

# Spring Data JPA for a RESTful API in Docker



# our goal

- build fast and easily a RESTful API using a data base supporting CRUD like
- POST                      /api/products                      create a new product
- GET                        /api/products                      get all products
- GET                        /api/products/:id                  get a product by its :id
- PUT                        /api/products                      update a product
- DELETE                    /api/products/:id                  delete a products by its :id

# Create a project

New Project

**Spring Initializr Project Settings**

Group:

Artifact:

Type: ☒ Maven ☐ Gradle

Language: ☒ Java ☐ Kotlin ☐ Groovy

Packaging: ☒ Jar ☐ War

Java Version:

Version:

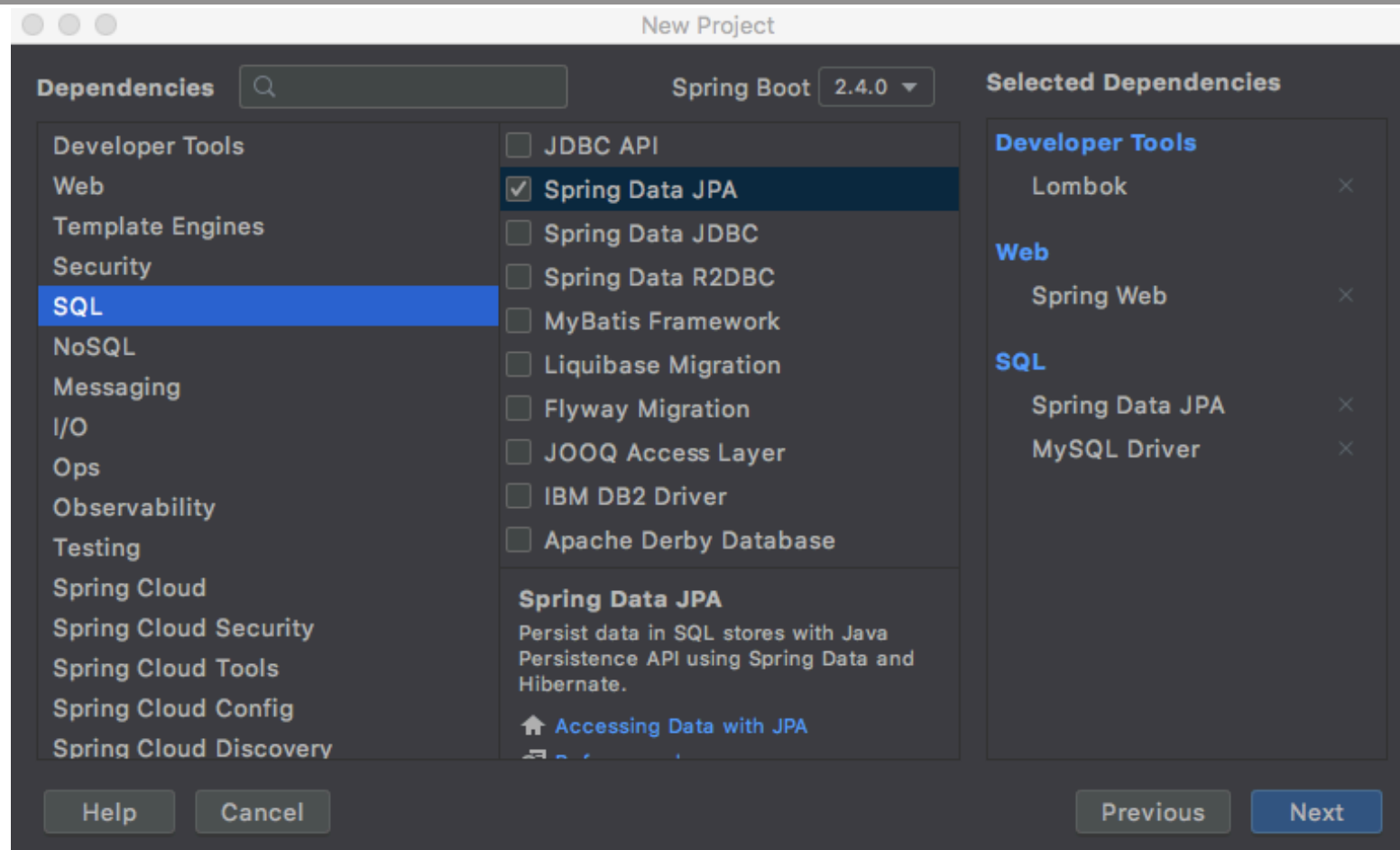
Name:

Description:

Package:



# set dependencies



# and paths


New Project


Project name:

Project location:  ...

