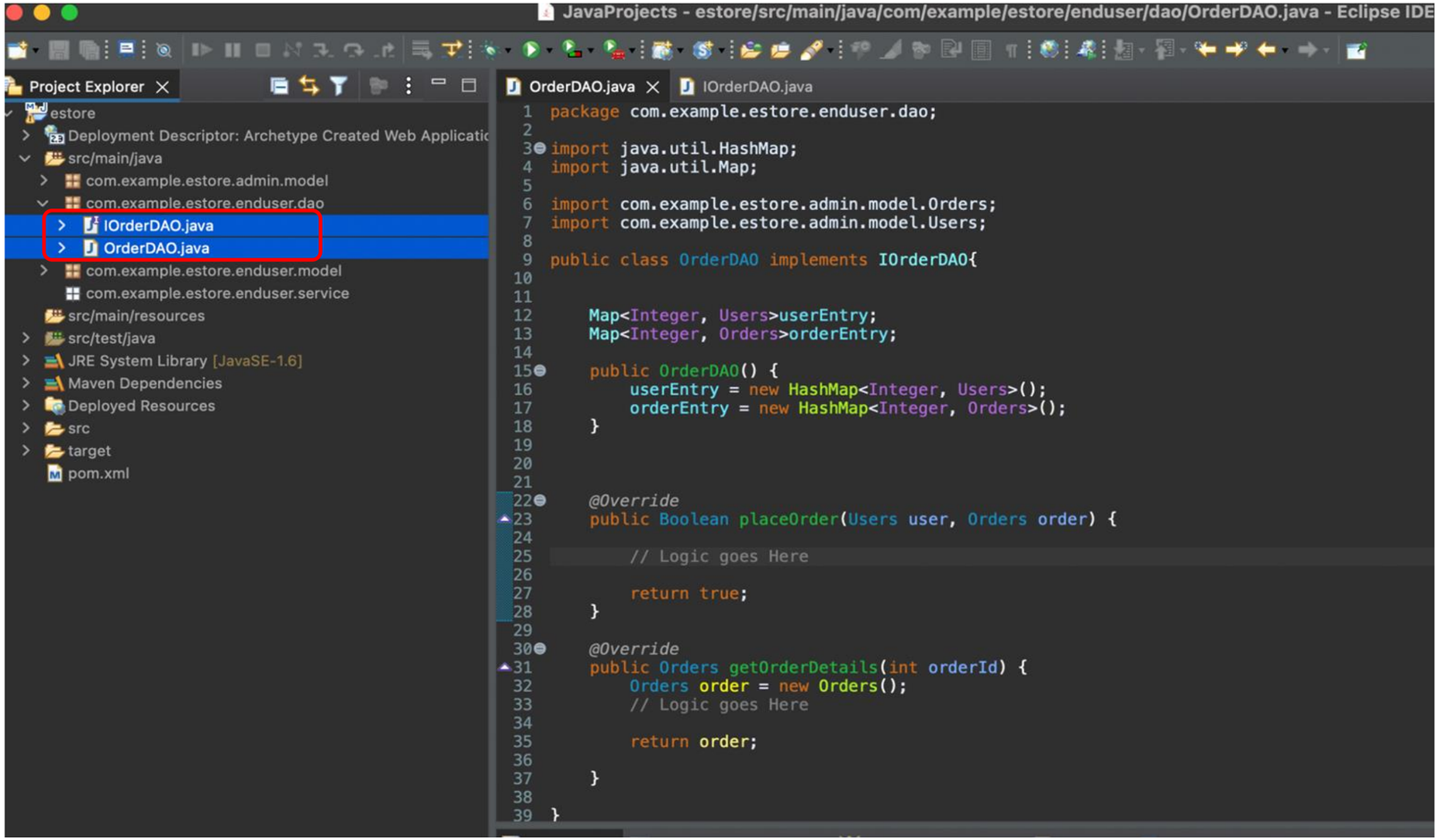
Create two packages:



- **dao** for creating DAO Classes
- **service** for creating Service classes

Implementing DAO

Create an interface classes IOrderDAO and a class OrderDAO, which shall implement the interface as shown.



