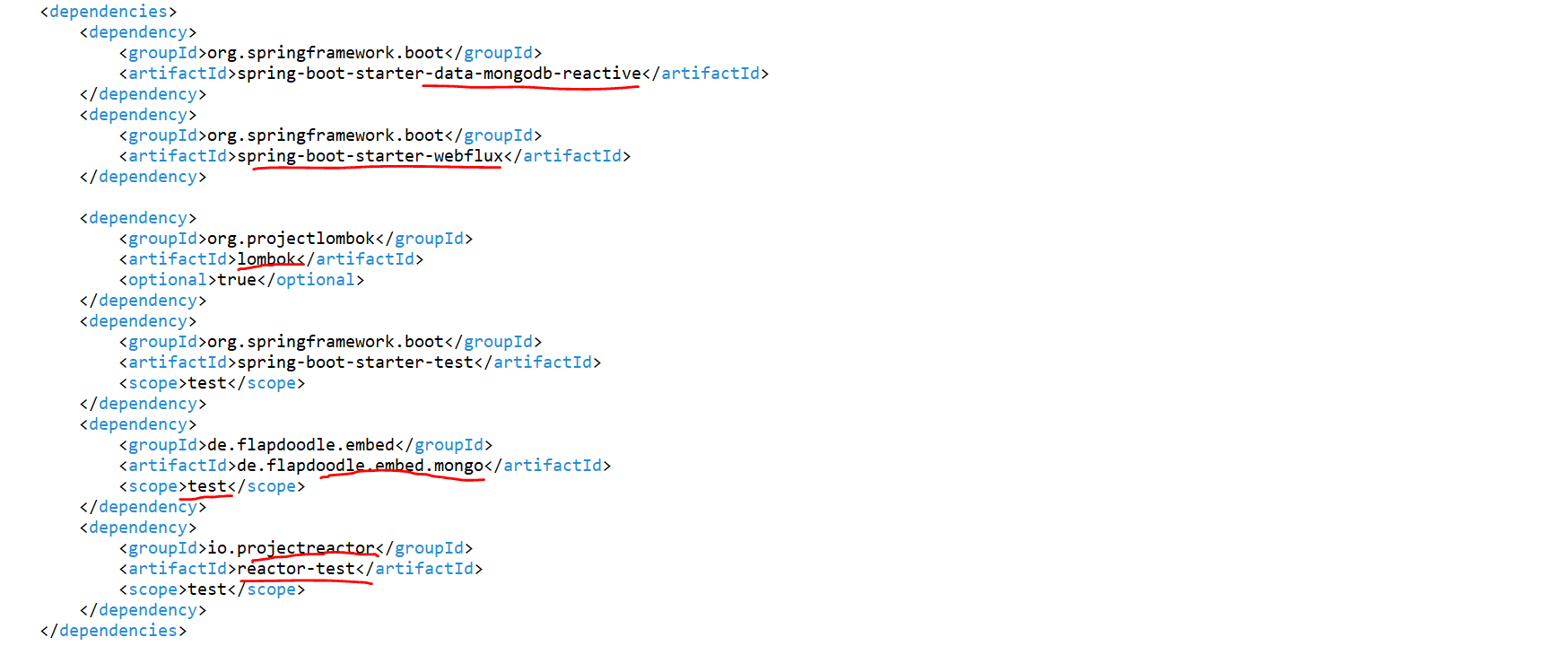
* **Spring Boot WebFlux | Spring Data Reactive MongoDB -CRUD Example**

