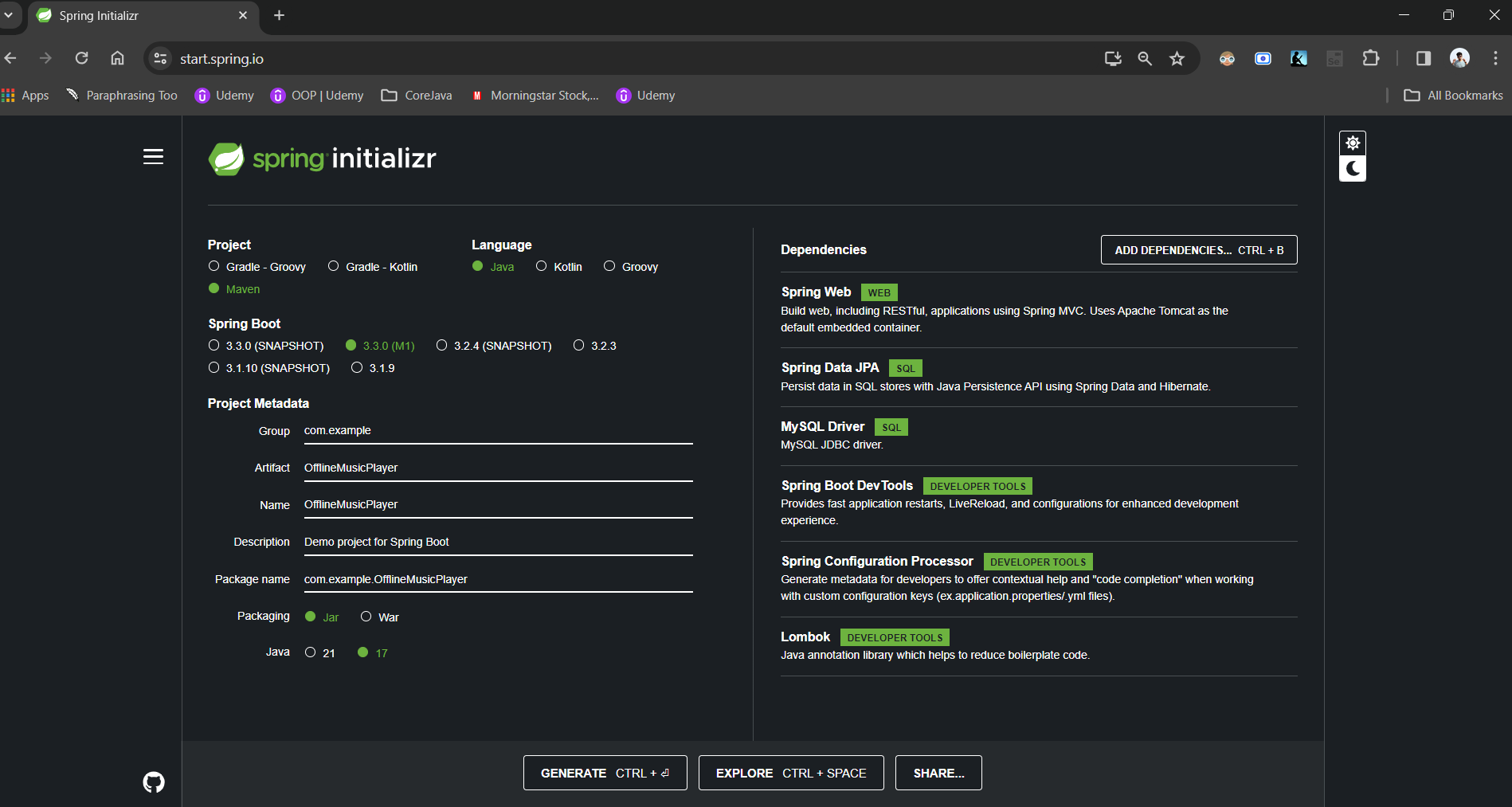
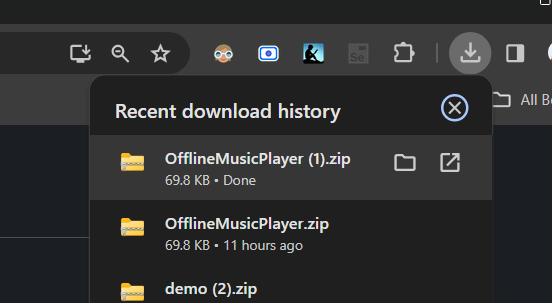
Based on Java+Springboot

Requirements:

Go to <https://start.spring.io/> to get all the mentioned dependancies.

