Subtask 1

The environment that we need is all set. Let us now continue and expose some resources to the client. As we do not have a database set up yet, we work with static data.

To get a feeling of what a common REST architecture looks like we initially start with a basic products web layer and then work our way up from there.

To do so, create a folder [root].domain.product
HINT: If you followed your lecturers steps your root should be accessible by main.java.ch.noser3

In there you want to add a new class named ProductWeb.java. As learned in the previous chapter the web layers sole responsibility is to expose resources. This is achieved by endpoints. Each endpoint has its own URL through which it is accessed. This in combination with an appropriate http method such as get, post, put, delete.
HINT: A product can be retrieved by invoking the endpoint localhost:8080/products/{productId}. The adequate http method is GET. If you are not intending to access a single resource but rather all products the appropriate URL is called localhost:8080/products.

Before we return products however, we want to start trivially and give back a good old hello World. Not in a Resource but Service oriented manner:

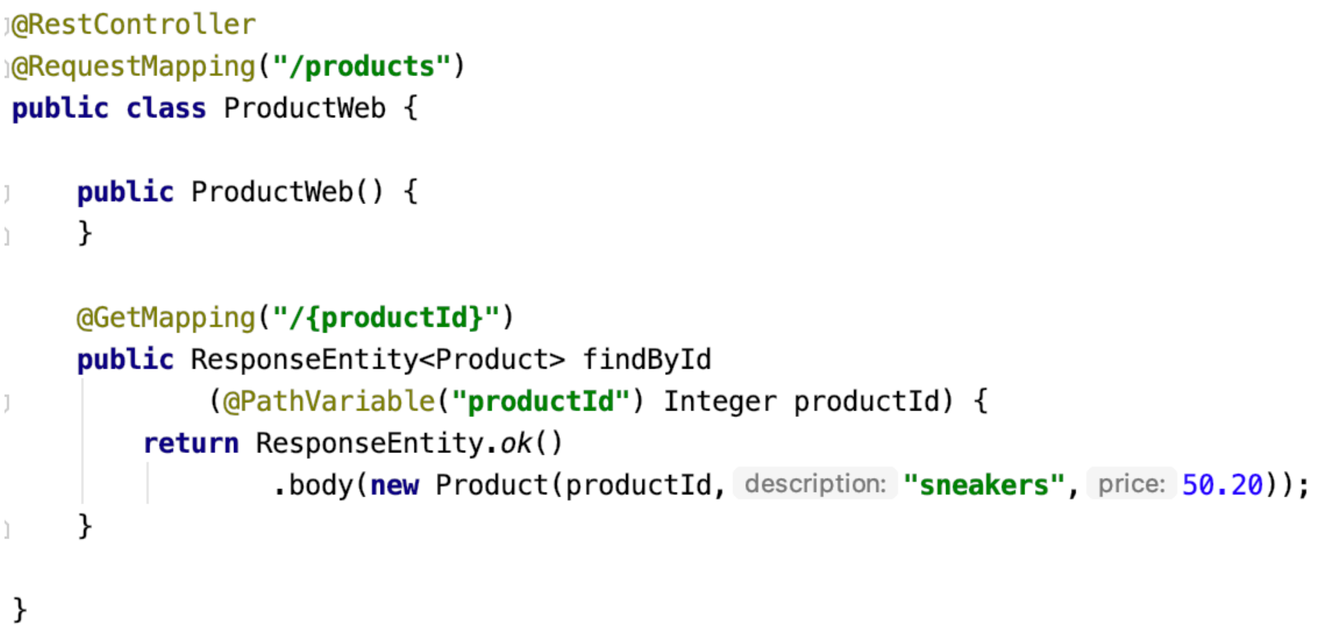
```java
@RestController
public class ProductWeb{

    public ProductWeb() {
    }

    @GetMapping("/helloWorld")
    public ResponseEntity<String> helloWorld() {
        return ResponseEntity.ok().body("helloWorld");
    }

}
```

Let's advance and give our hello World a bit more of a dynamic feeling:

```java
@RestController
public class ProductWeb {

    public ProductWeb() {
    }

    @GetMapping("/say/{requestedText}")
    public ResponseEntity<String> say
            (@PathVariable("requestedText") String requestedText) {
        return ResponseEntity.ok().body(requestedText);
    }

}
```

Subtask 2

Now it is time to forget all that hello World and pursue to our actual goal, which are products. Create a new class in the same folder called Product.java. These classes are basically data containers and hold no logic at all. They are also called business objects. As we haven't talked about Spring Data yet we want to leave out all hibernate related annotations for now and return products with static values only.



```java
@RestController
@RequestMapping("/products")
public class ProductWeb {

    public ProductWeb() {
    }

    @GetMapping("/{productId}")
    public ResponseEntity<Product> findById
            (@PathVariable("productId") Integer productId) {
        return ResponseEntity.ok()
                .body(new Product(productId, description: "sneakers", price: 50.20));
    }

}
```

HINT: Noticed the new annotation @RequestMapping? This is called a parent mapping and functions like a prefix. It applies to all endpoints within the class ProductWeb.java.

Subtask 3

To continue further on our route we not only want to look at http.GET but see what it is like to create, delete or update resources as well.
Time to download the latest version of Postman.
HINT: https://www.postman.com/downloads/

Once downloaded get in touch with your lecturer for a quick introduction of the tool in class.

Subtask 4

Now it is time for you to get your hands dirty. Implement a full web layer for the business object Product.java covering CRUD. This includes create, retrieve, update an delete and results in 5 endpoints.