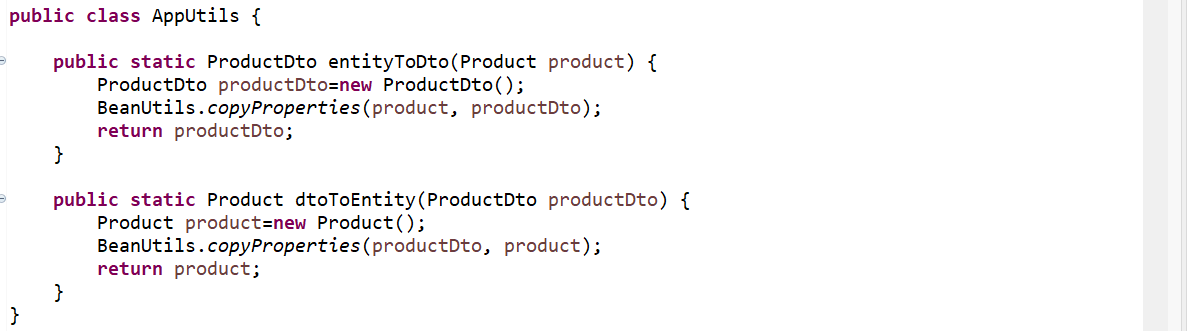
-------------------------------------------------------------------------------------------

App- **spring-reactive-mongo-crud**

Dependencies- **Spring data Reactive MongoDB**, **Spring Reactive Web**, **Lombok**, **Embedded** **MongoDB** **Database**



Now what I will do, I will just create a util package Where I will write the logic to convert entity to DTO and DTO to entity



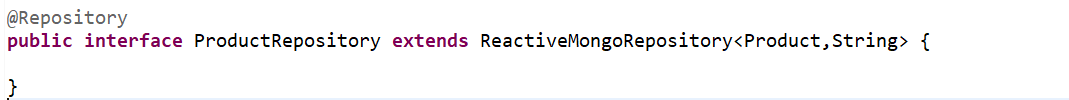
Entity is-



DTO is-

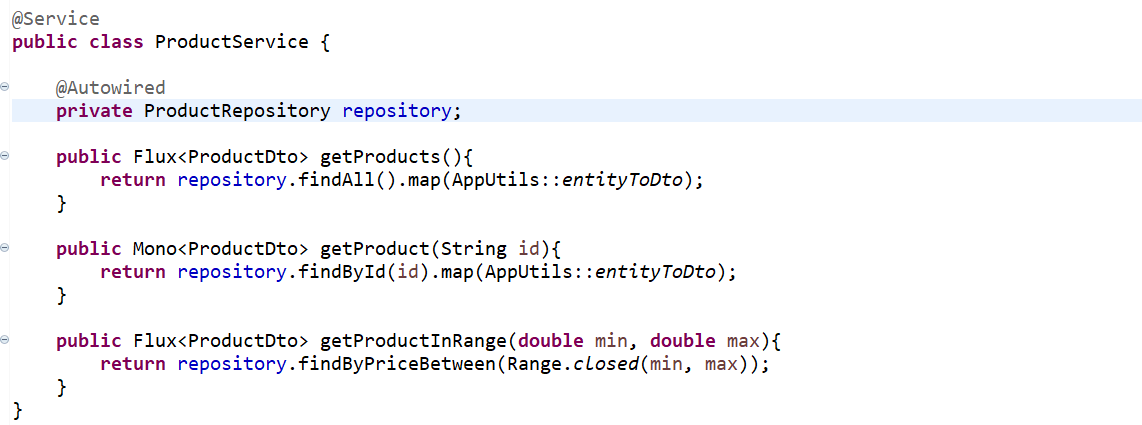


Repository is-



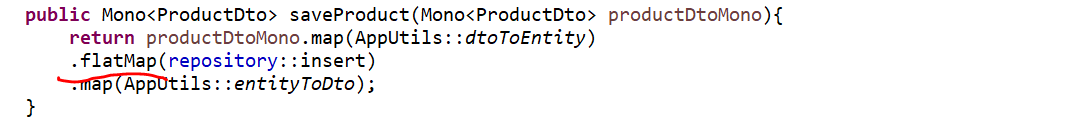
\*\*\* Now let's create the service layer. So, we will write all the crud operation in our service in a reactive manner. So, just create a service class. I will create a class called ProductService.

\*\*\* This is what the actual coding standard We don't need to directly play with our model or entity class. We always need to get a request and return the response with this DTO class.



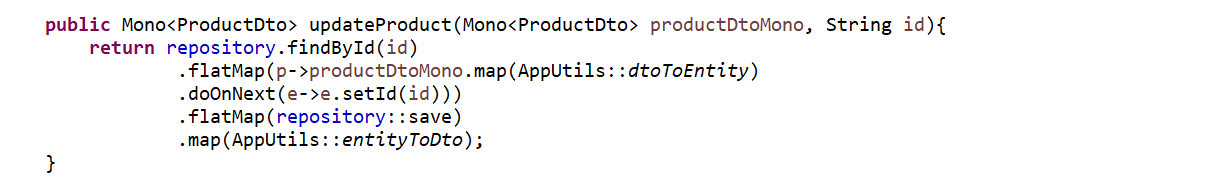
Now we created all the gate API. We just need to write a POST, PUT and DELETE. So, i will write a POST API.

So, we need to use the flat map here. So, this is how if you don't understand java8 how to use map and flat map You will get confused while writing reactive programming code. One to many you may need to use flat map. If it is a single mapping, then you can go with a map.

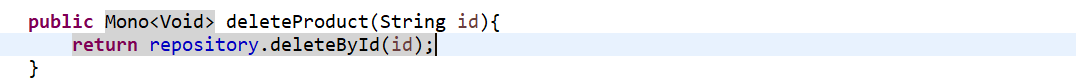


Let’s write PUT and DELETE mapping

First i was getting record from db. Then i used flat map, I got the request object and i convert that request object to entity as i was just updating the field. So, i just keep the id as it is Because this is the update operation Once i map that value i just save it. Then i return as a DTO.



-----------------------------------------------------------------------------------------------------------------------------------------



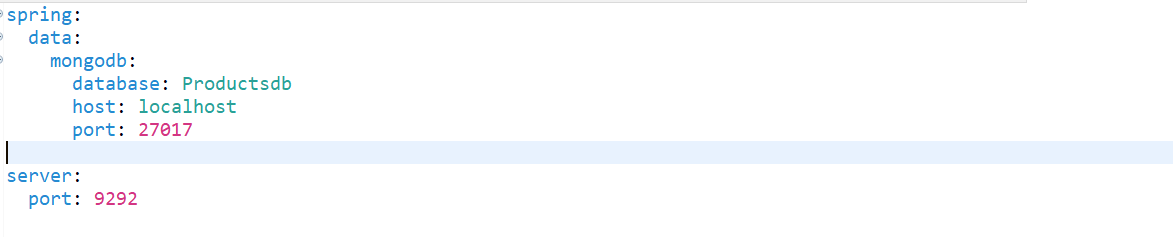
------------------------------------------------------------------------------------------------------------------------------------------



Now we can write our controller class



Now we need to add Mongo Driver related properties in our application.yml file



Now let’s start our application and open mongosh to connect to mongodb server 27017 and open mongo db compass community.

Application has been up on our port 9292.

Now go to postman and let’s verify our endpoint.