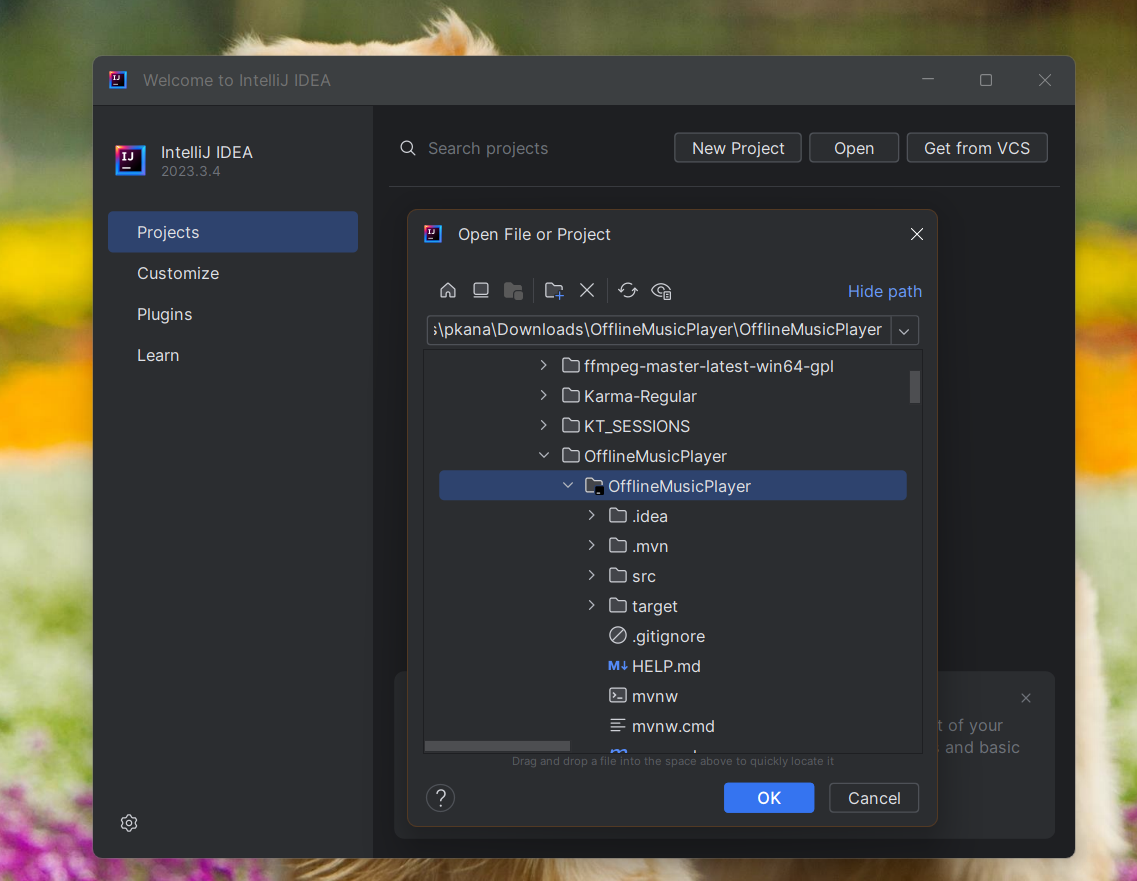


Click on generate and extract the downloaded zip.

