ASSIGNMENT – 10

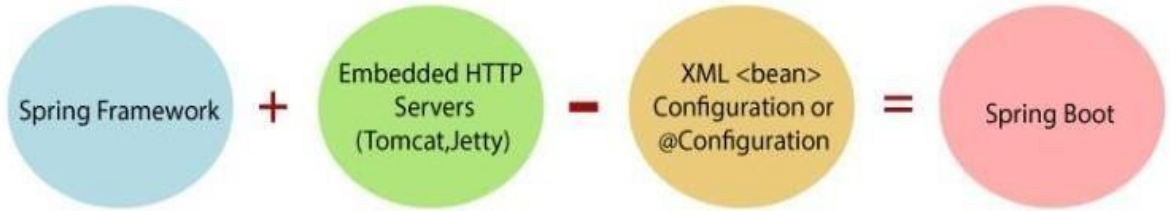
Aim: Spring Boot and Restful Web Services

Theory:

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".
Spring Boot is a Spring module that provides the RAD (Rapid ApplicationDevelopment) feature to the Spring framework.
Our Spring Boot Tutorial includes all topics of Spring Boot such, as features,project, maven project, starter project wizard, Spring Initializr, CLI, applications, annotations, dependency management, properties, starters, Actuator, JPA, JDBC, etc.

Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

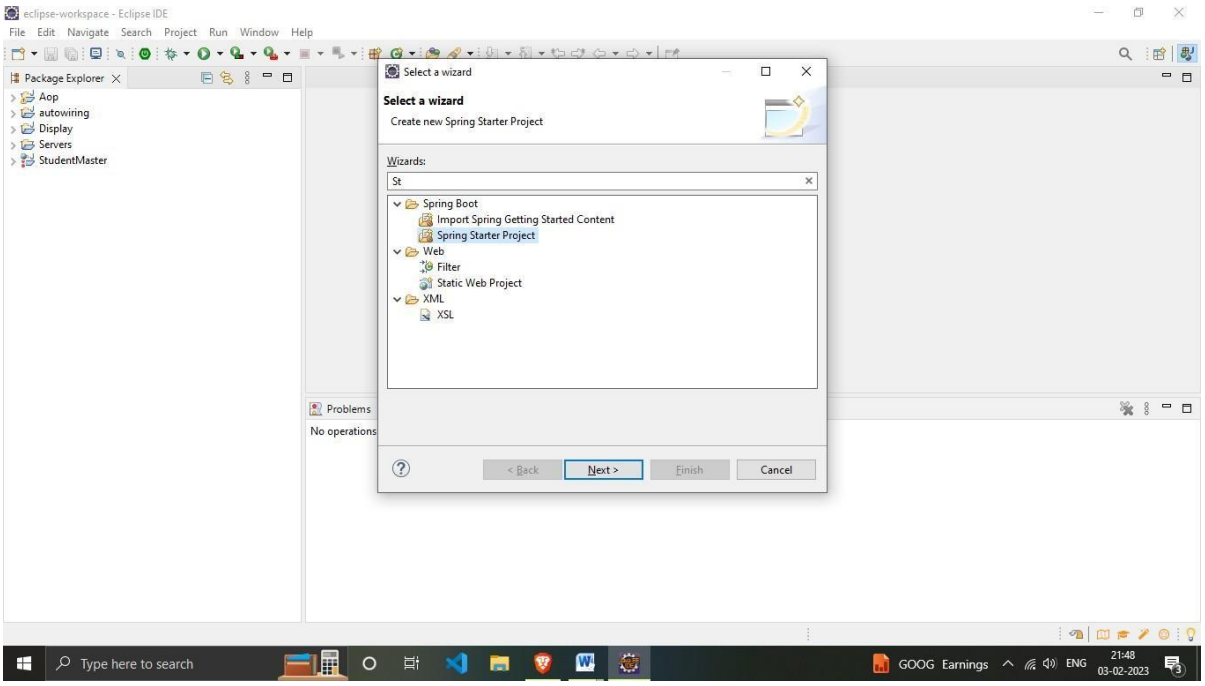


Questions:

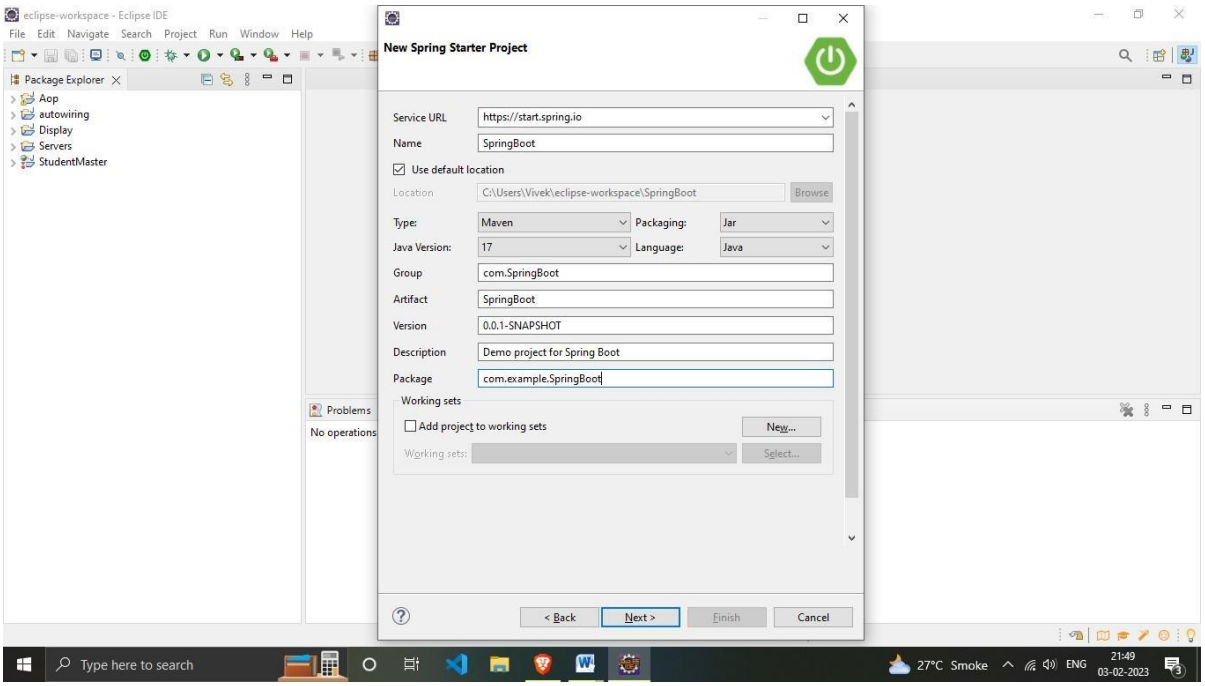
1. Write a program to create a simple Spring Boot application that prints a message.

Before starting, please make sure to have Spring Tools 4 installed on thesystem.

Step 1: In Eclipse, Go to File -> New -> Other and then select Spring Starterproject

