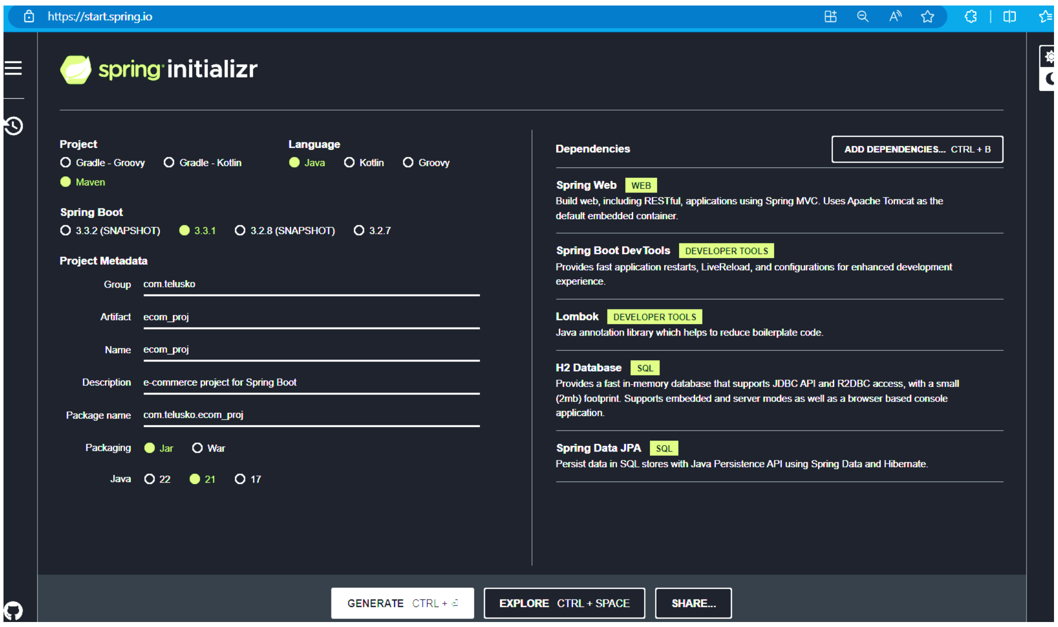
21-Spring Project | Model-Based Project Configuration

It's time to build the backend of the e-commerce project now that we have covered the frontend technology in the previous chapter.

Configuring the backend:

Using the same methodology, we will create a brand-new Spring web project from the ground up using the spring initializer website.



https://start.spring.io

spring initializr

Project
( ) Gradle - Groovy ( ) Gradle - Kotlin [✓ Maven]

Language
[✓ Java] ( ) Kotlin ( ) Groovy

Spring Boot
( ) 3.3.2 (SNAPSHOT) [✓ 3.3.1] ( ) 3.2.8 (SNAPSHOT) ( ) 3.2.7

Project Metadata

Group	com.telusko
Artifact	ecom_proj
Name	ecom_proj
Description	e-commerce project for Spring Boot
Package name	com.telusko.ecom_proj
Packaging	[✓ Jar] ( ) War
Java	( ) 22 [✓ 21] ( ) 17

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Boot DevTools DEVELOPER TOOLS
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok DEVELOPER TOOLS
Java annotation library which helps to reduce boilerplate code.

H2 Database SQL
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

Project name: ecom_proj

Project type: Maven

Language: Java

Packaging type: Jar

JDK version: 21

Spring Boot Version: 3.3.0

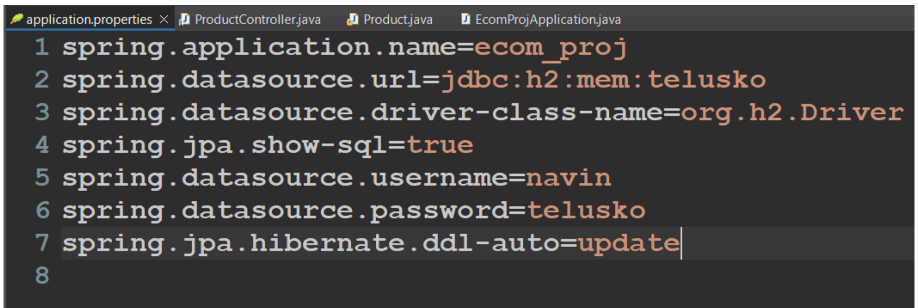
Project metadata:

I. **Group:** com.telusko
II. **Artifact:** ecom_proj
III. **Name:** ecom_proj
IV. **Description:** Ecommerce project for springboot
V. **Package-name:** com.telusko.ecom_proj

Dependencies: SpringWeb, Lombok, Spring DataJPA, Spring Boot Dev tools, and H2-database.

After that, in your IDE, click on Generate to unzip the file.

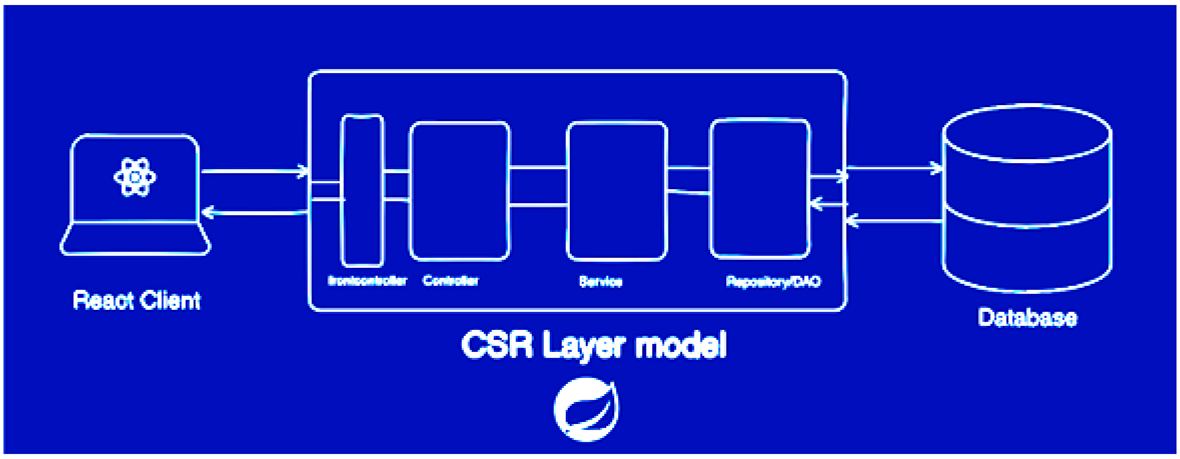
Unzipping the project after it has finished loading, let's try to verify which port our project is using so that it won't cause any issues later. Our embedded Tomcat server's default port is 8080, but we can modify it.
Our project is now operating on port 8080 following successful execution, so let's start by configuring our H2-database in the properties file and setting up the url, datasource, and other parameters.



```
spring.application.name=ecom_proj
spring.datasource.url=jdbc:h2:mem:telusko
spring.datasource.driver-class-name=org.h2.Driver
spring.jpa.show-sql=true
spring.datasource.username=navin
spring.datasource.password=telusko
spring.jpa.hibernate.ddl-auto=update

```

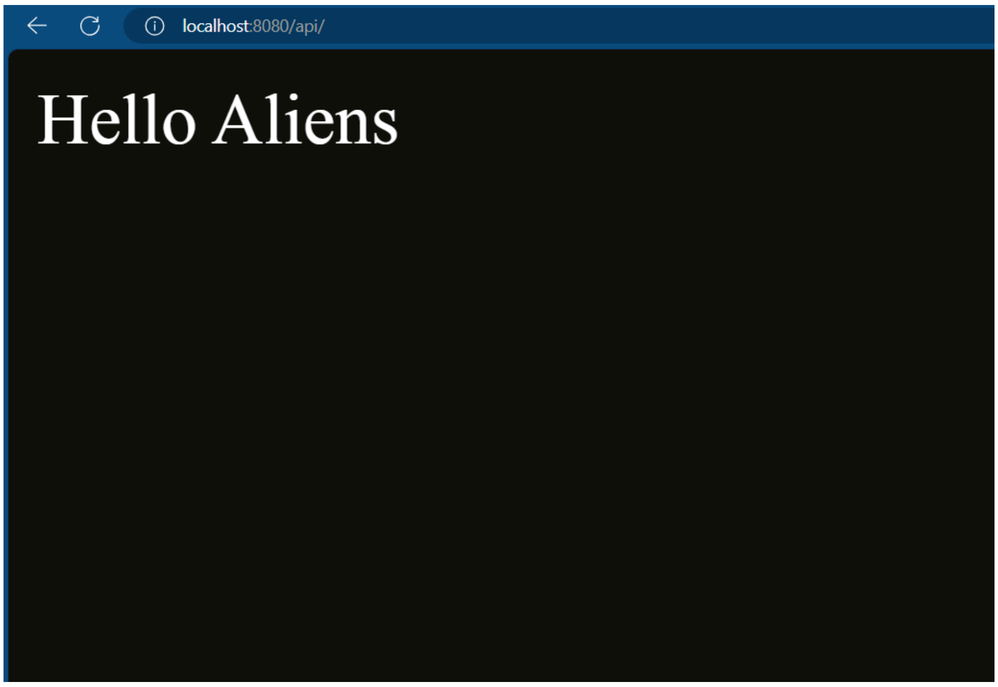
We are now developing our controller, service, and repository layers while adhering to our CSR layer architecture and creating package layers.

In order for our model Product class to be displayed, it must contain the following details: product id (primary key), name, description, price, category, availability, quantity, and, if it is feasible, an image of the product.



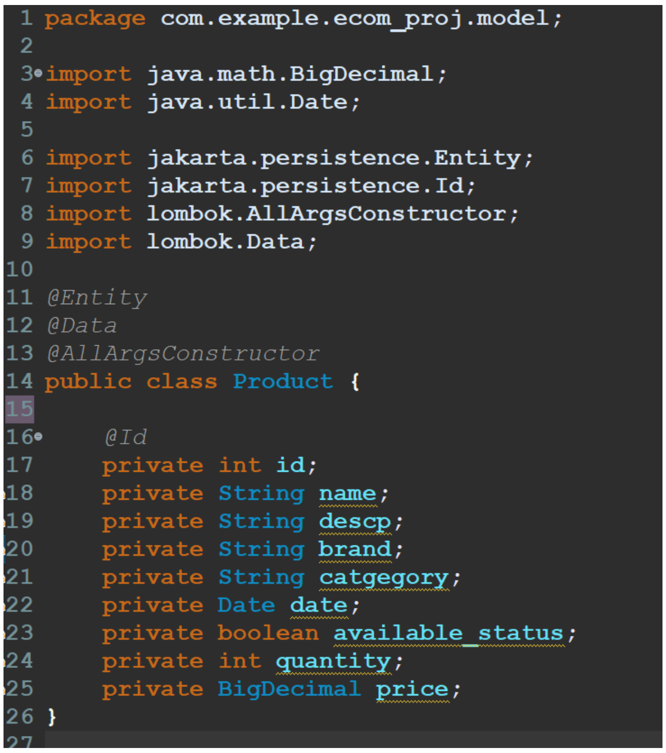
Packages:

Controller: The @RestController and @RequestMapping annotations will be used to declare our ProductController class. The method greet will return the string "Hello-Aliens" and the @RequestMapping endpoint will be "/" Having configured our browser to visit the url ***localhost:8080/api/*** to see if we are receiving our output.



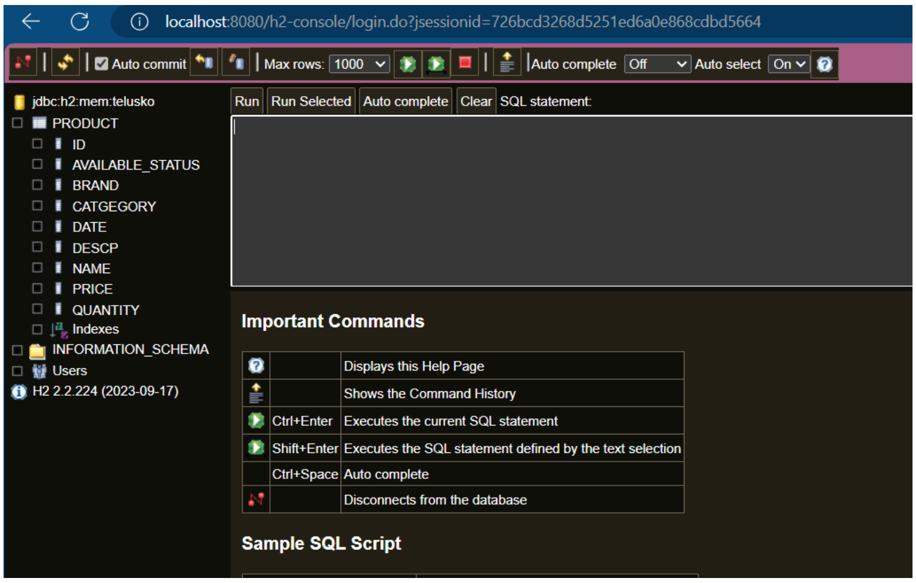
```
Hello Aliens
```

As previously mentioned, we are using Lombok dependency to create setter-getters, tostrings, and all argument constructors with @Data and @AllArgsContructor. Additionally, we are using data JPA, so in order to create a corresponding table, we must add @Entity to the class declaration and a @Id annotation to the variable id so that our product id is treated as the primary key.



```java
package com.example.ecom_proj.model;

import java.math.BigDecimal;
import java.util.Date;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;

@Entity
@Data
@AllArgsConstructor
public class Product {

    @Id
    private int id;
    private String name;
    private String descp;
    private String brand;
    private String catgegory;
    private Date date;
    private boolean available_status;
    private int quantity;
    private BigDecimal price;
}
```

Examining and monitoring the project to see if our JPA and all dependencies are operating as intended by going to localhost:8080/h2-console to access the H2-console



localhost:8080/h2-console/login.do?jsessionid=726bcd3268d5251ed6a0e868cdbd5664

Auto commit | Max rows: 1000 | Auto complete: Off | Auto select: On

```
jdbc:h2:mem:telusko
PRODUCT
    ID
    AVAILABLE_STATUS
    BRAND
    CATGEGORY
    DATE
    DESCP
    NAME
    PRICE
    QUANTITY
    Indexes
INFORMATION_SCHEMA
Users
H2 2.2.224 (2023-09-17)
```

Run | Run Selected | Auto complete | Clear | SQL statement:

Important Commands

		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script

Now, we have developed these two layers and their corresponding outputs. In the upcoming chapter, we will examine the implementation of additional layers and the process of adding products to the database.

Up until then, continue studying and have fun coding ☺ 👾