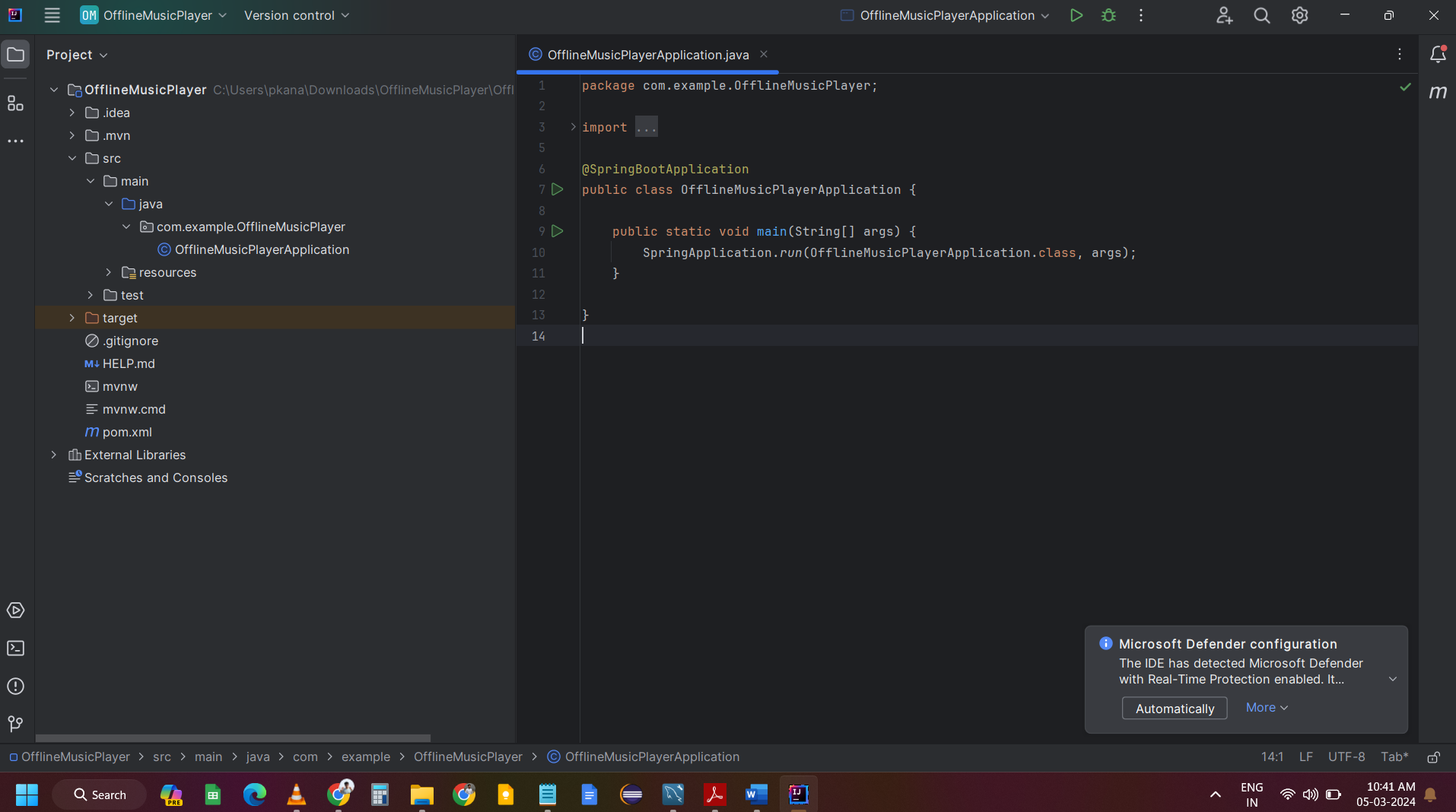


Open Intellij Community edition and click on Open Project.

Then navigate to the path where you’ve extracted.



