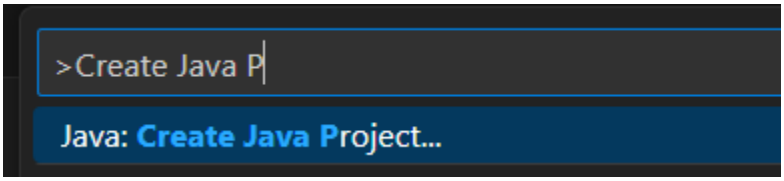
Spring Boot API Setup Guide

With MS SQL Server Database

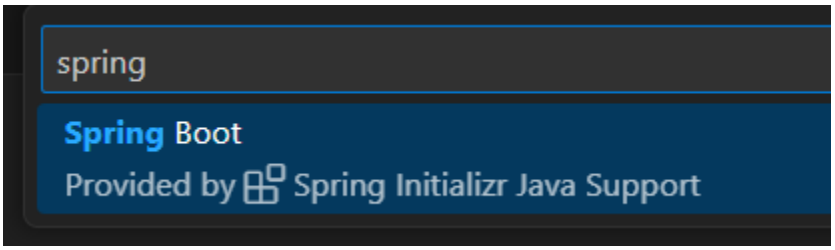
1. Create a Spring Boot project
2. Configure application.properties
    2b. Secret application.properties
3. Create a model class (Entity)
4. Create a repository
5. Create a controller
6. Wire repository to controller
7. Start server
    When you make a change to the code...
    To stop the server...
8. Automatically create database tables (optional)

1. Create a Spring Boot project

To create a Spring Boot project in VS Code, open the command palette (CTRL+SHIFT+P on Windows or start typing with a greater than sign ">" in the search bar). Then search for and select "Java: Create Java Project..."

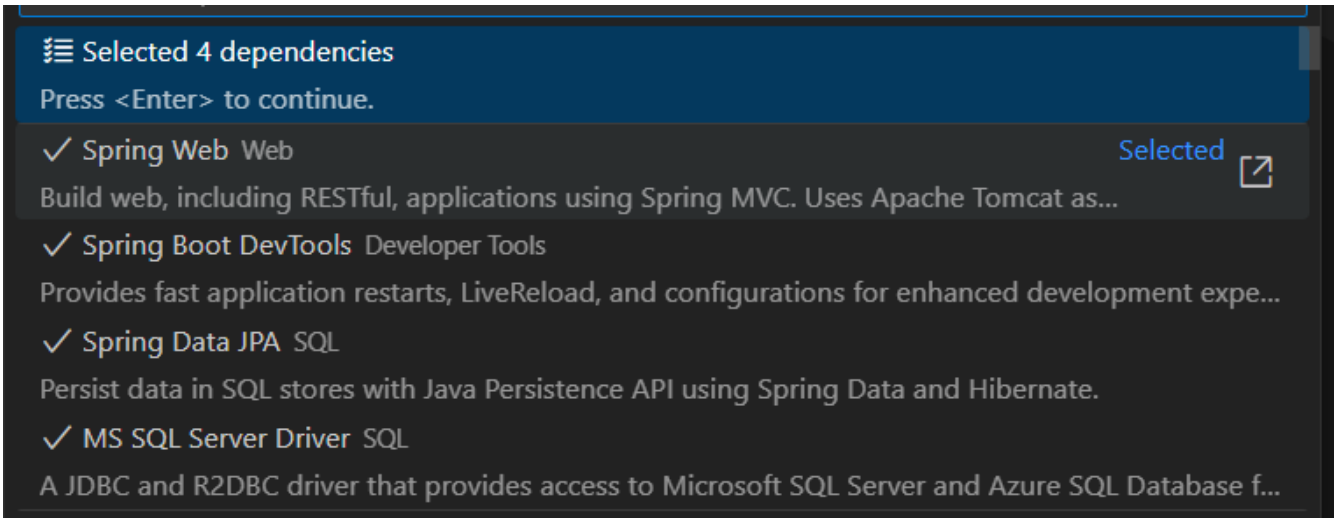


- Then select **Spring Boot** as the project type.



- Next select **Gradle Project**.
- Select the latest Spring Boot version, but NOT a SNAPSHOT version.

- Choose **Java** for the programming language.
- Enter a Group Id, e.g. **co.grandcircus**
- Enter an Artifact Id, e.g. blah-blah-api
- Packaging type: **Jar**
- Java version: Pick the latest Java Version that is at or below what you installed.
- Add four dependencies: **Spring *Web***, **Spring Boot *Dev*Tools**, **Spring Data *JPA***, and ***MS SQL* Server Driver**



- Select the parent folder where you want the project to be created. A new folder with the name of your Artifact Id will be created *inside* this folder.
- Open the project.

2. Configure application.properties

General database settings go here.

- Open **src/main/resources/application.properties**.
- Add the following. (HINT: Make sure there are no spaces at the beginning or end of the lines.)

```
spring.datasource.driverClassName=com.microsoft.sqlserver.jdbc.SQLServerDriver
# This optional property indicates whether hibernate automatically creates schema
tables for us
# Set it to create or update, start the app, then set it back to none.
spring.jpa.hibernate.ddl-auto=none
```

2b. Secret application.properties

Database connection info, password, and other secrets go here. This file should not be committed to GitHub.

- Inside **src/main/resources**, create a new Folder called **config**.
- Inside the config folder create a new File called **application.properties**.
- Add your database connection info here. For example:
    - Change the schema (`db_demos`) if need be.
    - Change the password to your SQL Server sa password.

```
# Database Connection
spring.datasource.url=jdbc:sqlserver://localhost;encrypt=true;trustServerCertificate=true;databaseName=db_demos
spring.datasource.username=sa
spring.datasource.password=password
```

- Open **.gitignore**. You may need to open the Navigator view in Eclipse to see this hidden file.
- At the bottom of .gitignore, add this line:
  src/main/resources/config/application.properties

3. Create a model class (Entity)

Create one for each database table.

WARNING: For this step, you must create your models in the main package or in a subpackage of the main package of your app.

- In src/main/java, inside your package, create a new Java class.
- Add the **@Entity** annotation before the class: (All of these annotations are in the **jakarta.persistence** package.)
- Most tables have an auto-incremented identity column. Add an id field with **@Id** and **@GeneratedValue** annotations. (see example below)
- Add getters and setters for all fields.
- Optionally, include a full constructor with all members and a default constructor with no params.
- Optionally, add toString.

Example:

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

@Entity
public class Flower {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;
	private String name;

	public Flower() {
	}
	public Flower(Long id, String name) {
		this.id = id;
		this.name = name;
	}

	public Long getId() {
		return id;
	}
	public void setId(Long id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

4. Create a repository

WARNING: For this step, you must create your Repository in the main package or in a subpackage of the main package of your app.

- In **src/main/java**, inside your package, create a new Java interface.
- Name it **<Something>Repository**

- It must extend **JpaRepository** using generic parameters for the entity type and the entity's ID type. For example, my entity is Flower and it has an ID of type Long...
- import org.springframework.data.jpa.repository.JpaRepository;

```
import org.springframework.data.jpa.repository.JpaRepository;
public interface FlowerRepository extends JpaRepository<Flower, Long> {

}
```

- Leave the body of the interface empty for now. It inherits all the methods you need to start; these methods are created automatically by the JPA (which is not typical for interfaces -- normally you create the implementations of the interfaces).

5. Create a controller

- In **src/main/java**, inside your package, create a new Java file.
- Name it **<Something>ApiController**.
- Annotate the class with **@RestController**
- Add a method that returns a String.
- Annotate the method with **@GetMapping("/some-url-path")**, using the URL path you want for this endpoint. ("/") for the root.
    - For different endpoints, use @GetMapping, @PostMapping, @PutMapping, or @DeleteMapping.
- Return an object, List, or array.
- Add the required imports
    - import org.springframework.stereotype.RestController;
    - import org.springframework.web.bind.annotation.GetMapping;

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class FlowerApiController {

     @GetMapping("/")
     public List<Flower> readAll() {
          return null;
     }
```

```
}
```

6. Wire repository to controller

- In your Controller, add a private field with the type of your Repository. Annotate this field with **@Autowired**.
- Within your controller methods, use this field to access the methods on your repository. You must use the field (camelCase), NOT the interface name (TitleCase) within your methods. In the example below, note flowerRepo.findAll() not FlowerRepository.findAll().
- Add the following imports
    - import org.springframework.beans.factory.annotation.Autowired;
    - import org.springframework.ui.Model;

For example:

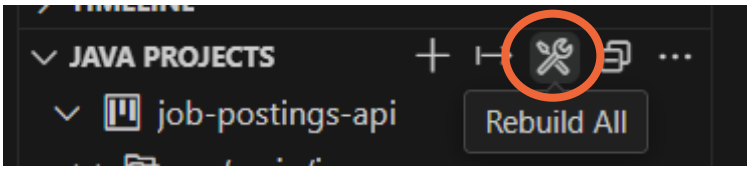
```
    @Autowired
    private FlowerRepository flowerRepo;

    @GetMapping("/flowers")
    public List<Flower> readAll() {
        return flowerRepo.findAll();
    }
```

7. Start server

- Select the **___Application.java** file.
- Right-click. Select Run Java.
- Visit http://localhost:8080 in your browser.

When you make a change to the code...

After making changes and saving your files, click the Rebuild All button in the Java Projects section. The API server will automatically restart and apply your changes.



To stop the server...

Click into the terminal within VS Code where the application is running. Press CTRL+C to stop the server.

8. Automatically create database tables (optional)

After you have created and annotated your models, Hibernate can automatically create matching database tables for you.

- In src/main/resources/**application.properties**, change **spring.jpa.hibernate.ddl-auto** to **update**.
- If your app is not already running, start it. (If it was already running it would automatically restart.) When it starts up, it will create the tables it needs. You can check, if you want, by going to MySQLWorkbench and refreshing the tables view.
- Go back to src/main/resources/**application.properties** and set **spring.jpa.hibernate.ddl-auto** back to **none**.
- If models change later, repeat this process. Switch ddl-auto to update, restart, then set it back to none.