▼ More Settings

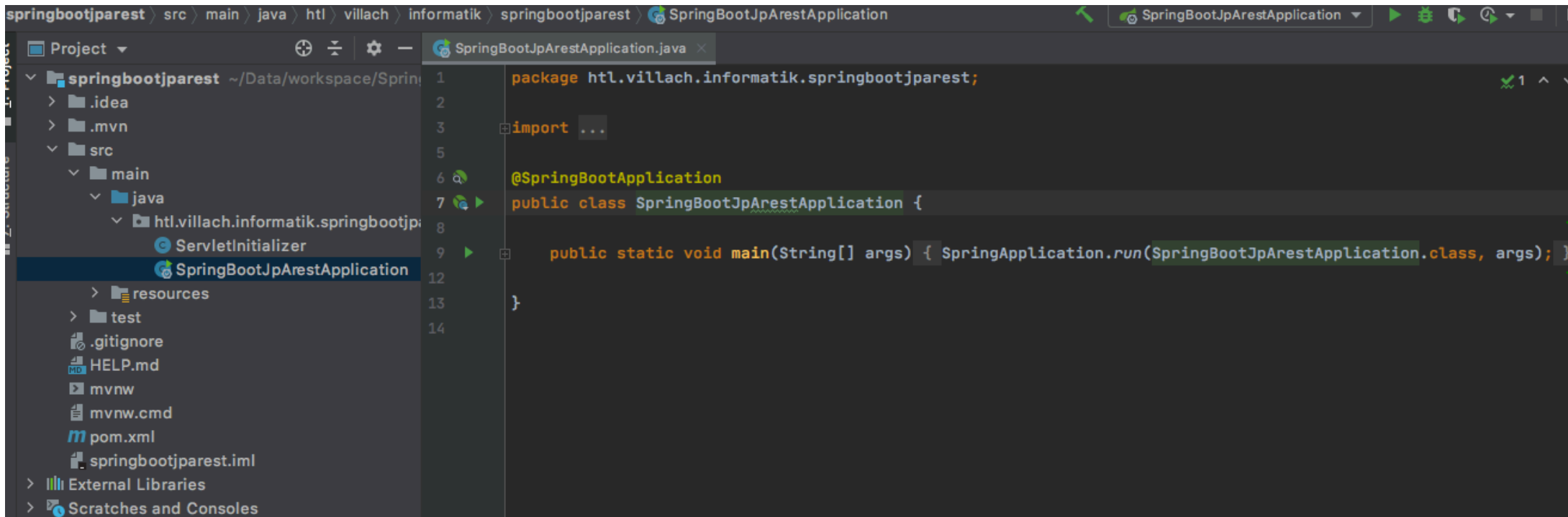
Module name:

Content root:  

Module file location:  

Project format:  ▼

et voila

A screenshot of an IDE window showing the file 'SpringBootJpArestApplication.java'. The left sidebar shows the project structure with 'SpringBootJpArestApplication' selected. The main editor area shows the following code:

```
1 package htl.villach.informatik.springbootjparest;  
2  
3 import ...  
4  
5  
6 @SpringBootApplication  
7 public class SpringBootJpArestApplication {  
8  
9     public static void main(String[] args) { SpringApplication.run(SpringBootJpArestApplication.class, args); }  
10  
11  
12 }  
13  
14
```

# optionally, add a hello controller

```
SpringBootJpArestApplication.java x
1  package htl.villach.informatik.springbootjparest;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RequestParam;
7  import org.springframework.web.bind.annotation.RestController;
8
9  @SpringBootApplication
10 @RestController
11 public class SpringBootJpArestApplication {
12
13     public static void main(String[] args) { SpringApplication.run(SpringBootJpArestApplication.class, args); }
14
15
16     @GetMapping("/api/v1/hello")
17     public String hello(@RequestParam(value = "name", defaultValue = "World") String name) {
18         return String.format("Hello %s!", name);
19     }
20 }
21 }
```

software  
inside

and a start will fail due to a missing data base

```
Run: SpringBootJpArestApplication x
▶ Console ▶ Endpoints
Error starting ApplicationContext. To display the conditions report re-run your application with 'debug' enabled.
2020-11-11 10:10:10.100 ERROR 25240 --- [main] o.s.b.d.LoggingFailureAnalysisReporter :

*****
APPLICATION FAILED TO START
*****

Description:

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

Reason: Failed to determine a suitable driver class
```



# define its application properties

```
#Spring datasource
spring.datasource.url=jdbc:mysql://localhost:3306/db
spring.datasource.username=usr
spring.datasource.password=pwd
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

## Hibernate Properties: a SQL dialect makes Hibernate generate better SQL for a chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

💡
#set jpa true = we can see any in console
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=none
spring.jpa.generate-ddl=true

server.port=8080
```

# With a data base, it works

■ Henne-Ei-Problem? zuerst DB, dann App, oder umgekehrt?