If everything is works, then you should see the window

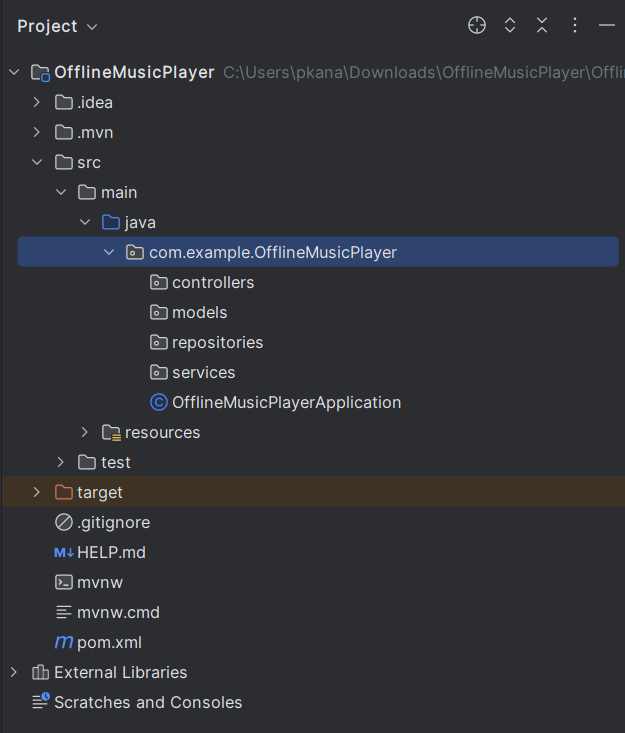


Certainly! Let’s break down the project structure for the “OfflineMusicPlayer” application based on the image you provided:

1. **Root Folder (“OfflineMusicPlayer”)**:
   * The top-level directory of the project.
   * Contains several subfolders and files.
2. **Subfolders and Files**:
   * .idea: This folder typically holds project-specific settings and configuration files for JetBrains IDEs.
   * .mvn: A folder that might contain Maven wrapper scripts or configuration settings.
   * src:
     + main:
       - java: Contains the main Java source code files.
         * com.example.OfflineMusicPlayer: Specific package where the main application Java classes are stored.
       - resources: A directory for non-code resource files (e.g., configuration files, images).
     + test: This directory is usually for test code, but it appears empty in the snapshot.
   * target:
     + Contains compiled bytecode, JARs, classes, etc., generated after building the project with Maven.
     + classes: Compiled bytecode of the project’s source code.
       - com.example: Bytecode corresponding to this specific package.
         * OfflineMusicPlayerApplication: Compiled class file for this specific Java class.
     + generated-sources: May contain sources generated during the build process.
     + annotations: Might store annotation-related generated sources or configurations.
   * HELP.md: A Markdown file that likely contains help documentation or instructions related to this project.
   * mvnw & mvnw.cmd: Maven wrapper scripts allowing users to build projects without requiring Maven to be installed locally.
   * pom.xml: The Project Object Model (POM) XML file defining the project structure, dependencies, plugins, goals, etc. It’s essential for building projects with Maven.
3. **Summary**:
   * The project appears to follow a Maven-based structure.
   * Main application code resides in the src/main/java directory.
   * Resources are stored in the src/main/resources folder.
   * The target directory holds compiled output.
   * The pom.xml file defines project details and dependencies.

Remember that this is a high-level overview, and specific details may vary based on the actual project. If you have any specific questions about a particular aspect, feel free to ask!

Let’s create necessary packages for this project.



Controllers :

Models :

The objects that we want to store the data for, will come in the models.

It’s like a representation of Table in the database. Each class is defined under model package, will have the Table of the same name as class name in the Database (MySQL Workbench).

Let’s say you want to store a Product information in the database.

Table name : Product

Column 1 : id 🡪 (Id of Product)

