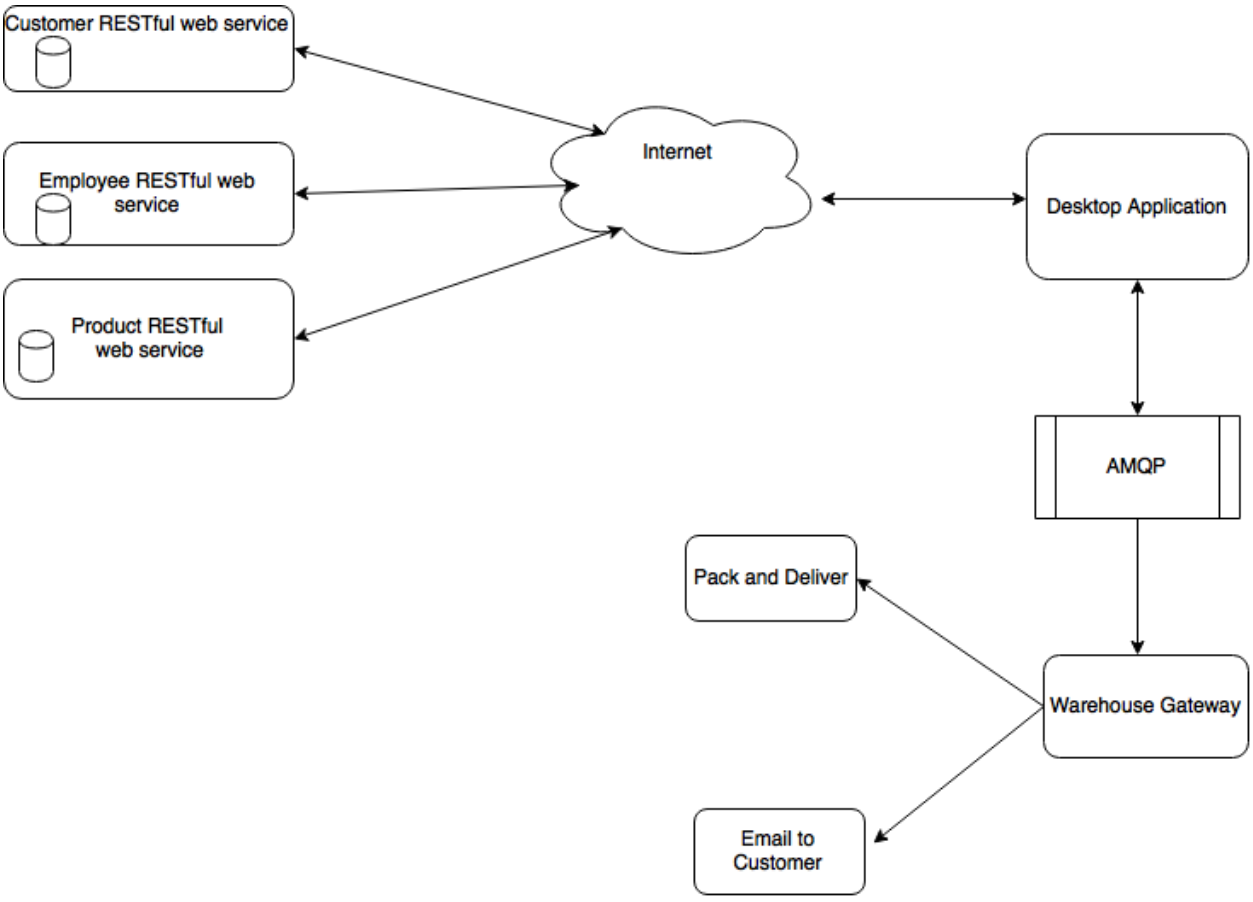
El Mehor

Introduction

El Mejor is an integrated enterprise solution for large e-commerce enterprises. It has 5 separate modules that work seamlessly while adhering to the separation of concerns principle.
It is implemented in a Component N-Tier architecture, where the main three components of the system are running separately as a RESTful web services. Access to these web services is controlled properly through a fine security mechanism provided by Spring framework.
Each web service is most probably be running on a separate machine.
The client module interacts with a warehouse module through Advanced Message Queuing Protocol.

High Level Design



Each of these three RESTful web services run it's own server, and they have their own small database. However, each of them will hit an external common database for

authentication and authorization. We made that decision for having a common and centralized security measure.

Customer Module

Out of the three parts in our project Elmejor, I was responsible for handling all the work required to be done for the customer module, which includes the basic operation of adding , updating , listing and deleting the customer from the database. The customer module comes into play when the user authenticate and is authorized to access the module. It has its own authorization.

Functionalities and technologies integrated

In this module there are only two domains namely Customer and Address with one-to-one mapping. Each customer has one Address. Introducing the Mapping technology while doing this and also the domains are annotated with the Validation tags. Showing our use of Validation technology. Similarly Aspect Oriented Programming is also practiced in this module which is responsible for logging inside any service class methods of edu.mum.elmejor.rest.service package with the concepts of Advice and Pointcuts. The concept of logging to external file is also implemented using slf4j-log4j12 library. The module is serving as a RESTful api to third party application.

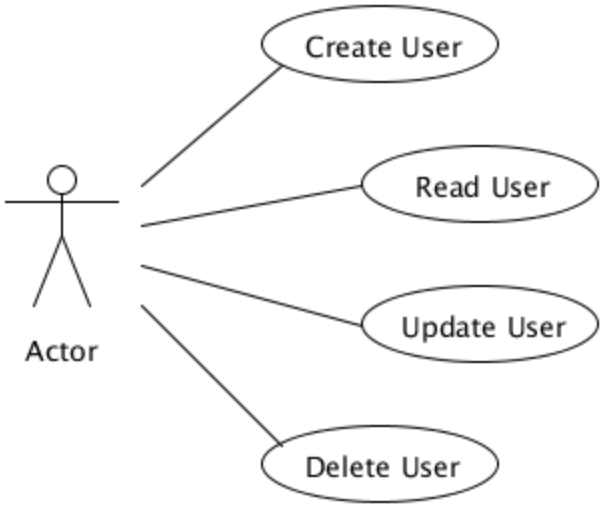
Problems and Solutions

While developing the module we're in a group and there we face problem while integrating with the remaining modules. There were few problems involving security integration in the module and later we figured it was not a big problem just a referencing to wrong file in configuration. While talking about the customer module all alone, I'd to spent little bit too much time while debugging the application with the validation and Logging using AOP and by using the lecture demos and making use of Internet I solved the issue. The issues were with passing the parameters in Advice in Aspect.

Possible Improvements

Currently I've just included the basic CRUD operation in this customer module but further more I'm planning to implement other functionalities like limiting the customer count while selecting customer list, searching the customer with different searching criteria and parameters provided. The use of AOP can be extended in other classes too.

Use Case Diagram



RESTfull Product service

As part of the three micro services constituting our project, the "**elmajor-Product"** RESTfull micro service has two simple domain classes. **Product** and **Category** classes which are associated many to many.

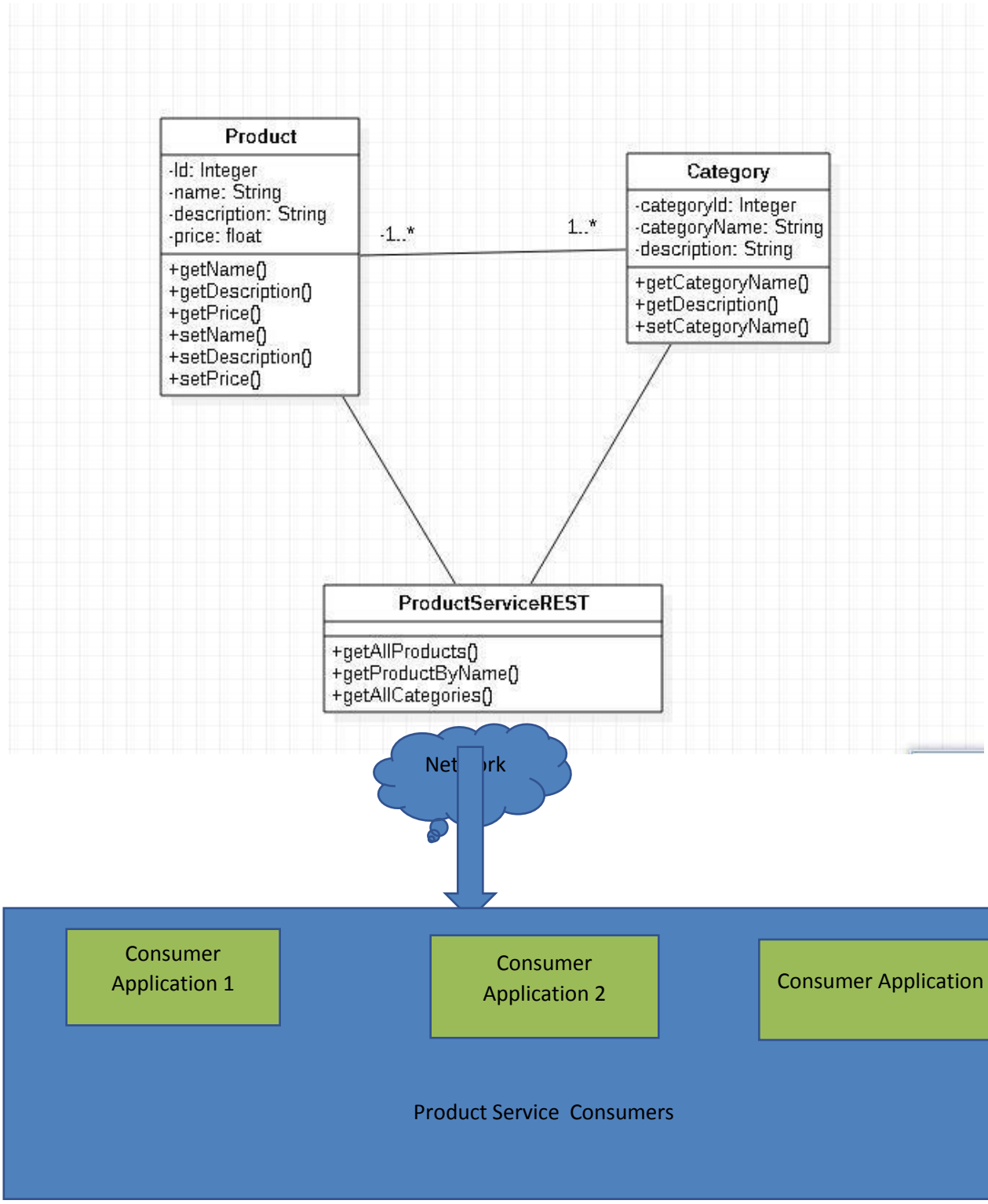
A class RESTfull service class is used to produce data from the database in different formats as per the request type coming form any REST client. The data format can be XML or JSON.

The produced Product xml/JSON data will be ready to be consumed by Other microservices running on different machines in the same LAN. The Consumer application doesn't need to have the product database. It only consumes the data through the RESTfull web service using path URI and the GET http method.

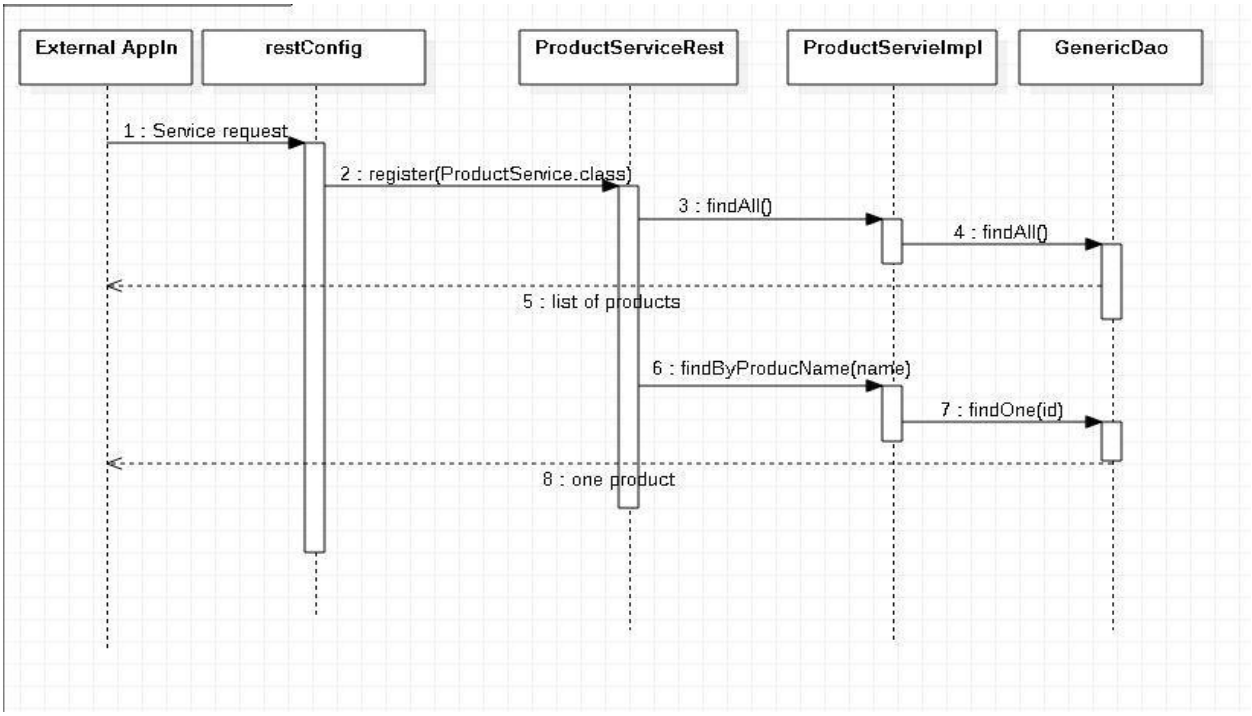
The RESTfull product class is as shown below.

```java
package edu.mum.elmejor.rest.service;

import java.util.List;

@Component
@Path("/products")
public class ProductServiceRest {

    @Autowired
    CategoryService categoryService;

    @Autowired
    ProductService productService;

   //Read All Products
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public List<Product> getAllProducts()
    {

        return productService.findAll();
    }


    @GET
    @Path("/productName")
    @Produces(MediaType.APPLICATION_JSON)
    public Product getProductByName(@PathParam("name") String name)
    {
        name="Toshiba";
        return productService.findByProducName(name);
    }
```

Class Diagram:



Sequence Diagram



The sequence diagram shows how an external application consumes product using the RESTfull product service.

Challenges:

While doing my part, I had difficulties making use of the JERSY for implementing the REST services.

The Jersey implementation provides a client library to communicate with a RESTful web service. At first, I couldn't get it working that, it failed to scan the service class in the web.xml. finally Mr. Gedion solved the problem and it worked fine.

There were also issues and exceptions with the eclipse **pivotal tc server**. I got server un able to start error, that I couldn't run some of the services on server.