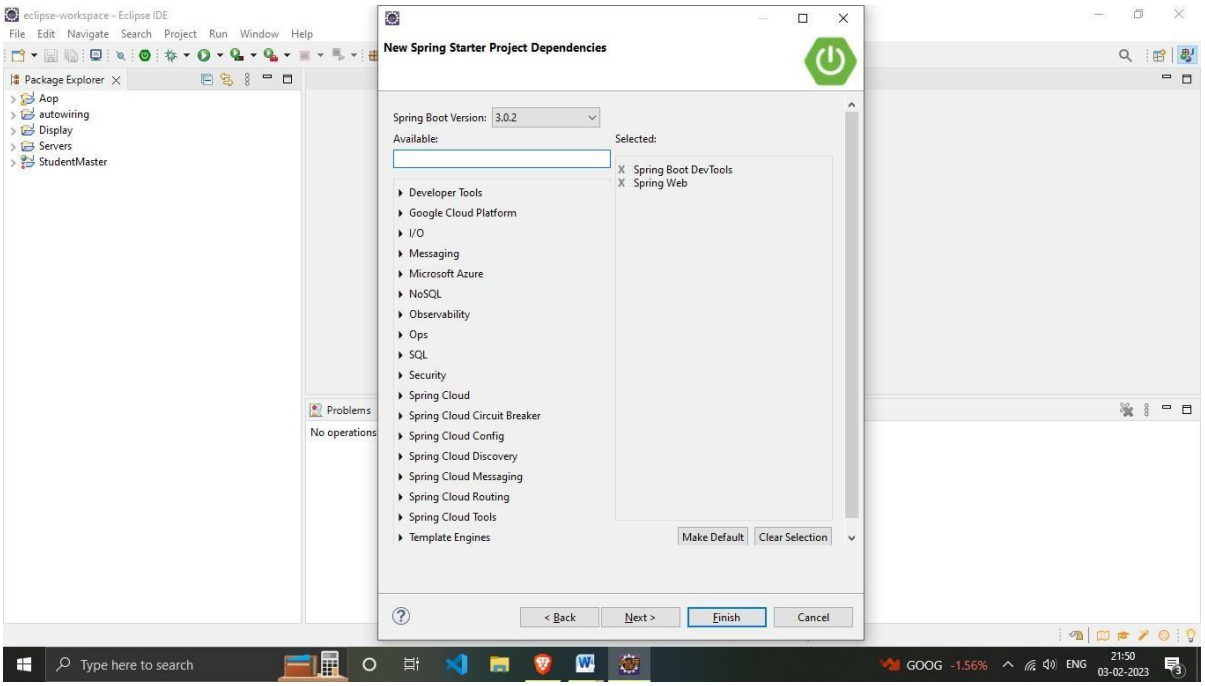


Step 2: Fill the project details to create the new project as shown below. Select Maven in type.