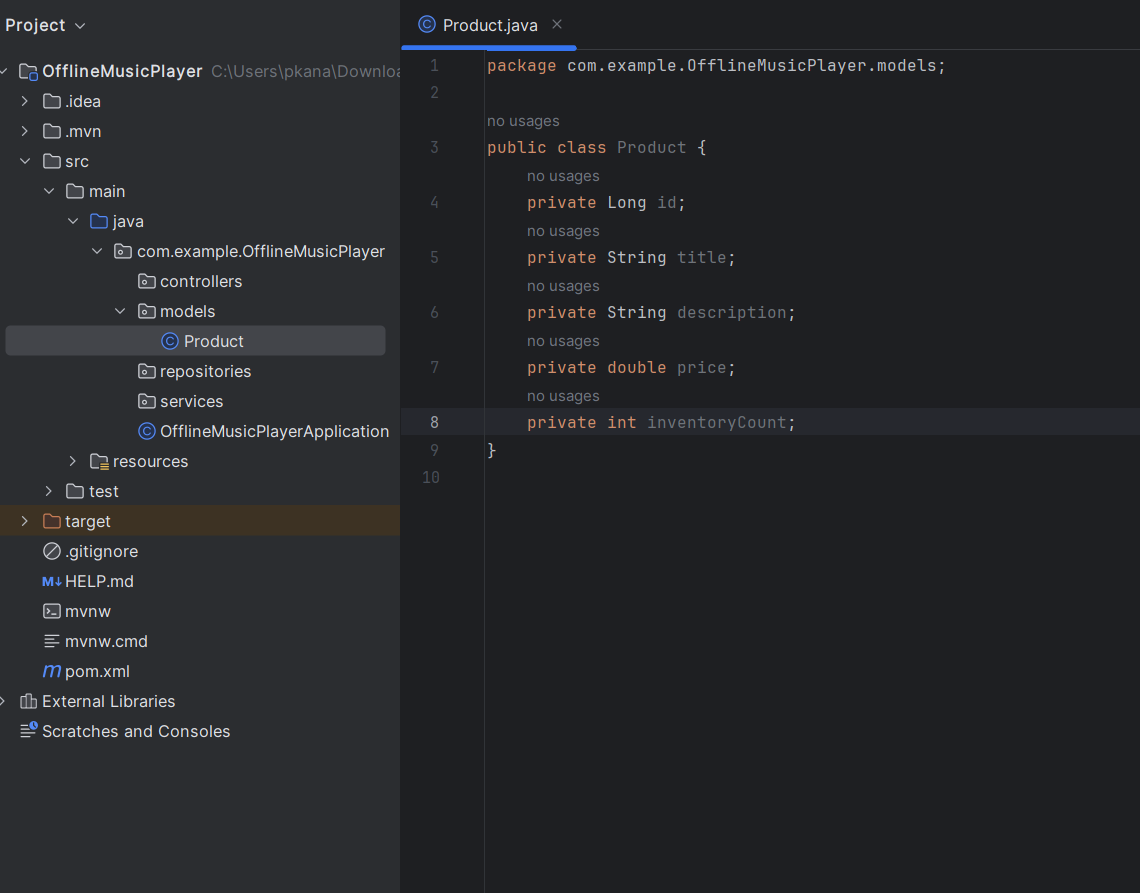
Column 2 : title 🡪 (Name of Product)

Column 3 : description 🡪 (Brief description of product)

Column 4 : price 🡪 (Cost of the product)

