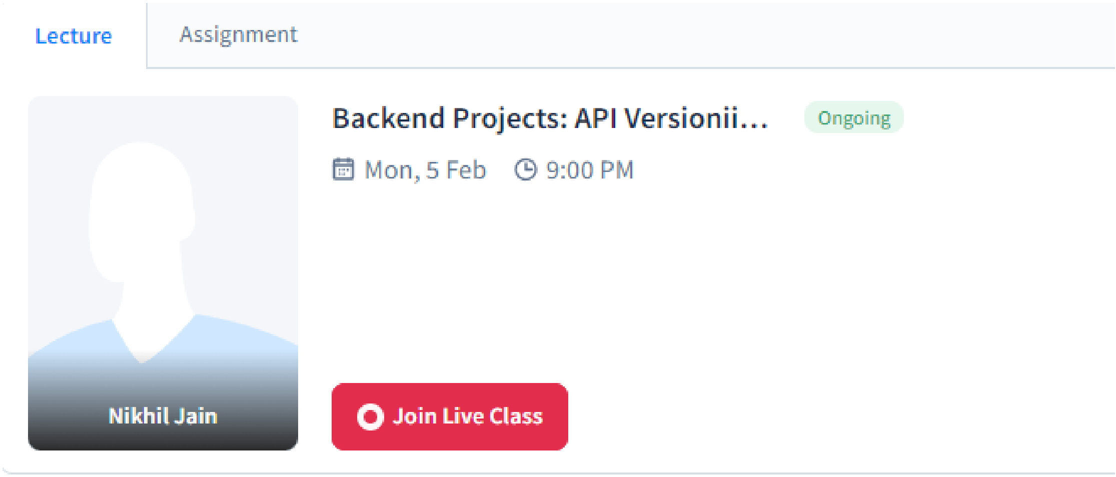


Backend Projects: API Versionii…

Mon, 5 Feb 9:00 PM

https://chat.whatsapp.com/HP7sLbUJAxbGZhrAuzqzUC

Nikhil Jain11:03 PM

https://www.geeksforgeeks.org/difference-between-controller-and-restcontroller-annotation-in-spring/

Nikhil Jain10:50 PM

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html#acceptHeaderRequestCallback(java.lang.Class)

Nikhil Jain10:45 PM

https://stackoverflow.com/questions/39890849/what-exactly-is-field-injection-and-how-to-avoid-it

Nikhil Jain10:43 PM

https://fakestoreapi.com/docs

Nikhil Jain9:53 PM

https://medium.com/javarevisited/understanding-the-distinction-between-controller-and-restcontroller-essential-knowledge-for-b8b1ae84097a#:~:text=A%20Controller%20returns%20a%20View,be%20consumed%20by%20other%20applications.

Nikhil Jain9:17 PM

https://projectlombok.org/

Nikhil Jain9:07 PM

https://docs.google.com/document/d/1Gn2ib5YhhpcFUiWGAUbCpg0ZPh3m_wSA-9IolGMjkIE/edit#heading=h.hteovoit9b96