IOrderDAO.java

Firstly create an interface which defines 2 methods:

placeOrder : A user can place the order using it.

getOrderDetails : A user can get the details of the order by giving order Id as input.

```
public interface IOrderDAO {

	public Boolean placeOrder(Users user, Orders order);
	public Orders getOrderDetails(int orderId);

}
```

OrderDAO.java

In OrderDAO class, both the methods are defined.

```
public class OrderDAO implements IOrderDAO{

      Map<Integer, Users>userEntry;
      Map<Integer, Orders>orderEntry;

      public OrderDAO() {
            userEntry = new HashMap<Integer, Users>();
            orderEntry = new HashMap<Integer, Orders>();
      }
```

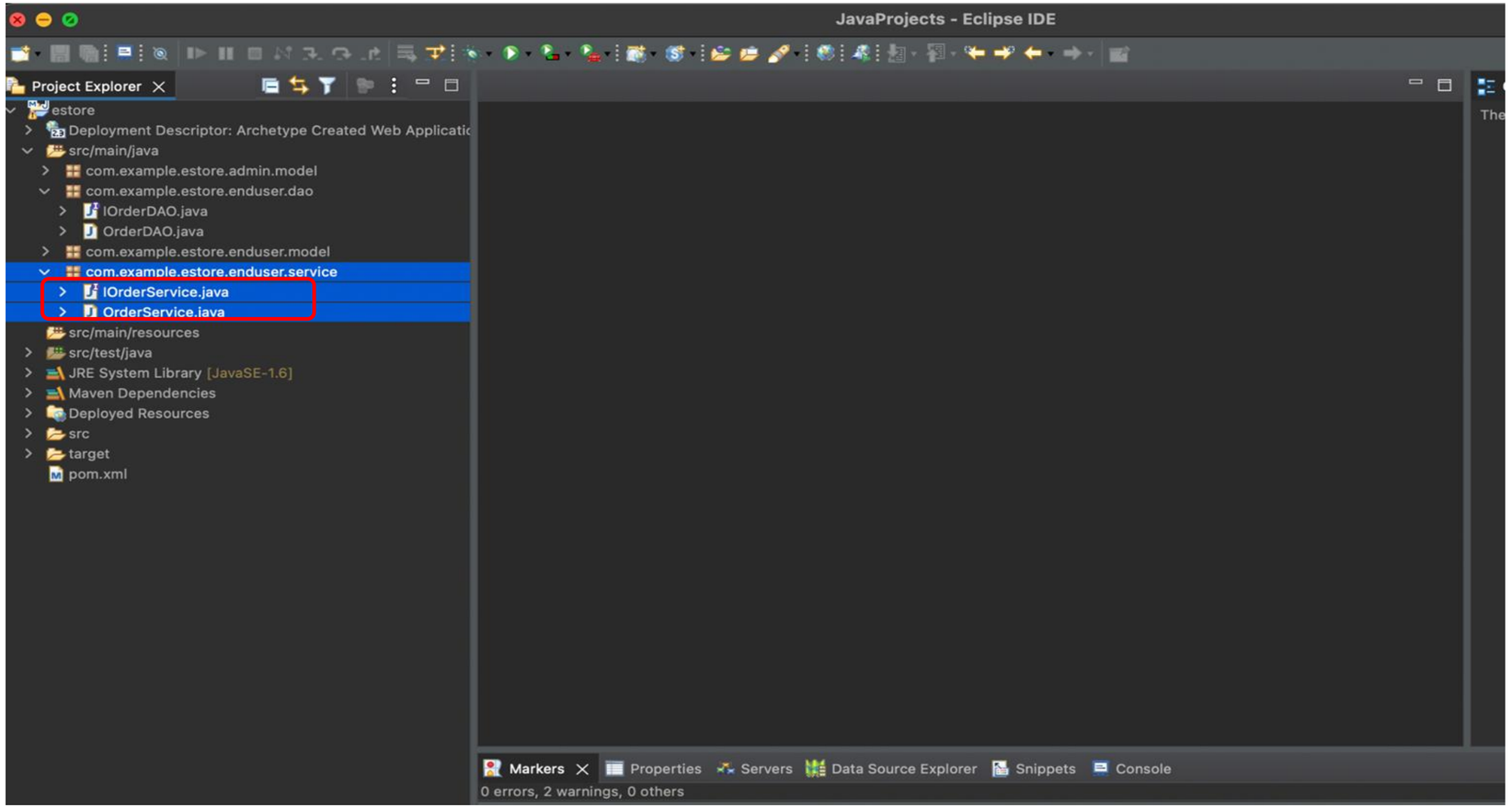
OrderDAO.java

HashMap can be used to temporarily hold the information for user details and order details.

```
@Override
	public Boolean placeOrder(Users user, Orders order) {
		// Logic goes Here
		return true;
	}
	@Override
	public Orders getOrderDetails(int orderId) {
		Orders order = new Orders();
		// Logic goes Here

		return order;
	}

}
```

Implementing Service

Create a service that will use the DAO and an interface IOrderService and a class OrderService implementing it.