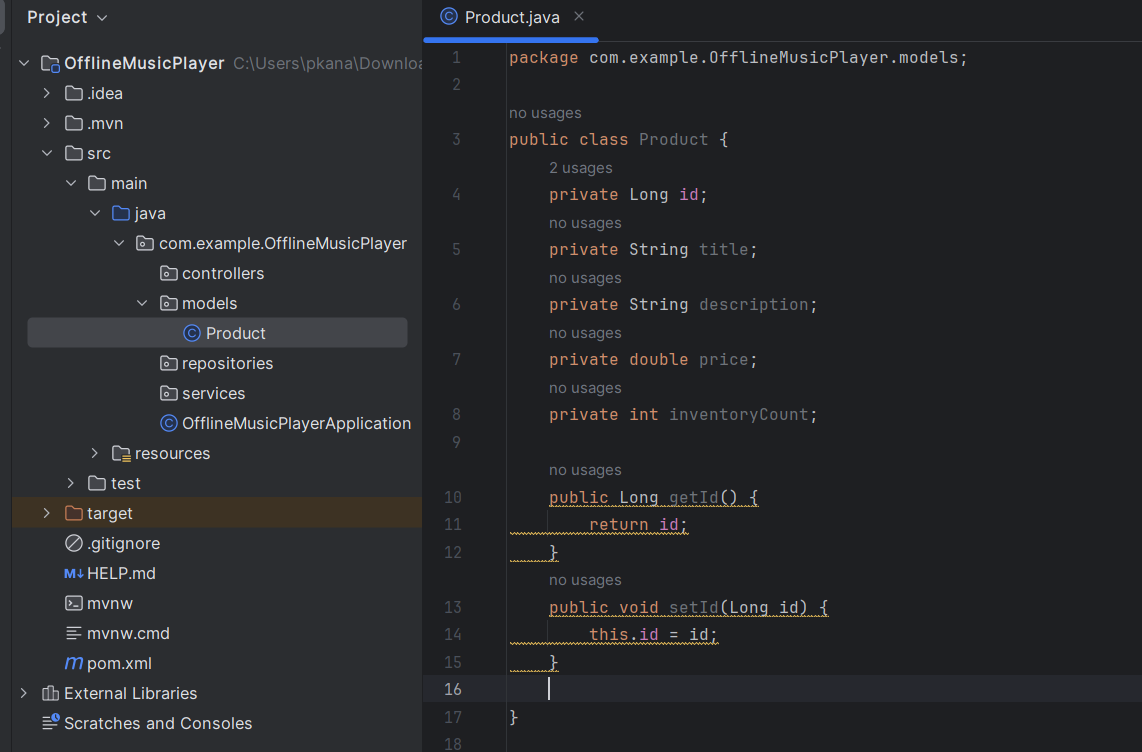
Column 5 : inventoryCount 🡪 ( How many products are available in the Inventory)

Now, how can we write it in Java class?



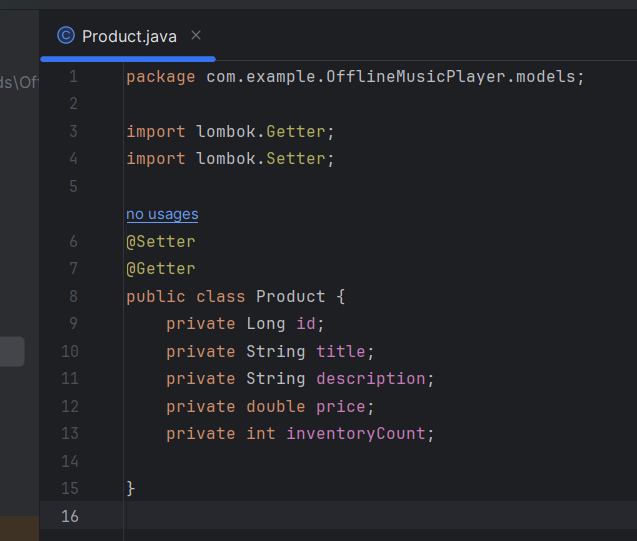
Wait.. the variables are private then how can we access them outside the class?

By using Getters and Setters



Don’t you think if you have more variables, and for each variable you want to create Gettes and Setters method. The code will be lengthier, isn’t it?

So, to address that issue we have a library called LOMBOK. You just have to add @Getter and @Setter just above class.

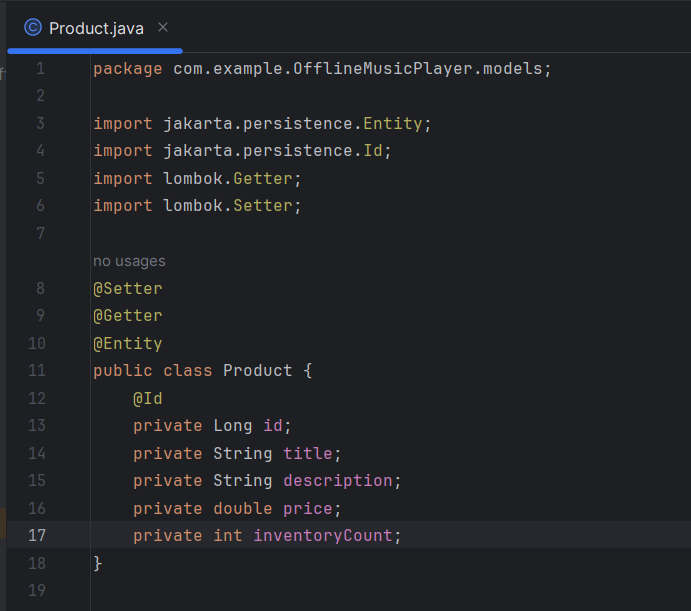


Magically! you will be able to use all the Getters and Setters methods of this Product Class

Now, Let’s Create Product Table in the database to store the product details.