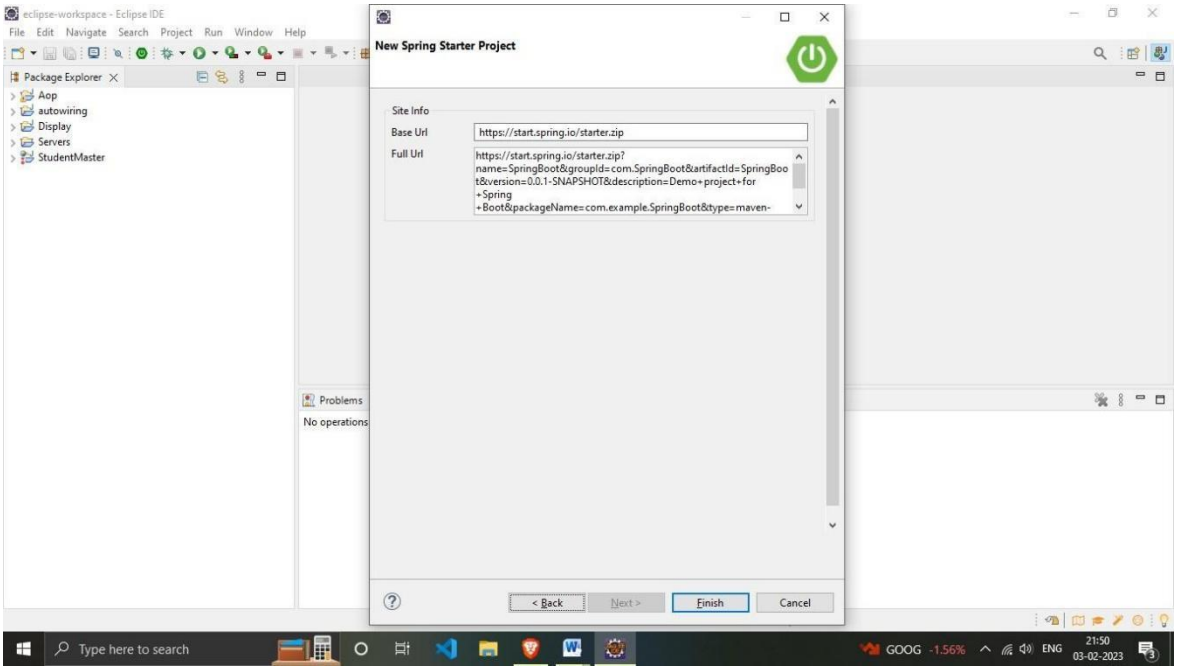


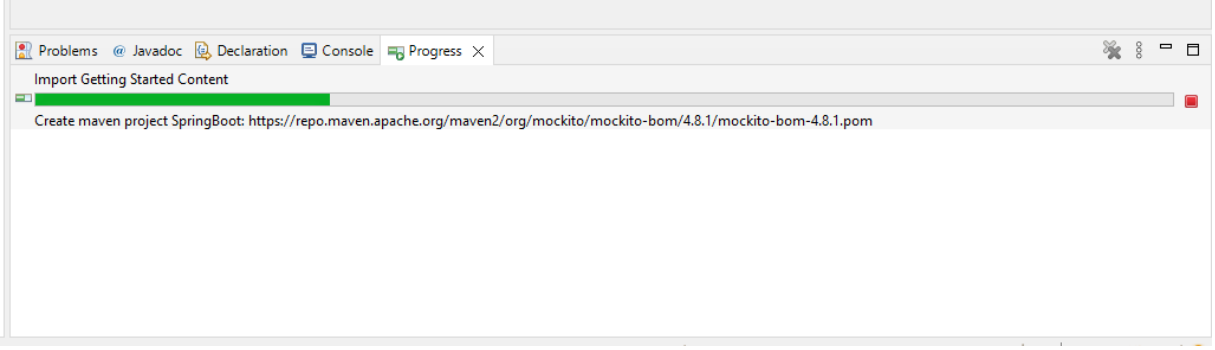
Step 3: On the next step, select the following dependencies from the list –

a. Spring Web
b. Spring Boot Dev Tools
c. Thymleaf

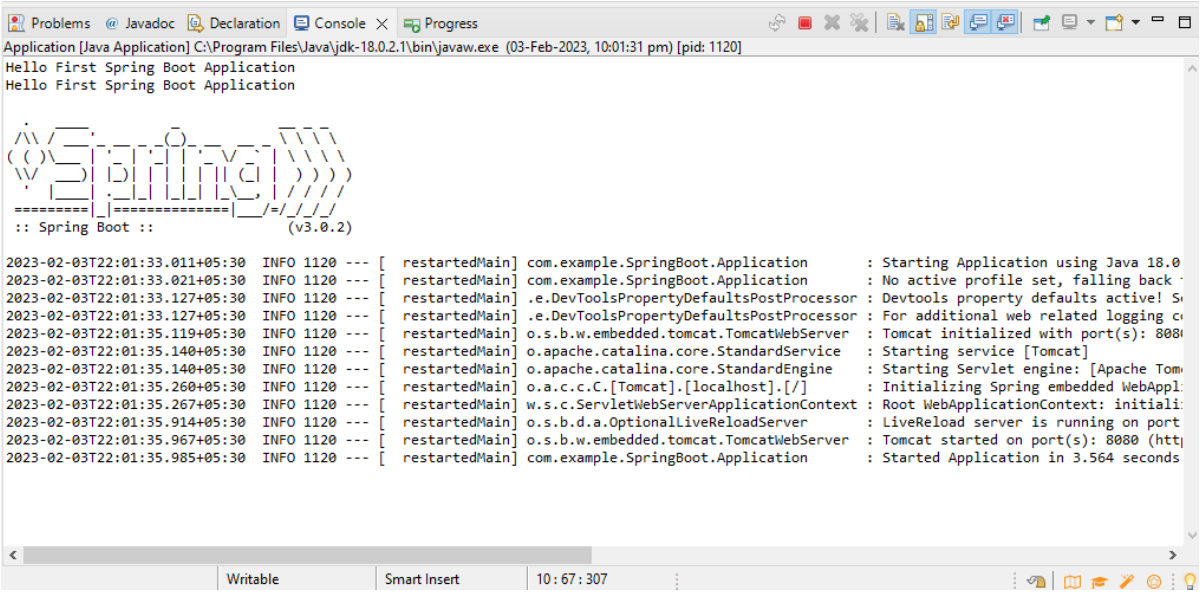


Step 4: Click on finish and wait for the project to be created.