IOrderService.java

Create an interface that defines a method placeOrder, using which the user can place the order.

```
public interface IOrderService {

      public Boolean placeOrder(Users user, Orders
order);

}
```

OrderService.java

OrderService class method placeOrder is defined. OrderDAO is used to place an order.

```
public class OrderService implements IOrderService{

      OrderDAO dao;

      public OrderService() {
            dao = new OrderDAO();
      }
```

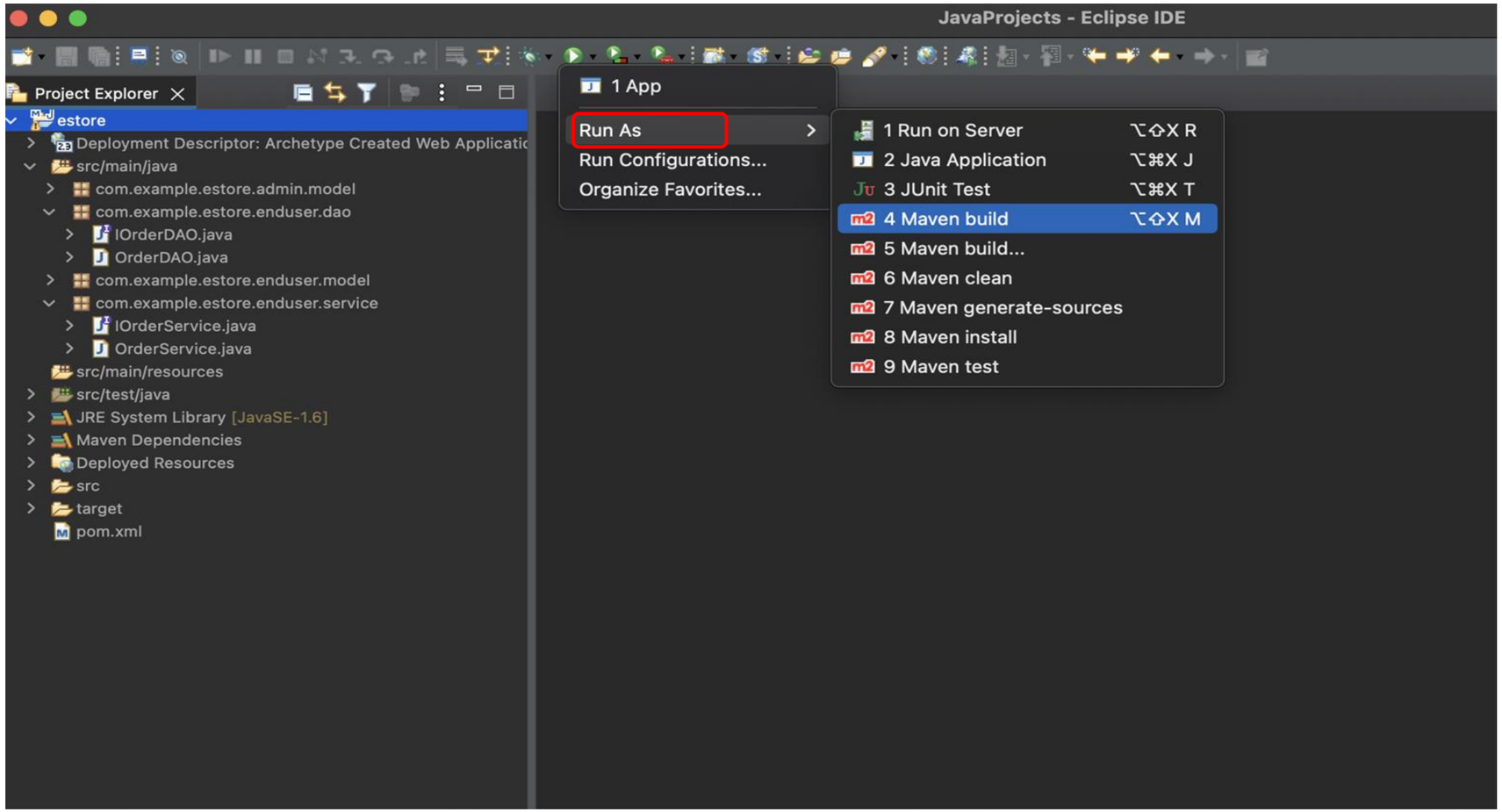
OrderService.java

Add some Exception Handling mechanism and implementation of validations using user defined method before the user places an order.

```
@Override
	public Boolean placeOrder(Users user, Orders order) {
		return dao.placeOrder(user, order);

	}
	// Some more validation methods before the order is placed
goes here

}
```

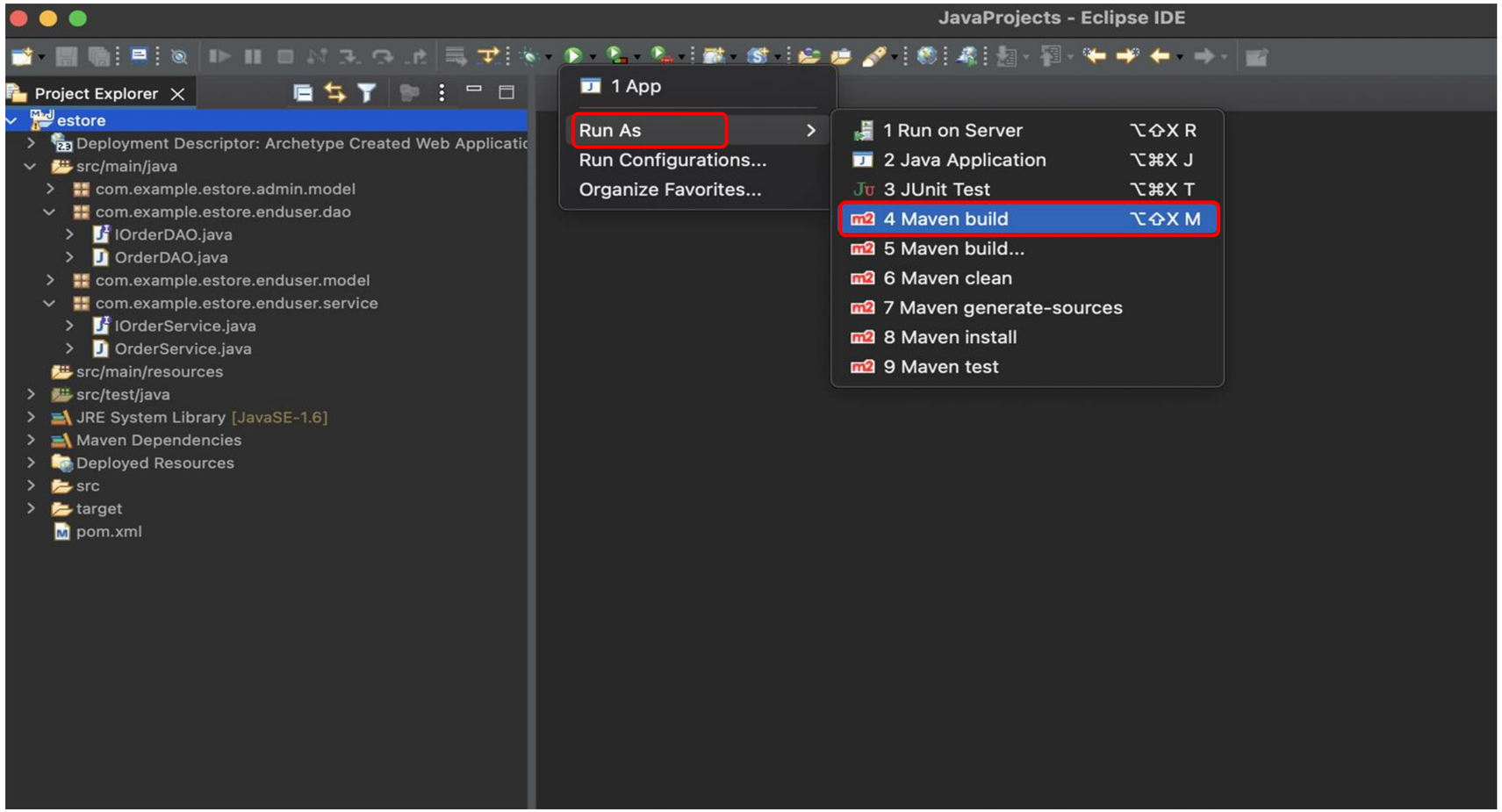
Build and Run the Project

Run the project to see if any errors exists.



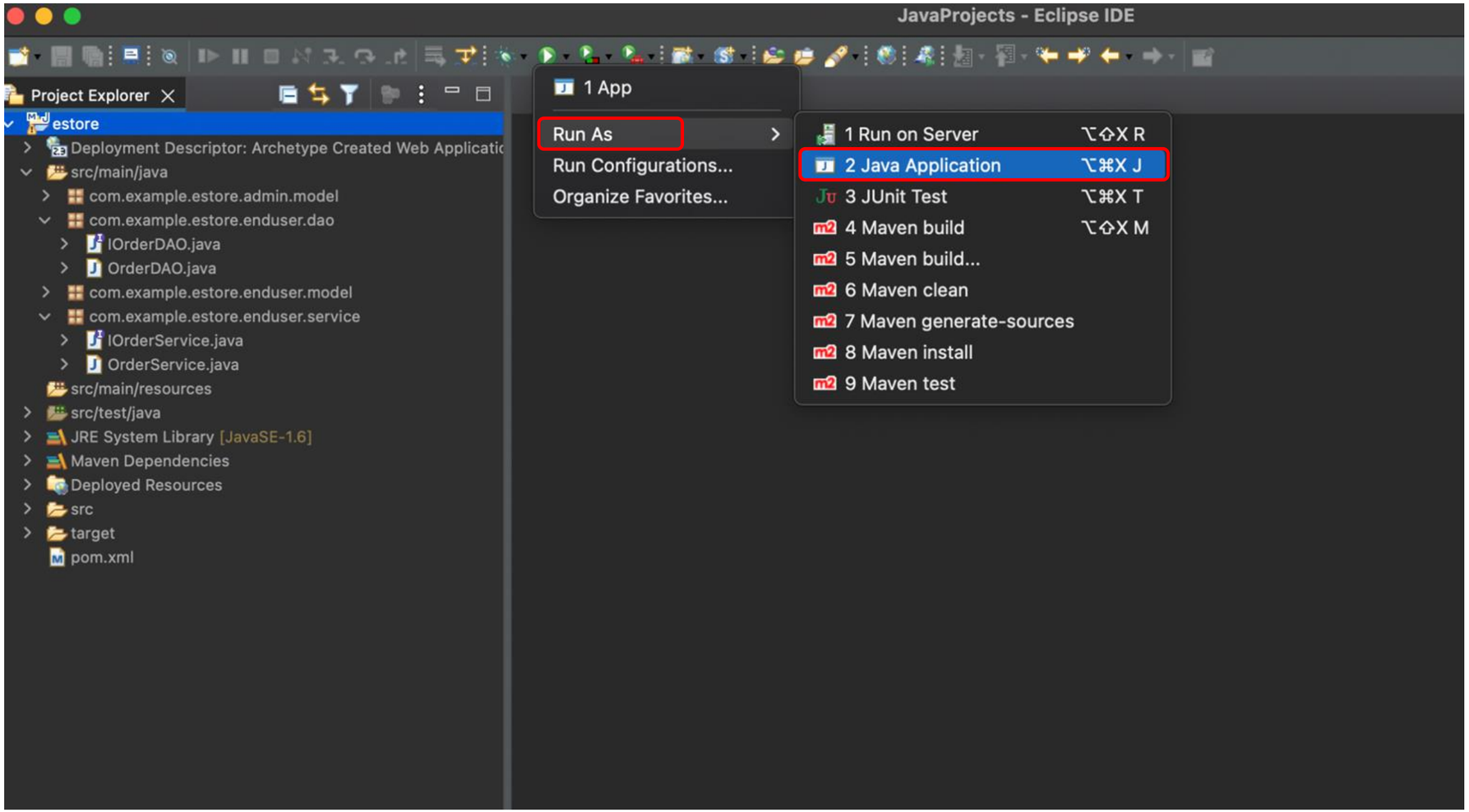
Build and Run the Project

Firstly, build the project by selecting the run button and then run as Maven Build. Follow the shortcuts as provided.



Build and Run the Project

Run the Project as Java Application. Follow the shortcuts as provided.



Key Takeaways

- Maven project is created using various steps.
- POM is the fundamental unit of work in maven, which is an xml file.
- POJO is a Plain Old Java Object which contains attributes for an object and along with it, constructors (default and parameterized), and a toString method.
- DAO is Data Access Object and is used to separate the data persistence logic in a separate layer.

Before the Next Class

Since you have successfully completed this session. Before next discussion you should go through:

- JDBC
- Servlets
- JSP

What's Next?

Now we have finished our Classes and Design Pattern for the Backend Project. In our next live discussion, we will:

- Explore how to configure dependencies
- Configure Apache Tomcat Web Server
- See how to create JSP and Servlet
- Work with Design Patterns
- Build a war file with Maven for Web Project