The screenshot shows a Spring Boot application running in a Docker container. The application is displaying a "Whitelabel Error Page" with the message: "This application has no explicit mapping for /error, so you are seeing this as a fallback. Fri Nov 27 18:58:20 CET 2020 There was an unexpected error (type=Not Found, status=404)." The application is running on localhost:8080. The Docker Desktop interface shows the container "products\_webservice" running on port 3306, with a "ProductsDB mysql" database running on port 3306.

Run: SpringBootJpArestApplication x

Console Endpoints

localhost:8080

## Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Nov 27 18:58:20 CET 2020  
There was an unexpected error (type=Not Found, status=404).

localhost:8080/api/v1/hello

Hello World!

localhost:8080/api/v1/hello?name=Sepp

Hello Sepp!

Containers / Apps

Images

products\_webservice  
OTHER

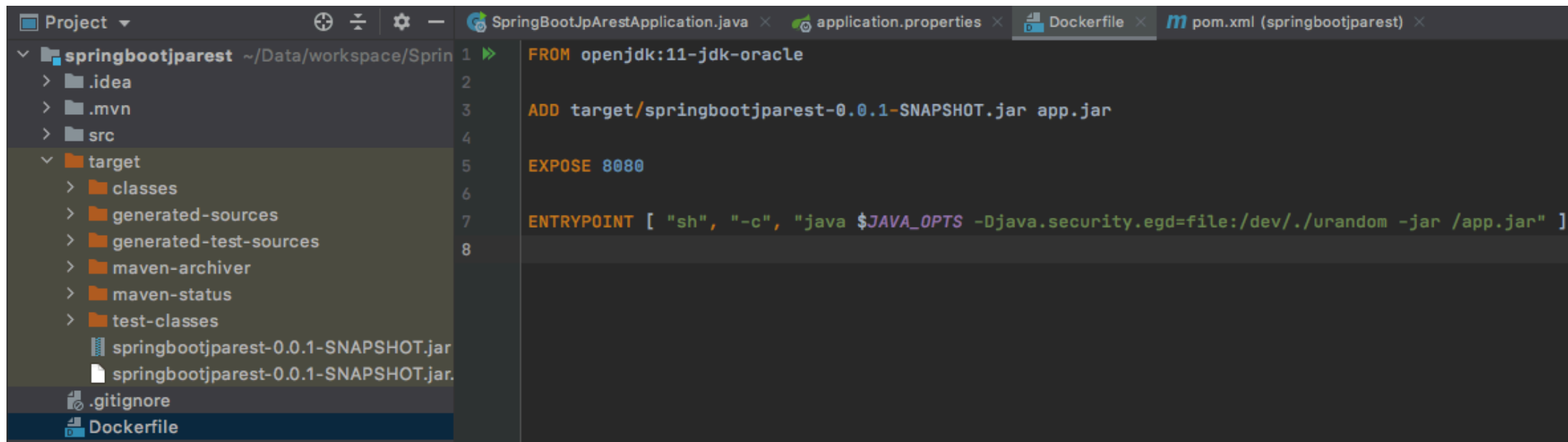
ProductsDB mysql  
RUNNING PORT: 3306

software inside

# build your application by maven

```
ChEMc:springbootjparest ChES mvn package -DskipTests=true
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ht1.villach.informatik:springbootjparest >-----
[INFO] Building SpringBootJPAREst 0.0.1-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:resources (default-resources) @ springbootjparest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] Copying 1 resource
[INFO] Copying 0 resource
[INFO] The encoding used to copy filtered properties files have not been set. This means that the same encoding will be used to copy filtered properties files as when copy
tered resources. This might not be what you want! Run your build with --debug to see which files might be affected. Read more at https://maven.apache.org/plugins/maven-res
/examples/filtering-properties-files.html
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ springbootjparest ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.2.0:testResources (default-testResources) @ springbootjparest ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Using 'UTF-8' encoding to copy filtered properties files.
[INFO] skip non existing resourceDirectory /Users/ChE/Data/workspace/Spring/SpringBootJPAREst.v4/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ springbootjparest ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ springbootjparest ---
[INFO] Tests are skipped.
[INFO]
[INFO] --- maven-war-plugin:3.3.1:war (default-war) @ springbootjparest ---
[INFO] Packaging webapp
[INFO] Assembling webapp [springbootjparest] in [/Users/ChE/Data/workspace/Spring/SpringBootJPAREst.v4/target/springbootjparest-0.0.1-SNAPSHOT]
[INFO] Processing war project
[INFO] Building war: /Users/ChE/Data/workspace/Spring/SpringBootJPAREst.v4/target/springbootjparest-0.0.1-SNAPSHOT.war
[INFO]
[INFO] --- spring-boot-maven-plugin:2.4.0:repackage (repackage) @ springbootjparest ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] BUILD SUCCESS
```

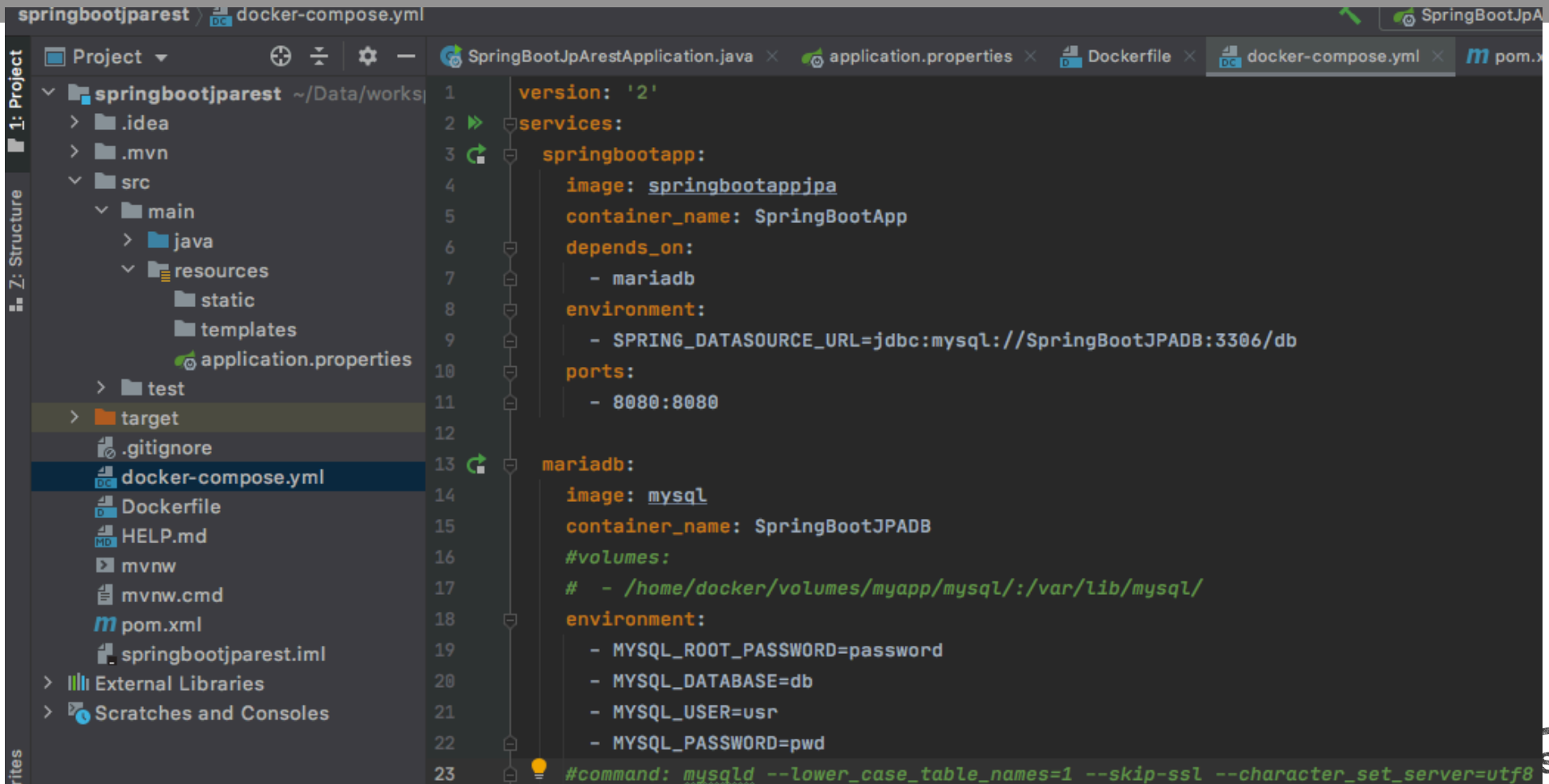
# Create a Dockerfile



The screenshot shows an IDE with a project named 'springbootjparest' located at '~/.Data/workspace/Sprin'. The project structure includes folders for '.idea', '.mvn', 'src', and 'target'. The 'target' folder contains subfolders for 'classes', 'generated-sources', 'generated-test-sources', 'maven-archiver', 'maven-status', and 'test-classes', along with two JAR files: 'springbootjparest-0.0.1-SNAPSHOT.jar'. A 'Dockerfile' is also present in the project. The Dockerfile content is as follows:

```
1 FROM openjdk:11-jdk-oracle
2
3 ADD target/springbootjparest-0.0.1-SNAPSHOT.jar app.jar
4
5 EXPOSE 8080
6
7 ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
8
```

# Create a docker-compose.yml

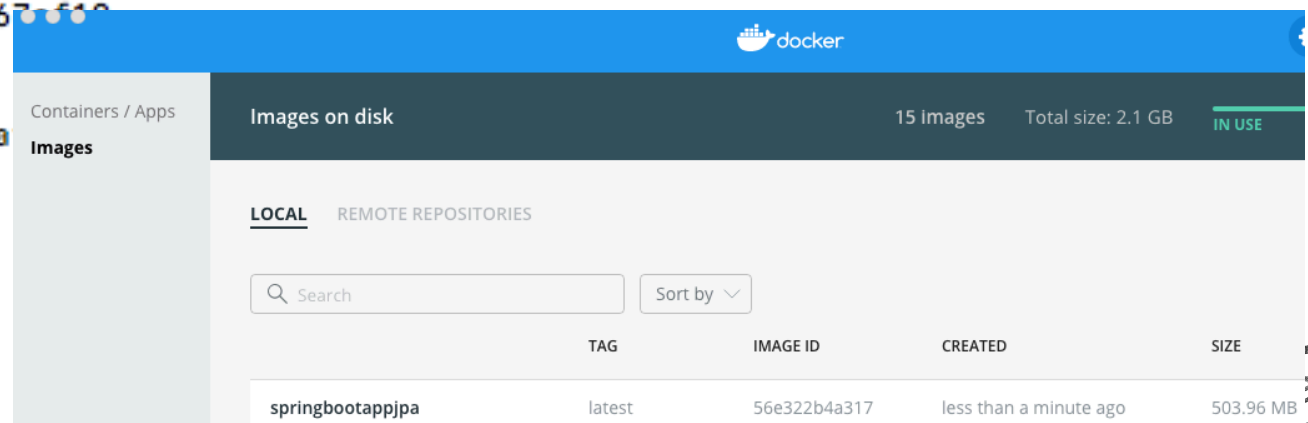


The screenshot shows an IDE with a project named 'springbootjparest'. The left sidebar displays the project structure, including folders like '.idea', '.mvn', 'src', 'main', 'java', 'resources', 'static', 'templates', 'application.properties', 'test', 'target', and files like '.gitignore', 'docker-compose.yml', 'Dockerfile', 'HELP.md', 'mvnw', 'mvnw.cmd', 'pom.xml', and 'springbootjparest.iml'. The main editor area shows the content of 'docker-compose.yml'.

```
1 version: '2'
2
3 services:
4   springbootapp:
5     image: springbootappjpa
6     container_name: SpringBootApplication
7     depends_on:
8       - mariadb
9     environment:
10      - SPRING_DATASOURCE_URL=jdbc:mysql://SpringBootJPADB:3306/db
11     ports:
12      - 8080:8080
13
14   mariadb:
15     image: mysql
16     container_name: SpringBootJPADB
17     #volumes:
18     # - /home/docker/volumes/myapp/mysql:/var/lib/mysql/
19     environment:
20      - MYSQL_ROOT_PASSWORD=password
21      - MYSQL_DATABASE=db
22      - MYSQL_USER=usr
23      - MYSQL_PASSWORD=pwd
24     #command: mysqld --lower_case_table_names=1 --skip-ssl --character_set_server=utf8
```