Step 5: Go to src/main/java and go inside the package you created. Click on Application.java and run it. The output will be as shown below



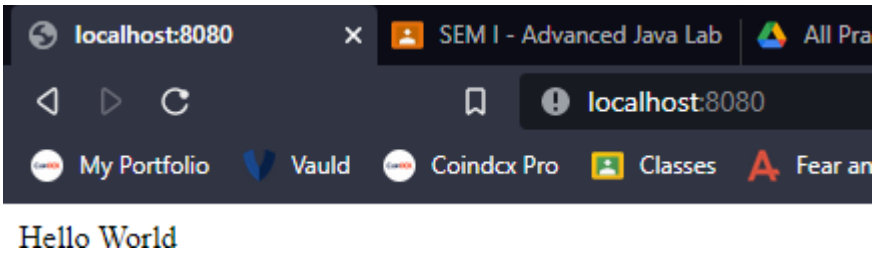
Step 6: Now make changes in the Application.java as given below-

Code:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController; import
org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
@RestController
public class Application {
public static void main(String[] args) {
SpringApplication.run(Application.class,
args);
}
```

```
@RequestMapping(value = "/")
public String hello() {
return "Hello World";
}
}
```

Output:



2. **Write a program to demonstrate RESTful Web Services with spring boot.**

Follow the same steps as in previous question to create a new Spring Boot Project.

Step 1: Go to src/main/java and write the following and create a file named “Product.java” which will act as the POJO class.

Product.java:

```
package
com.example.SpringBootApp;public
class Product {
private String id;
private String name;
public String getId()
{return id;
}
public void setId(String id)
{this.id = id;
}
public String getName()
{return name;
```

```
}
public void setName(String name)
{this.name = name;
}
}
```

Step 2: Create another java class named “ProductController.java” inside the same package.

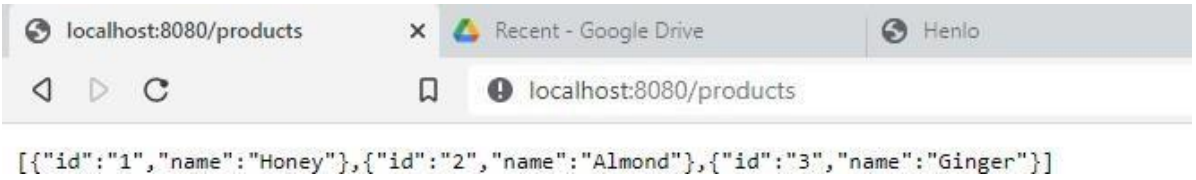
ProductController.java:

```
package
com.example.SpringBootApp;import
java.util.HashMap;
import java.util.Map;
import com.example.SpringBootApp.Product;
import org.springframework.http.HttpStatus; import
org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;
@RestController
public class ProductServiceController {
private static Map<String,Product> productRepo= new
HashMap<>();static {
Product honey = new Product();
honey.setId("1");
honey.setName("Honey");
productRepo.put(honey.getId(), honey);
Product almond = new Product();
almond.setId("2");
almond.setName("Almond");
productRepo.put(almond.getId(), almond);
}
@RequestMapping(value = "/products/{id}", method =
RequestMethod.DELETE)
public ResponseEntity<Object> delete(@PathVariable("id") String id) {
```

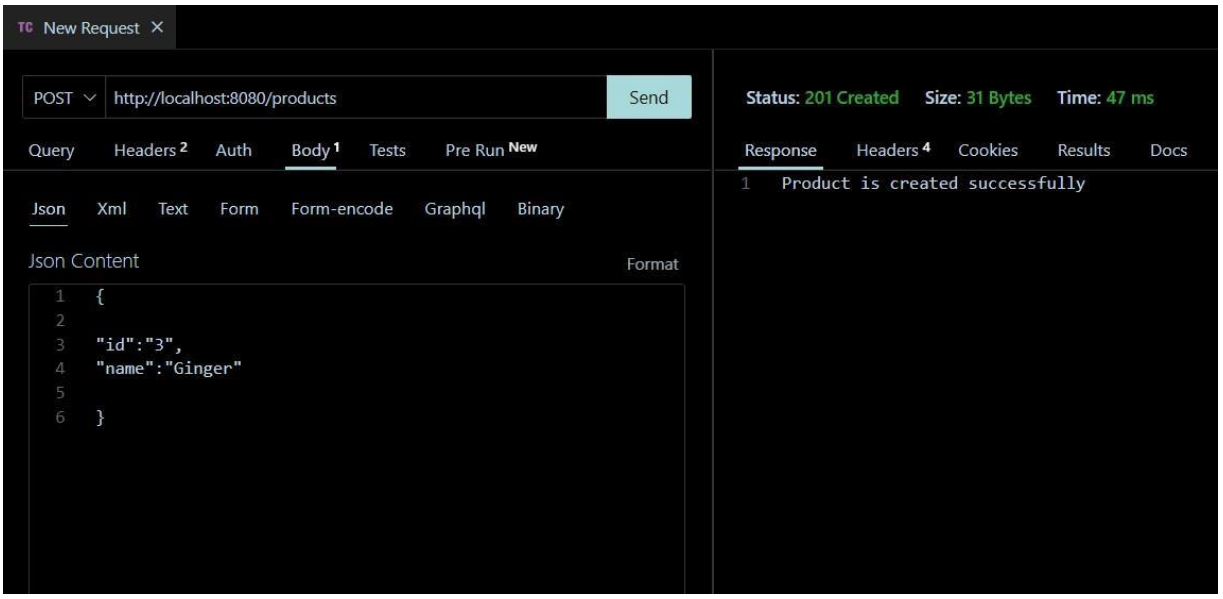
```
productRepo.remove(id);
return new ResponseEntity<>("Product is deleted successsfully",
HttpStatus.OK);
}
@RequestMapping(value = "/products/{id}", method = RequestMethod.PUT)
//update or create
public ResponseEntity<Object> updateProduct(@PathVariable("id") String
id,@RequestBody Product product)
{
product.setId(id);
productRepo.put(id, product);
return new ResponseEntity<>("Product is updated successsfully",
HttpStatus.OK);
}
@RequestMapping(value = "/products", method = RequestMethod.POST)
public ResponseEntity<Object> createProduct(@RequestBody Product
product) {
productRepo.put(product.getId(), product);
return new ResponseEntity<>("Product is created successfully",
HttpStatus.CREATED);
}
@RequestMapping(value = "/products")
public ResponseEntity<Object> getProduct()
{
return new ResponseEntity<>(productRepo.values(), HttpStatus.OK);
}
}
```

Step 3: After creating both the files, run the Spring boot application and go to use the api endpoints specified in the product controller to test GET,POST,PUTand DELETE operations using any Api testing application.

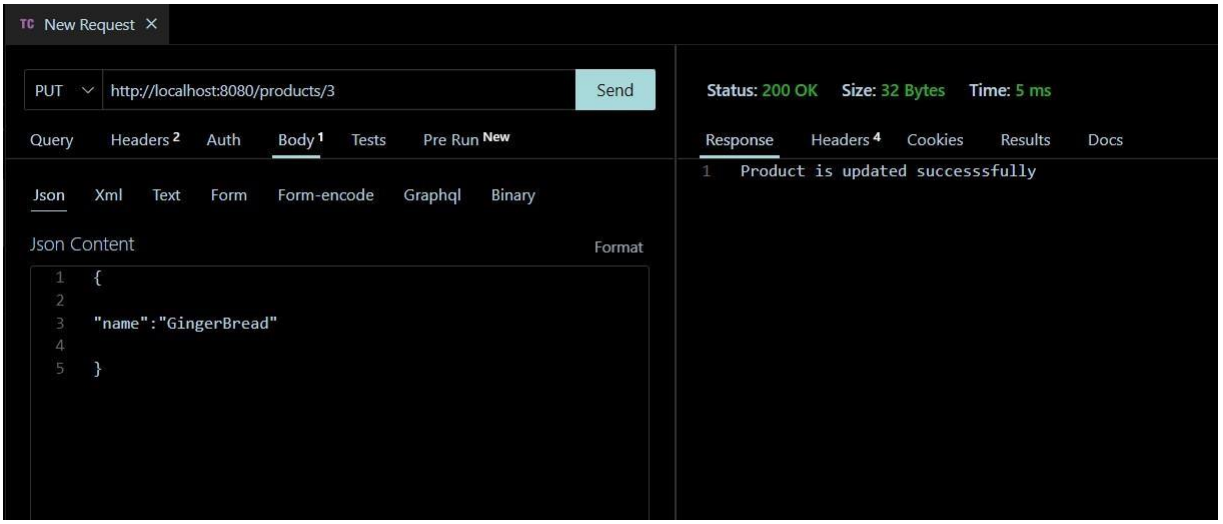
GET REQUEST:



POST REQUEST:



PUT REQUEST:



DELETE REQUEST: