# ***E-Commerce Nexus Project***

## **Pre-Requisite:**

* Java JDK 17 (https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html)
* Apache Maven (https://maven.apache.org/download.cgi)
* MySQL Server (https://www.mysql.com/downloads/)
* IDE: IntelliJ IDEA (https://www.jetbrains.com/idea/download/?section=windows)
* Postman(https://www.postman.com/downloads/)

## **Development Walkthrough:**

For the sake of understanding I'm dividing the development into several phases.

## **Phase-1: Foundation and Core Backend API:**

### **Step-1: Initialize Your Spring Boot Project:**

Open browser and go to (https://start.spring.io/)

Fill in the Project Metadata on the left side of the page as follows:

* **Project:** Maven
* **Language:** Java
* **Spring Boot:** Choose a recent stable version (e.g., 3.3.1).
* **Group:** com.nexus
* **Artifact:** ecommerce
* **Name:** ecommerce
* **Description:** E-commerce Nexus Project
* **Package name:** com.nexus.ecommerce
* **Packaging:** Jar
* **Java:** 17
* **Add Dependenceies:**
  + **Spring Web:** Needed to build RESTful web applications. It includes the Apache Tomcat server by default.
  + **Spring Data JPA:** To persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
  + **MySQL Driver:** The specific JDBC driver required to connect to a MySQL database.
  + **Lombok:** A fantastic library that reduces boilerplate code like getters, setters, and constructors by using annotations.
  + **Spring Boot DevTools:** Provides fast application restarts, and other development-time features.

Click on Generate and unzip it to the local folder.

### **Step-2: Configure the MySQL Database Connection:**

Navigate to the src/main/resources folder in your project.

Open the application.properties file and add following configuration code:

* spring.datasource.url=jdbc:mysql://localhost:3306/ecommerce\_db?createDatabaseIfNotExist=true
* spring.datasource.username=your\_mysql\_username
* spring.datasource.password=your\_mysql\_password
* spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
* spring.jpa.hibernate.ddl-auto=update
* spring.jpa.show-sql=true
* spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**Notes:**

* Replace your\_mysql\_username and your\_mysql\_password with the username and password you use for your local MySQL server (a common default is root for the username).
* The property createDatabaseIfNotExist=true in the URL will automatically create a database named ecommerce\_db for you the first time the application runs.
* spring.jpa.hibernate.ddl-auto=update is a powerful development feature. It compares your Java entity classes to the database schema and automatically adds tables, columns, etc., as needed.

### **Step-3: Create First Data Model:**

Inside the src/main/java/com/nexus/ecommerce package, create a new sub-package named model.

Inside the model package, create a new Java class called Product.java.

**Notes:**

* @Entity: Marks this class as JPA entity, meaning it will be mapped to a database table.
* @Table(name = "products"): Specifies that this entity maps to a table named products. If omitted, it would default to a table name product.
* @Data: This is from Lombok. It's a shortcut that automatically generates all the boilerplate code: getters for all fields, setters for all non-final fields, and appropriate toString, equals, and hashCode implementations.
* @Id: Designates this field (id) as the primary key for the table.
* @GeneratedValue(strategy = GenerationType.IDENTITY): Configures the way the primary key is generated. IDENTITY means the database will handle auto-incrementing the value.
* @Column: Used to specify details about the column that a field will be mapped to. For example, nullable = false makes it a required field, and length = 1000 sets the column size.

### **Step 4: Build the Repository Layer:**

Inside your project's main package (com.nexus.ecommerce), create a new sub-package named repository.

Inside the repository package, create a new Java interface (not a class) called ProductRepository.java.

**Notes:**

* @Repository: This annotation marks the interface as a Spring Data repository. It tells Spring to treat it as a bean and enables exception translation, converting database-specific exceptions into a consistent Spring exception hierarchy.
* extends JpaRepository<Product, Long>: This is the key part.
  + We are extending the JpaRepository interface.
  + The first generic type, Product, tells the repository that it will be managing Product entities.
  + The second generic type, Long, specifies the data type of the Product entity's primary key (id).

### **Step 5: Create the Product Service:**