As of now, can we say that the above Product is just a class my project? Yes.

So, to create a Table for that class (Product class) just use @Entity above the class.



What is @Entity?

This annotation is from SpringDataJPA library, used for creation of Table in the database.

When you run the code, automatically SpringDataJPA will create a table in the DB.

What is @Id?

This annotation we are adding a Primary Key for Id column.

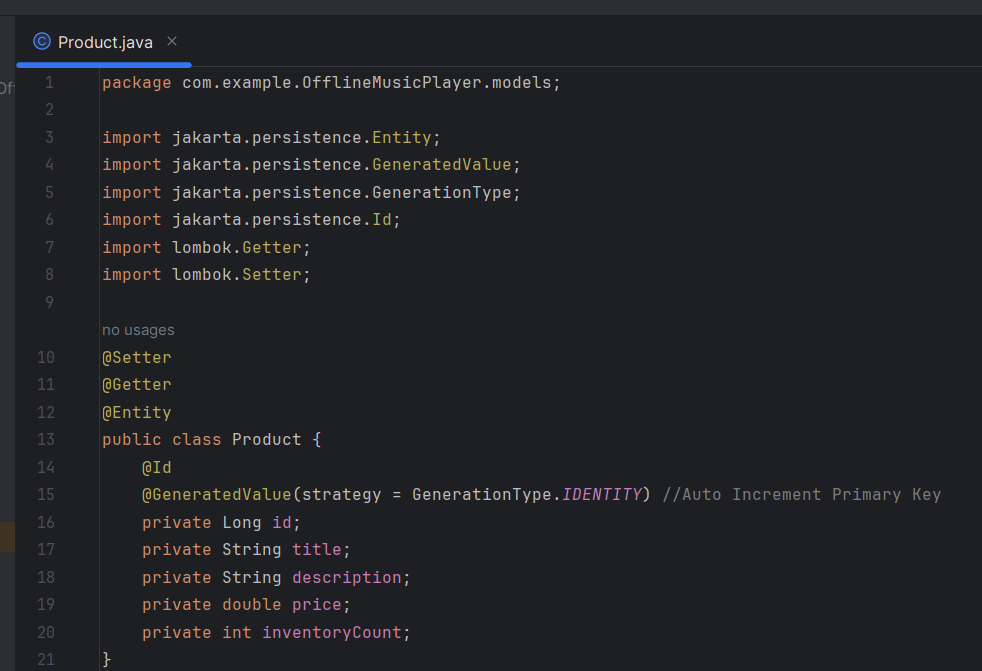
Its like we are telling the SpringDataJPA that, whatever is the database operations is performing it will be based on id column.

Apart from that,

Let’s say you want to insert the data in the table, would you create Id’s Manually? No right?

It should be auto incremented.

For that we will use @GeneratedValue(strategy = GenerationType.IDENTITY)



Now, Lets connect to the Database.

Let’s say, we are Creating a product service, then what are things we need for the service?

Repositories :

Services :

What Dependencies We’ve added and Why?

Spring Data JPA :

SpringDataJPA is very very powerful tool, to make your life easy when you interact with the database.

Lets say when you interacting with MySQL database, you would be Creating a Table, Deleting tables, Getting the data. For that writing queries like (Select \* from users where id is …) etc.

All these thing SpringDataJPA will do it automatically for you 😊

SpringWeb :

It is the dependace which is used to expose your API’s as a WebAPI.

LOMBOK

It is Java library that provides you functionality like making Getters and Setters method for class variables..etc.