**POST** : <http://localhost:9292/products>

**{**

**"name": "p0142",**

**"qty": 22,**

**"price": 16272838**

**}**

**Response-**

**{**

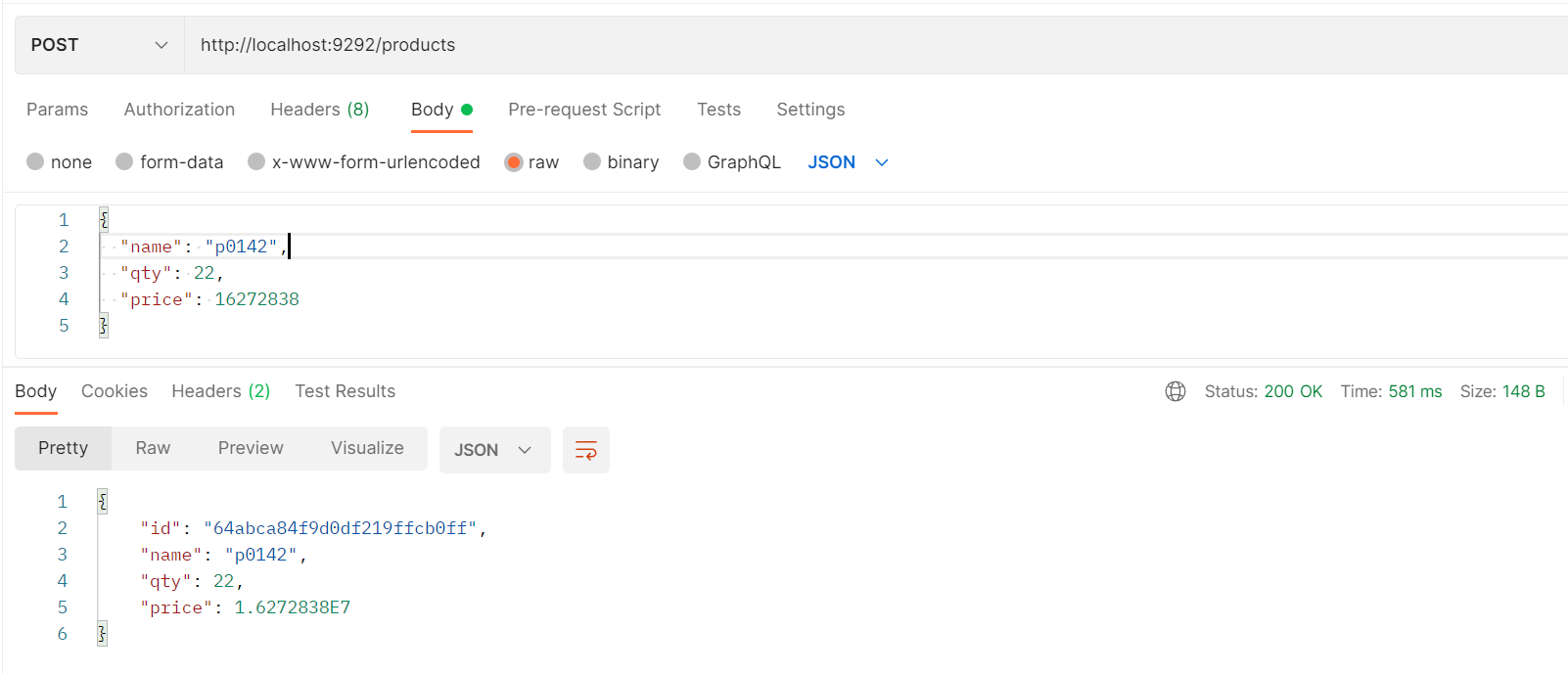
**"id": "64abca84f9d0df219ffcb0ff",**

**"name": "p0142",**

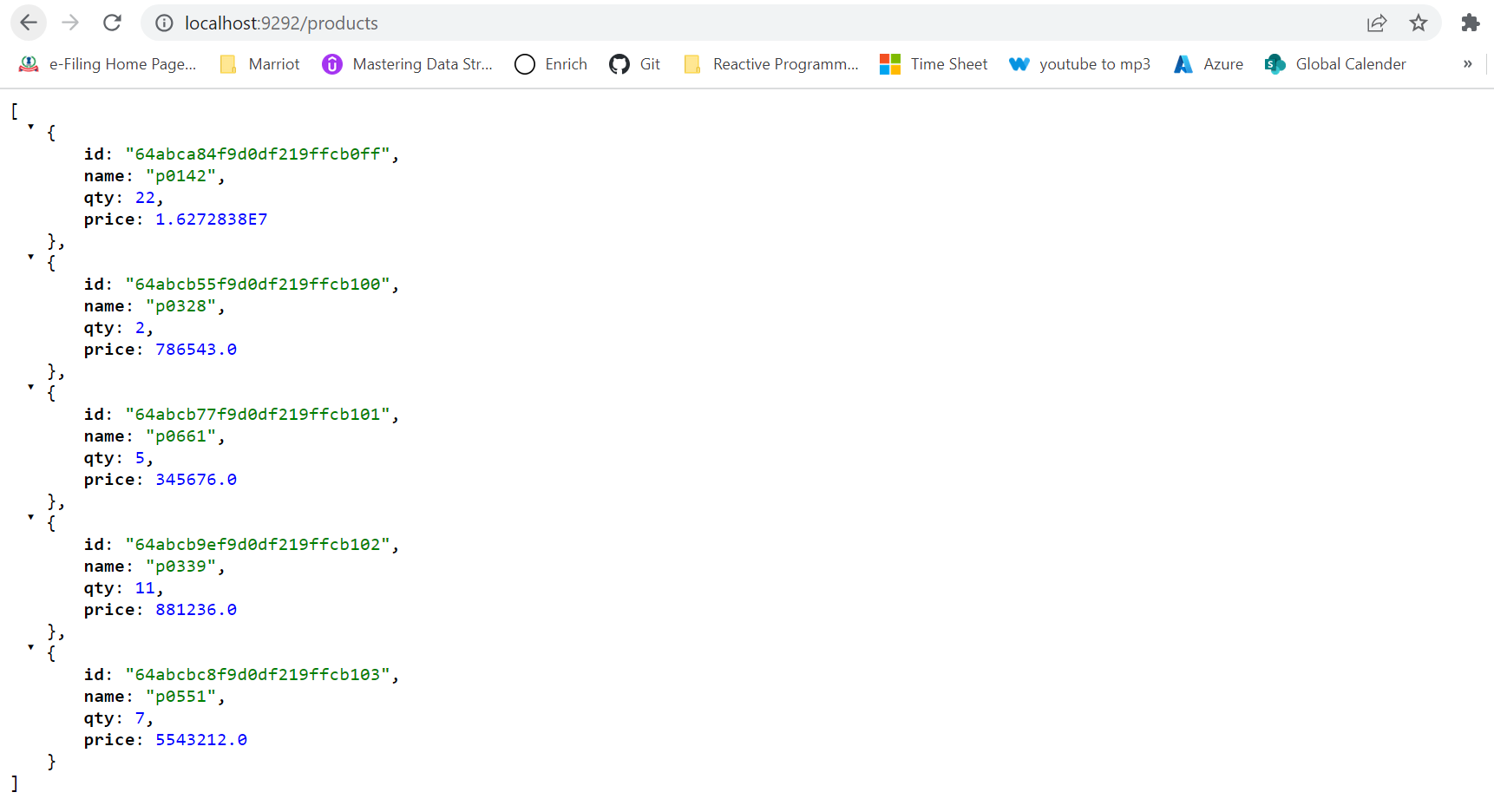
**"qty": 22,**

**"price": 1.6272838E7**

**}**



**GET** <http://localhost:9292/products>



Let’s search product based on id.

**GET** : <http://localhost:9292/products/64abca84f9d0df219ffcb0ff>

Response-

**{**

**"id": "64abca84f9d0df219ffcb0ff",**

**"name": "p0142",**

**"qty": 22,**

**"price": 1.6272838E7**

**}**

**GET** <http://localhost:9292/products/product-range?min=300000&max=900000>

**[**

**{**

**"id": "64abcb55f9d0df219ffcb100",**

**"name": "p0328",**

**"qty": 2,**

**"price": 786543.0**

**},**

**{**

**"id": "64abcb77f9d0df219ffcb101",**

**"name": "p0661",**

**"qty": 5,**

**"price": 345676.0**

**},**

**{**

**"id": "64abcb9ef9d0df219ffcb102",**

**"name": "p0339",**

**"qty": 11,**

**"price": 881236.0**

**}**

**]**

Now I just want to update the one product object. I want to update product price for id **64abcb77f9d0df219ffcb101** to **300**.

**PUT** <http://localhost:9292/products/update/64abcb77f9d0df219ffcb101>

**Request**-

**{**

**"name": "p0661",**

**"qty": 5,**

**"price": 300**

**}**

Response-

**{**

**"id": "64abcb77f9d0df219ffcb101",**

**"name": "p0661",**

**"qty": 5,**

**"price": 300.0**

**}**

Now let’s delete this updated product from product database.

**DELETE** <http://localhost:9292/products/delete/64abcb77f9d0df219ffcb101>

**GET** <http://localhost:9292/products>