PRD_ HLD of
Ecommerce Website

Agenda

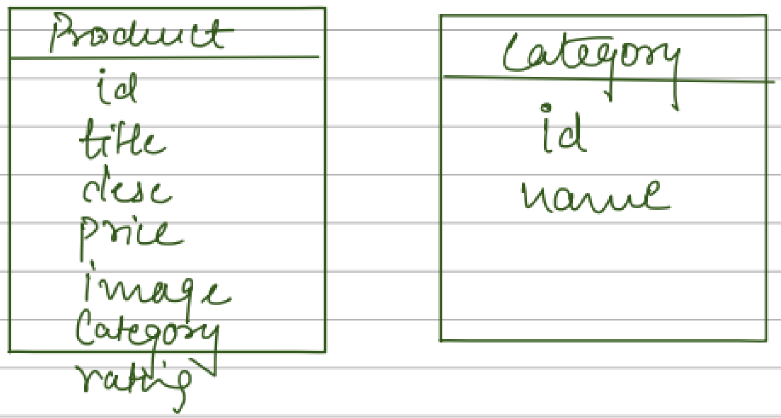
Start 9:05pm

1) LLD of product Service
2) Code of product Service

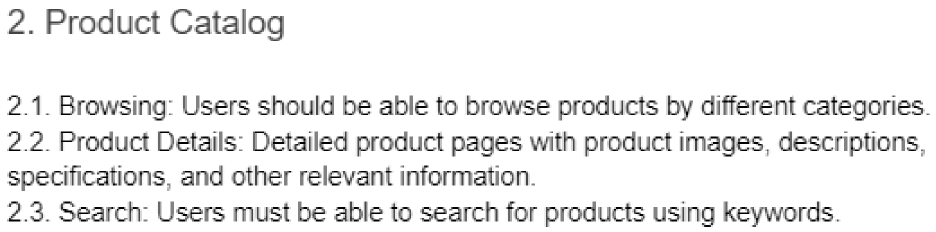
Product Service

↳ Handles all the products

Class diagram



PRD doc



2. Product Catalog

2.1. Browsing: Users should be able to browse products by different categories.
2.2. Product Details: Detailed product pages with product images, descriptions, specifications, and other relevant information.
2.3. Search: Users must be able to search for products using keywords.

Product service: we should be able to see all the products.. based on content what are the product needs to have??? In LLD what all we needed. Class diagram?

Below attribute product should have:

ID, name/title, description, Price, Image, Category, Rating etc..

This product service talks about product…

We also need to have a category class: ID, Name.

If I have to get all products by a category how can I get it? I don’t need join, I can do group by by category ID… where category_id = x.

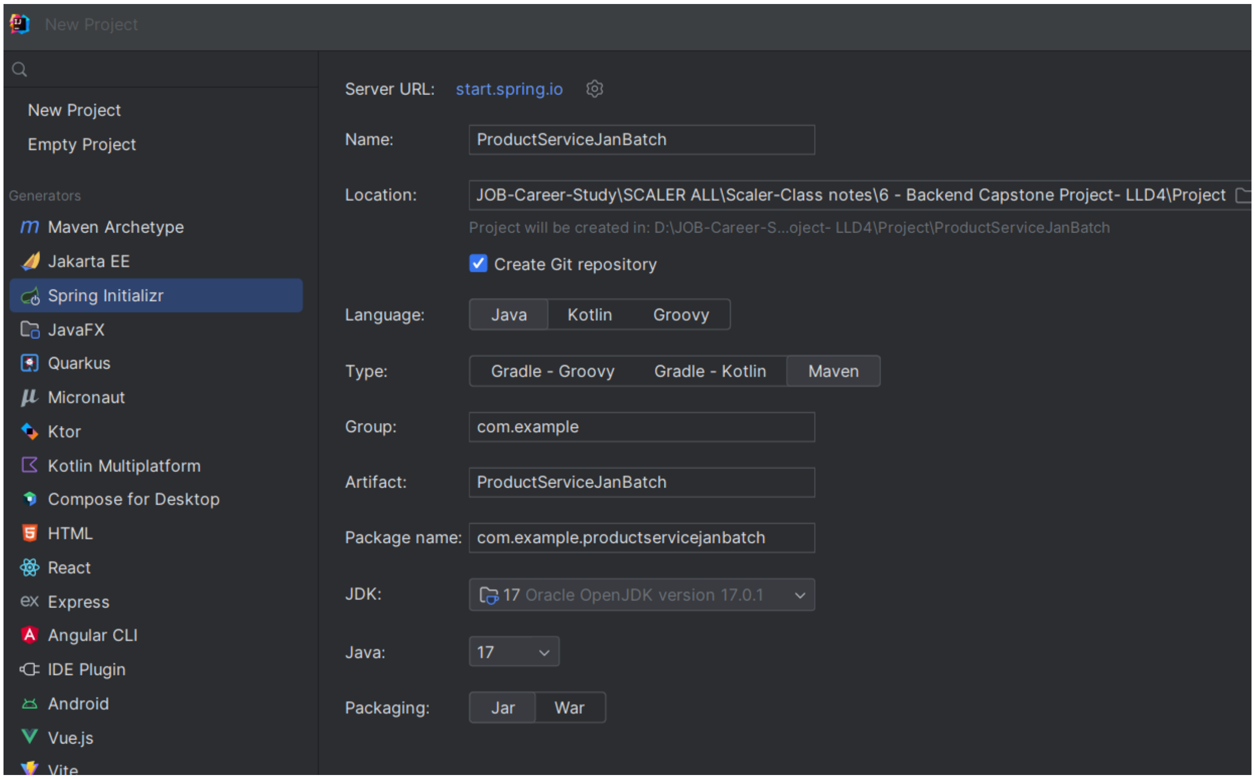
What should we expect by the end of product service?

