Java Spring Boot and Maven Setup for REST API with MS SQL Server Database

The steps to set up a Spring Boot project using Maven and configure it to work with an MS SQL Server database for REST API development.

Prerequisites

Ensure you have the following installed:

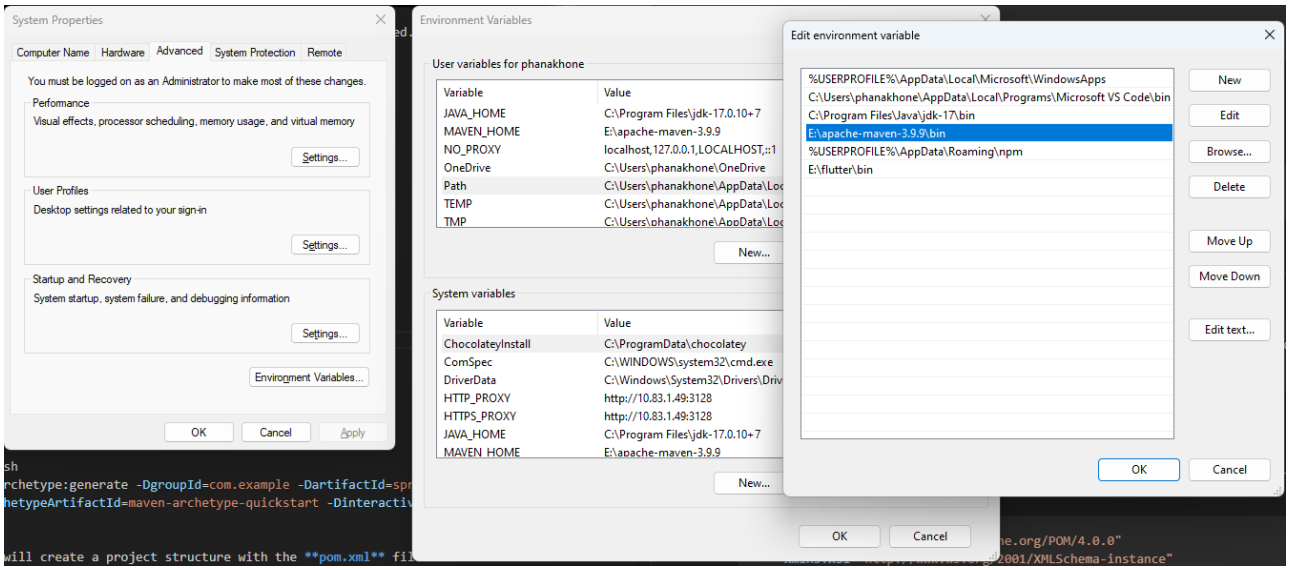
1. Java Development Kit (JDK) - Java 17 or higher is recommended.
    - Download JDK

      Verify installation:

      ```
      java -version
      ```
2. Maven - Apache Maven 3.6.3 or higher.
    - Download Maven

      Verify installation:

      ```
      mvn -version
      ```
3. Set Environment for Windows
    - JAVA_HOME=<to_path_java_jdk>
    - MAVEN_HOME=<to_path_maven>
    - Add maven bin dir to Path: <to_path_maven>/bin



- Verify environment with Command Prompt

```
echo %JAVA_HOME%
```

```
echo %MAVEN_HOME%
```

```
echo %PATH%
```

Create a New Maven Project

- Create the project manually:

```
mvn archetype:generate -DgroupId=com.example -DartifactId=springboot-mssql -
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

This will create a project structure with the **pom.xml** file.

- Add Dependencies to pom.xml

In your **pom.xml**, add the following dependencies for Spring Boot, MS SQL Server, and other required components:

```
<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>springboot-mssql</artifactId>
    <version>1.0-SNAPSHOT</version>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.0</version>
        <relativePath/>
    </parent>

    <dependencies>
        <!-- Spring Boot Starter Web for REST API -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Spring Boot Starter Data JPA for database access -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- MS SQL Server JDBC Driver -->
        <dependency>
            <groupId>com.microsoft.sqlserver</groupId>
            <artifactId>mssql-jdbc</artifactId>
            <version>9.4.0.jre11</version>
        </dependency>

        <!-- Spring Boot Starter Test for unit tests -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

- Configure the application.properties File

Configure your Spring Boot application to connect to the MS SQL Server. In src/main/resources/application.properties, add the following configurations:

```
# Server configuration
server.port=8080

# MS SQL Server configuration
spring.datasource.url=jdbc:sqlserver://localhost:1433;databaseName=mydb
spring.datasource.username=your_username
spring.datasource.password=your_password
spring.datasource.driver-class-
name=com.microsoft.sqlserver.jdbc.SQLServerDriver

# Hibernate JPA settings
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.SQLServerDiale
ct
4. Create the JPA Entity Class
Create an entity class to represent a database table. Example:

java
Copy code
package com.example.springbootmssql.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private double price;

    // Getters and Setters
}
```

- Create the Repository Interface Create a repository interface to interact with the database:

```
package com.example.springbootmssql.repository;

import com.example.springbootmssql.entity.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
```

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {
}
6. Create a REST Controller
Create a controller to handle HTTP requests:

java
Copy code
package com.example.springbootmssql.controller;

import com.example.springbootmssql.entity.Product;
import com.example.springbootmssql.repository.ProductRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/products")
public class ProductController {

    @Autowired
    private ProductRepository productRepository;

    @GetMapping
    public List<Product> getAllProducts() {
        return productRepository.findAll();
    }