# Build your docker image


```
ChEMc:springbootjparest ChES$ docker build -t springbootappjpa .
Sending build context to Docker daemon 40.69MB
Step 1/4 : FROM openjdk:11-jdk-oracle
----> 680c5ad86000
Step 2/4 : ADD target/springbootjparest-0.0.1-SNAPSHOT.jar app.jar
----> 0c291bd47741
Step 3/4 : EXPOSE 8080
----> Running in 340fb01aa387
Removing intermediate container 340fb01aa387
----> c64148e3853d
Step 4/4 : ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
----> Running in 68df6ee67af19
Removing intermediate container 68df6ee67af19
----> 56e322b4a317
Successfully built 56e322b4a317
Successfully tagged springbootappjpa:latest
```



# start your docker composition


```
ChEMc:springbootjparest ChES$ docker-compose -f docker-compose.yml up -d springbootapp
Recreating SpringBootJPADB ... done
Creating SpringBootApplication ... done
```

		TAG	IMAGE ID	CREATED	SIZE
springbootappjpa	IN USE	latest	56e322b4a317	15 minutes ago	503.96 MB
mariadb				ago	406.51 MB
tomcat	IN USE			s ago	648.66 MB
mysql	IN USE			s ago	545.31 MB




springbootjparest  
RUNNING

---



SpringBootDB [mysql](#)  
RUNNING PORT: 3306

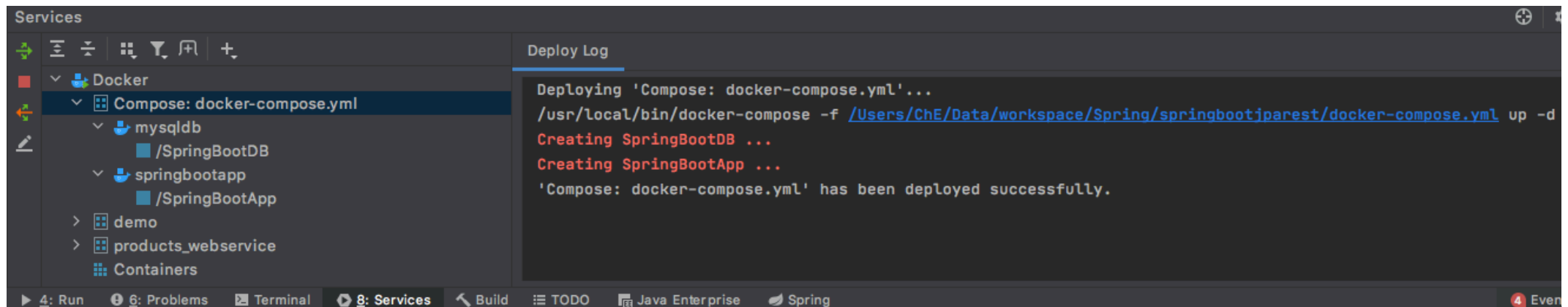
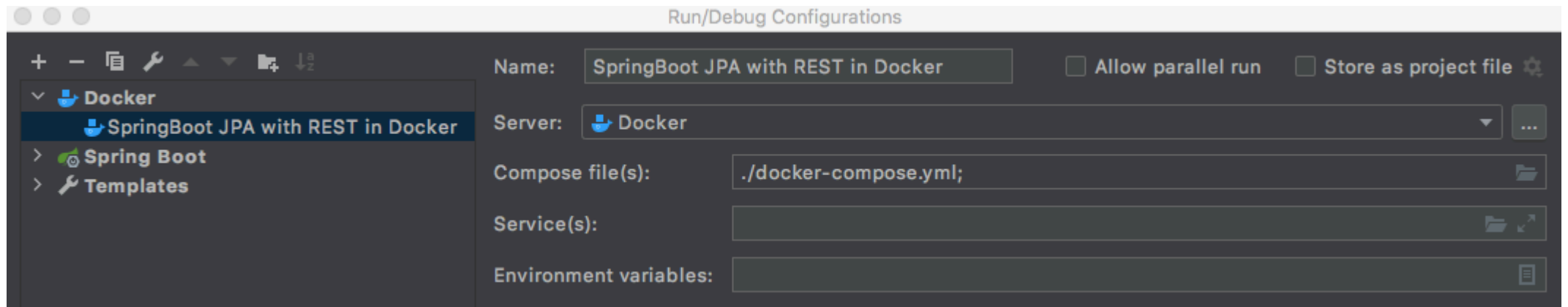
---