We will store product in DB, at the end of service(2-2.5 class) we will have product in DB.. by end of service we will have productMicroservice which should connect to a DB and expose some API for the product and all data should be stored in DB.

Now in Intellij:

Create a New project

D:\JOB-Career-Study\SCALER ALL\Scaler-Class notes\6 - Backend Capstone Project- LLD4\Project



Server URL: start.spring.io

Name: ProductServiceJanBatch

Location: JOB-Career-Study\SCALER ALL\Scaler-Class notes\6 - Backend Capstone Project- LLD4\Project

Project will be created in: D:\JOB-Career-S...oject- LLD4\Project\ProductServiceJanBatch

☑ Create Git repository

Language: Java | Kotlin | Groovy

Type: Gradle - Groovy | Gradle - Kotlin | Maven

Group: com.example

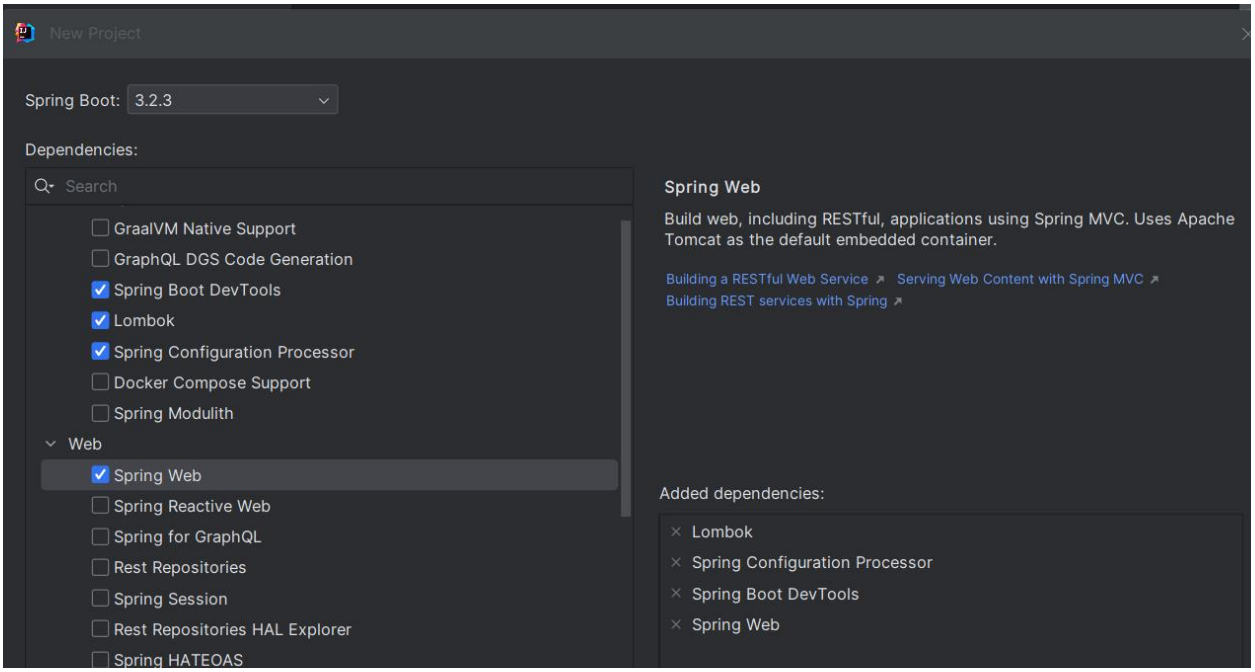
Artifact: ProductServiceJanBatch

Package name: com.example.productservicejanbatch

JDK: 17 Oracle OpenJDK version 17.0.1

Java: 17

Packaging: Jar | War



Spring Boot: 3.2.3

Dependencies:

- ☐ GraalVM Native Support
- ☐ GraphQL DGS Code Generation
- ☑ Spring Boot DevTools
- ☑ Lombok
- ☑ Spring Configuration Processor
- ☐ Docker Compose Support
- ☐ Spring Modulith
- Web
  - ☑ Spring Web
  - ☐ Spring Reactive Web
  - ☐ Spring for GraphQL
  - ☐ Rest Repositories
  - ☐ Spring Session
  - ☐ Rest Repositories HAL Explorer
  - ☐ Spring HATEOAS

Spring Web

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

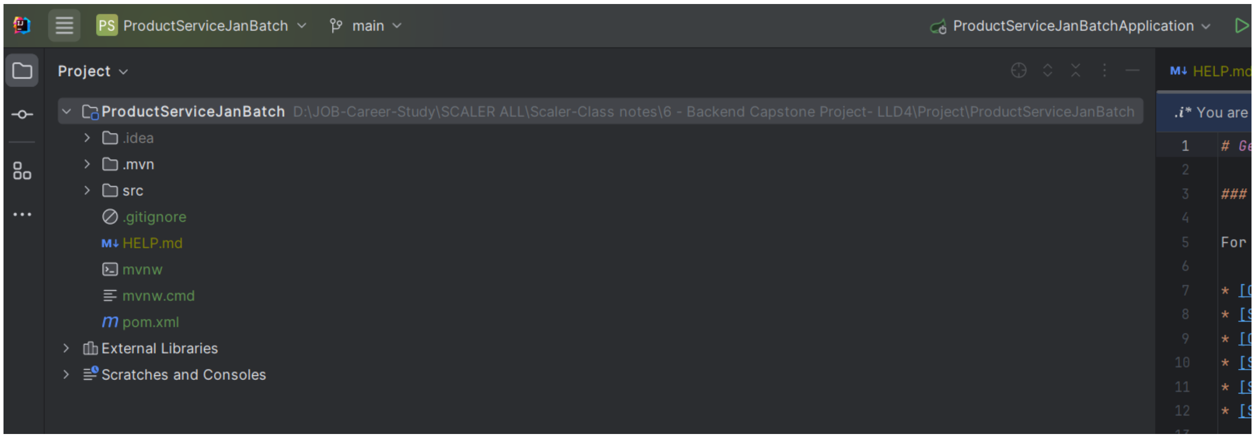
Building a RESTful Web Service ↗ Serving Web Content with Spring MVC ↗
Building REST services with Spring ↗

Added dependencies:

- Lombok
- Spring Configuration Processor
- Spring Boot DevTools
- Spring Web

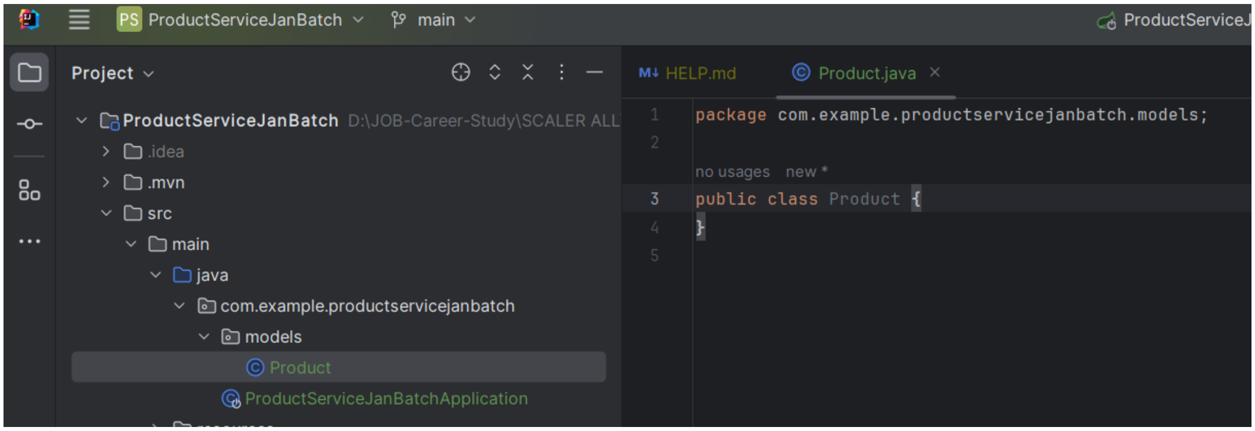
Lombork: https://projectlombok.org/

Says how does lombork help with some boiler plate code.. BPC: provide us getter setter and constructer..


ProductServiceJanBatch
main
ProductServiceJanBatchApplication
Project
HELP.md
ProductServiceJanBatch D:\JOB-Career-Study\SCALER ALL\Scaler-Class notes\6 - Backend Capstone Project- LLD4\Project\ProductServiceJanBatch
.idea
.mvn
src
.gitignore
HELP.md
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles

When we start creating a service I first create Models.

For model I will create a package first.


ProductServiceJanBatch
main
ProductService
Project
ProductServiceJanBatch D:\JOB-Career-Study\SCALER ALL
.idea
.mvn
src
main
java
com.example.productservicejanbatch
models
Product
ProductServiceJanBatchApplication
HELP.md
Product.java

```
package com.example.productservicejanbatch.models;

no usages   new *
public class Product {
}
```

Why Long?

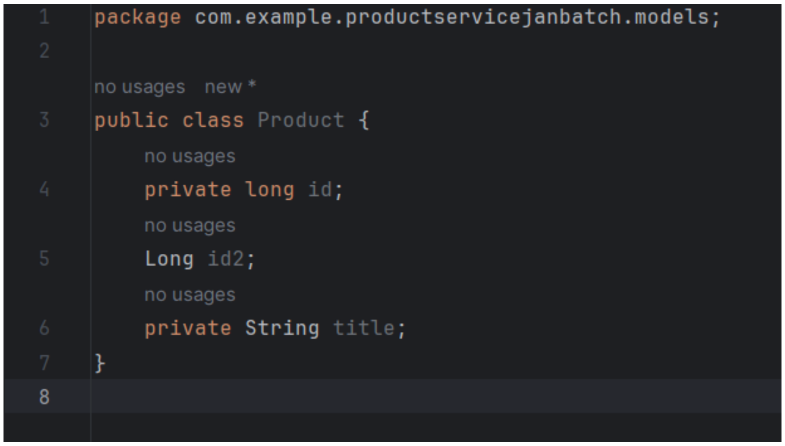
he Long class wraps a value of the primitive type long in an object. An object of type Long contains a single field whose type is long.

In addition, this class provides several methods for converting a long to a String and a String to a long, as well as other constants and methods useful when dealing with a long.

This is a value-based class; programmers should treat instances that are equal as interchangeable and should not use instances for synchronization, or unpredictable behavior may occur. For example, in a future release, synchronization may fail.

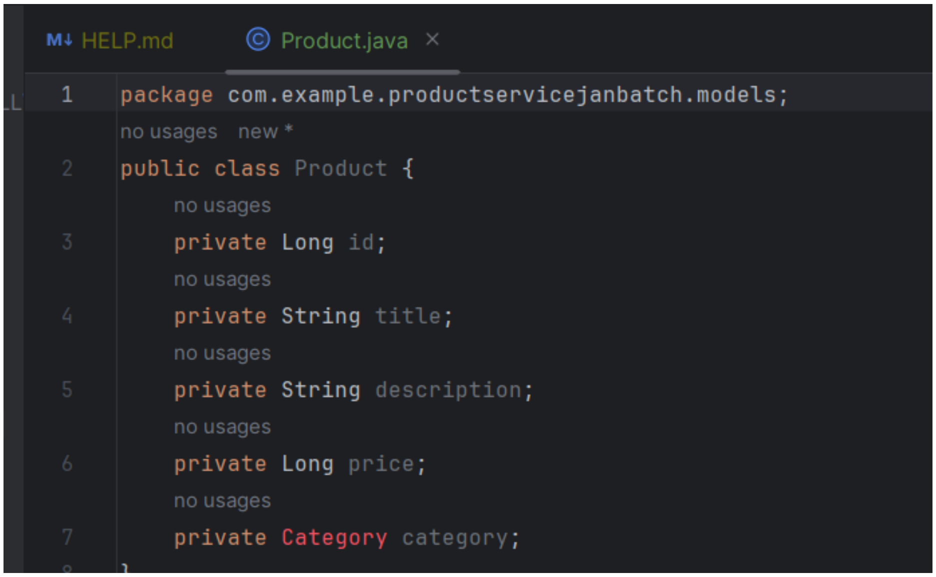
Another reason: When u r using this data for serilization and put it out in an API in json (serilizaed string)..

In product we have id [we will use which kind of id? long].. there are 2 ways of defining long. Long and Long (Wrapper class)



```
package com.example.productservicejanbatch.models;

no usages   new *
public class Product {
    no usages
    private long id;
    no usages
    Long id2;
    no usages
    private String title;
}

```

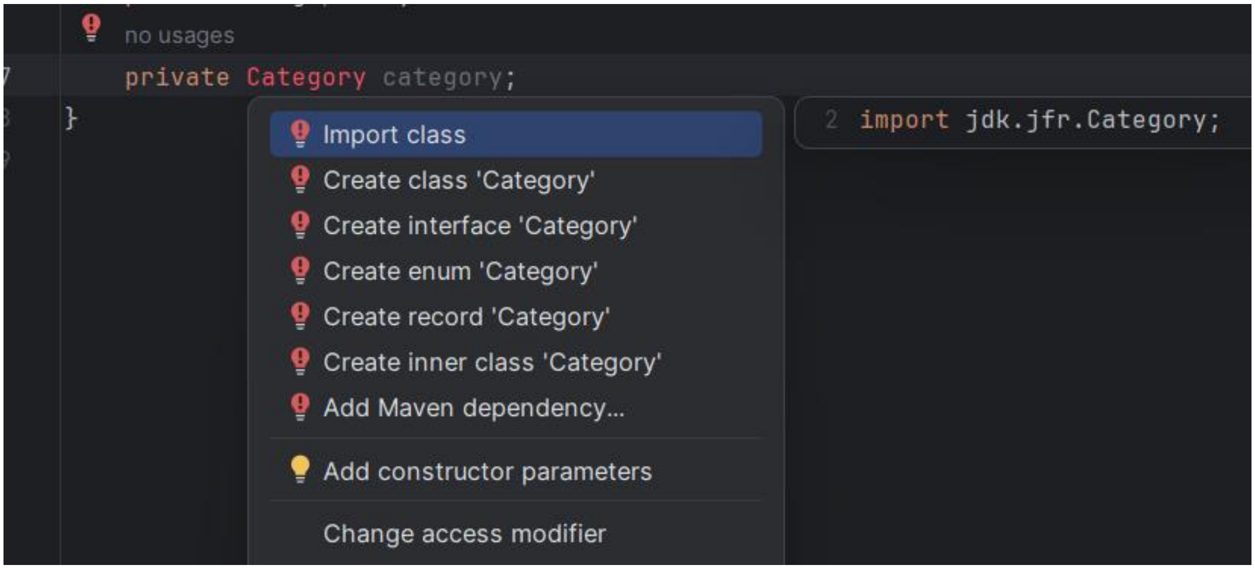
Y Long? When u r using this data for serilization and put it out in an API in json (serilizaed string)..



```
package com.example.productservicejanbatch.models;
no usages   new *
public class Product {
    no usages
    private Long id;
    no usages
    private String title;
    no usages
    private String description;
    no usages
    private Long price;
    no usages
    private Category category;
}
```