    @PostMapping
    public Product createProduct(@RequestBody Product product) {
        return productRepository.save(product);
    }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) {
        return productRepository.findById(id).orElseThrow(() -> new
RuntimeException("Product not found"));
    }

    @PutMapping("/{id}")
    public Product updateProduct(@PathVariable Long id, @RequestBody Product
productDetails) {
        Product product = productRepository.findById(id).orElseThrow(() ->
new RuntimeException("Product not found"));
        product.setName(productDetails.getName());
        product.setPrice(productDetails.getPrice());
        return productRepository.save(product);
    }

    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) {
        Product product = productRepository.findById(id).orElseThrow(() ->
```

```
        new RuntimeException("Product not found"));
                productRepository.delete(product);
            }
        }
```

Run the Application

To run the Spring Boot application, you can use the following Maven command:

```
mvn spring-boot:run
```

Once the application is running, the REST API will be available at http://localhost:8080/api/products.

Testing the API

You can use tools like Postman or cURL to test the API.

Example of getting all products using curl:

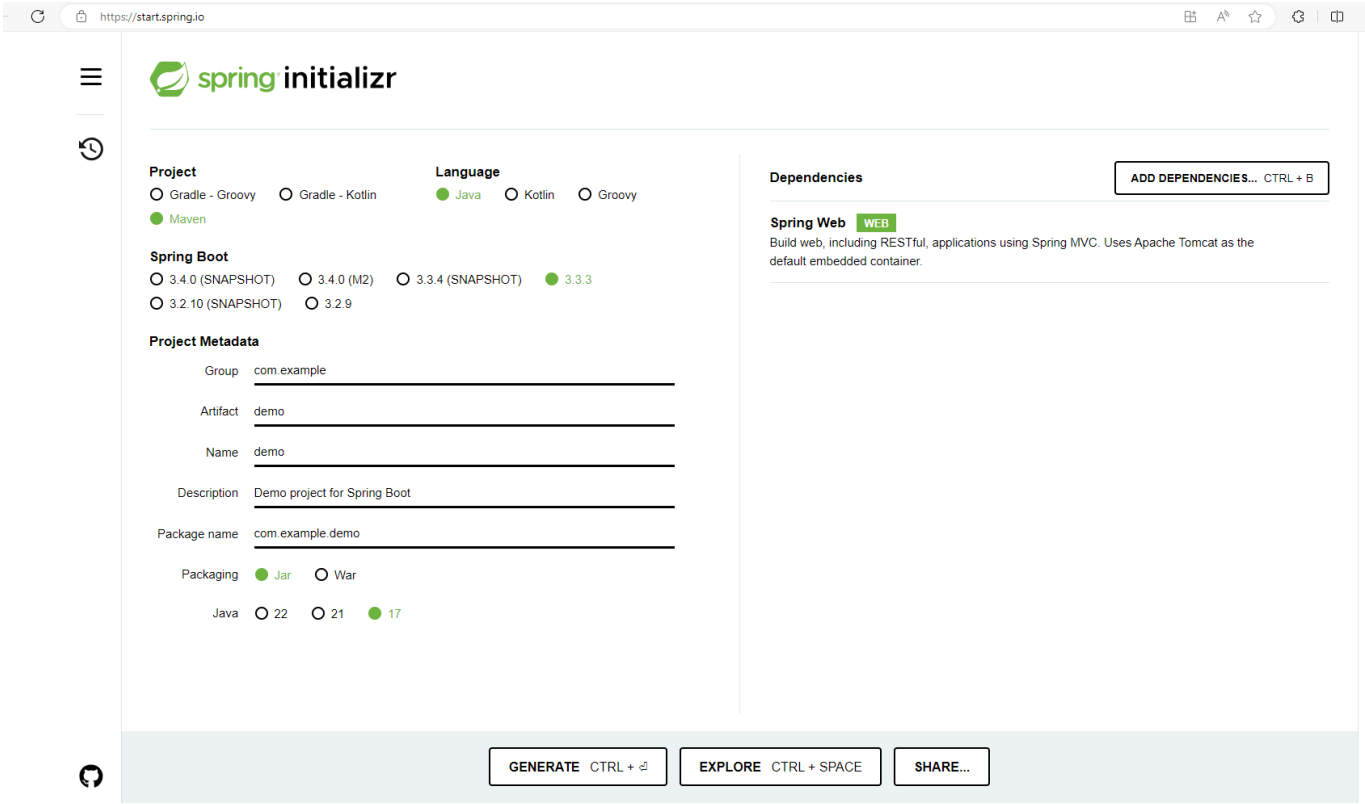
```
curl -X GET http://localhost:8080/api/products
```

Example of creating a new product:

```
curl -X POST http://localhost:8080/api/products \
-H "Content-Type: application/json" \
-d '{"name": "Laptop", "price": 999.99}'
```

Create a New Maven Project (Alternative)

Use start.spring.io to create a “web” project. In the “Dependencies” dialog search for and add the “web” dependency as shown in the screenshot. Hit the “Generate” button, download the zip, and unpack it into a folder on your computer.


https://start.spring.io
spring initializr
Project
Gradle - Groovy
Gradle - Kotlin
Maven
Language
Java
Kotlin
Groovy
Spring Boot
3.4.0 (SNAPSHOT)
3.4.0 (M2)
3.3.4 (SNAPSHOT)
3.3.3
3.2.10 (SNAPSHOT)
3.2.9
Project Metadata
Group com.example
Artifact demo
Name demo
Description Demo project for Spring Boot
Package name com.example.demo
Packaging Jar War
Java 22 21 17
Dependencies
ADD DEPENDENCIES... CTRL + B
Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
GENERATE CTRL + ⏎
EXPLORE CTRL + SPACE
SHARE...