SpringBootApplication [springbootappjpa](#)  
RUNNING PORT: 8080

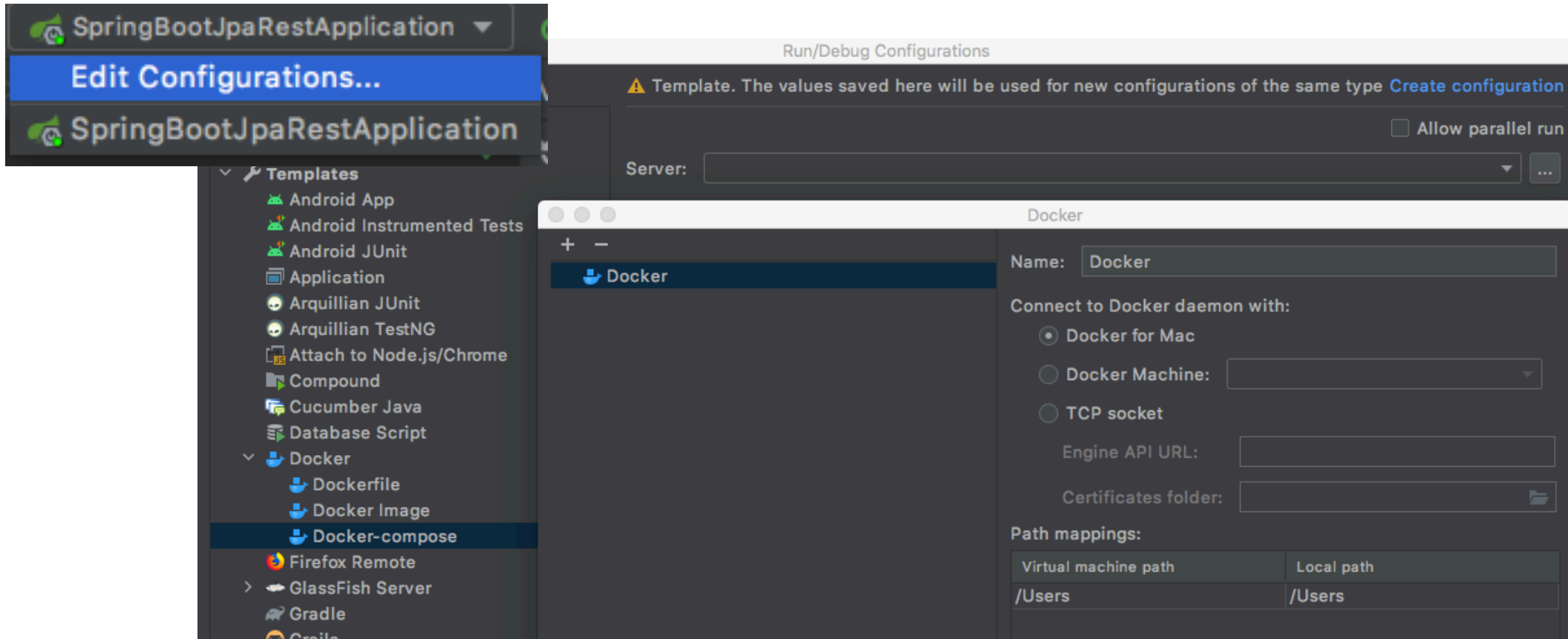


# or in IntelliJ





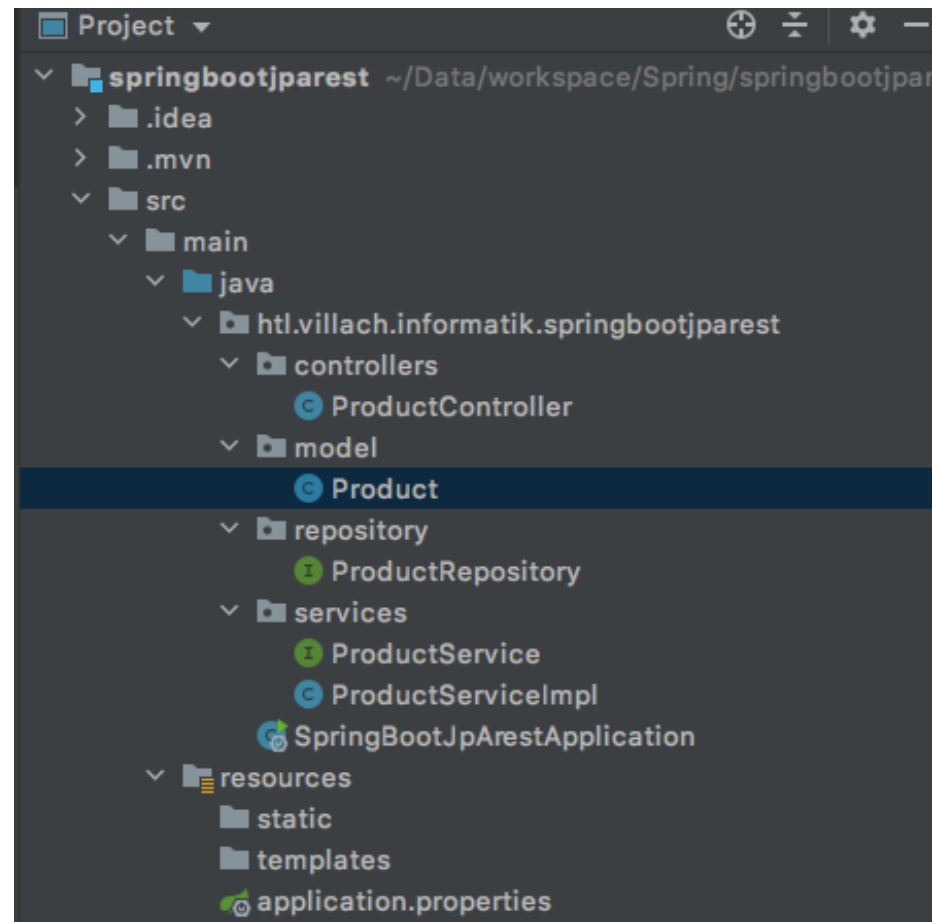
# by Edit and Create configuration



The screenshot shows the 'Run/Debug Configurations' dialog in IntelliJ IDEA. On the left, a list of templates is shown, with 'Docker-compose' selected under the 'Docker' category. The 'Edit Configurations...' button is highlighted. The main panel shows the 'Docker' configuration template. The 'Name' field is set to 'Docker'. Under 'Connect to Docker daemon with:', 'Docker for Mac' is selected. The 'Engine API URL' and 'Certificates folder' fields are empty. The 'Path mappings' table shows a mapping from '/Users' to '/Users'.

Virtual machine path	Local path
/Users	/Users

# Establish Spring Boot MVC structure



# Add the model(s)

```
package htl.villach.informatik.springbootjparest.model;

import lombok.Getter;
import lombok.Setter;

import javax.persistence.*;
import java.io.Serializable;

@Entity
@Table(name = "products")
public class Product implements Serializable {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Getter @Setter private Long id;

    @Getter @Setter private String productName;
    @Getter @Setter private String price;
}
```

software  
inside

# Enable its JPA repository on products by id

```
package htl.villach.informatik.springbootjparest.repository;

import htl.villach.informatik.springbootjparest.model.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

}
```

# Define a service interface

```
package htl.villach.informatik.springbootjparest.services;

import htl.villach.informatik.springbootjparest.model.Product;

import java.util.List;

public interface ProductService {
    public List<Product> getAllProduct();
    public Product      getOneProduct(Long id);
    public Product      createProduct(Product product);
    public boolean       deleteProduct(Long id);
}
```

# implement it by using your repository

```
package htl.villach.informatik.springbootjparest.services;

import ...

@Service
public class ProductServiceImpl implements ProductService{

    @Autowired
    @Getter @Setter private ProductRepository repo;

    public List<Product> getAllProduct() { return repo.findAll(); }

    public Product getOneProduct(Long id) { return repo.findById(id).get(); }

    public Product createProduct(Product product) { return repo.save(product); }

    public boolean deleteProduct(Long id) {
        repo.deleteById(id);
        return true;
    }
}
```

software  
inside

and your controller by using your service

```
package htl.villach.informatik.springbootjparest.controllers;

import ...

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

    @Autowired
    private ProductServiceImpl svc;

    @GetMapping("/")
    public List<Product> getAllProduct() { return svc.getAllProduct(); }

    @GetMapping("/{id}")
    public Product getProductById(@PathVariable Long id) { return svc.getOneProduct(id); }

    @PostMapping("/")
    public Product createProduct(@RequestBody Product Product) { return svc.createProduct(Product); }

    @PutMapping("/")
    public Product updateProduct(@RequestBody Product Product) { return svc.createProduct(Product); }

    @DeleteMapping("/{id}")
    public void deleteProduct(@PathVariable Long id) { svc.deleteProduct(id); }
}
```