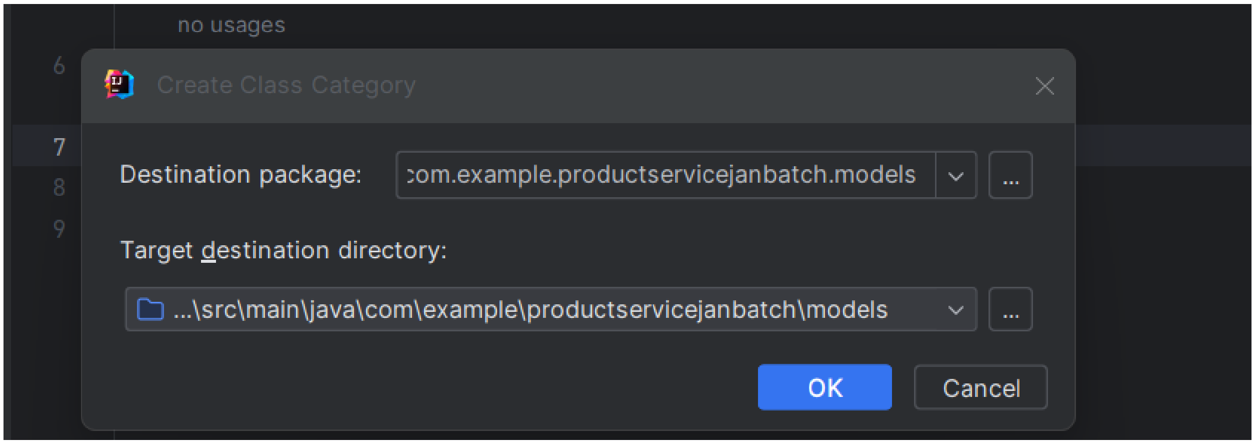
We need to create category class in models.

Click more action on category:



```
no usages
private Category category;
}
```

```
2 import jdk.jfr.Category;
```



Destination package: com.example.productservicejanbatch.models

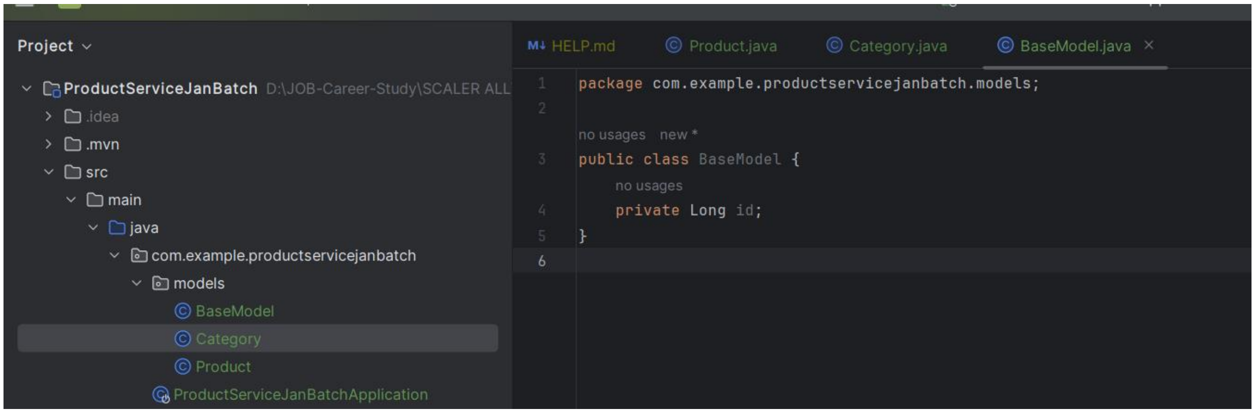
Target destination directory: ...\src\main\java\com\example\productservicejanbatch\models

In category class also have id and name.

```
public class Category {
    private Long id;
    private String name;
}
```

they have commonality… id… it’s a small app have 2 class. As app grows it will have more commonality. So I will have basemodel class which will have these common.

I add just id in basemodel..



```
package com.example.productservicejanbatch.models;

public class BaseModel {
    private Long id;
}
```

Also I extend product and category with basemodel.

```
package com.example.productservicejanbatch.models;

public class Category extends BaseModel{
    private String name;
}
```

```
package com.example.productservicejanbatch.models;
public class Product extends BaseModel {
    private String title;
    private String description;
    private Long price;
    private Category category;
}
```

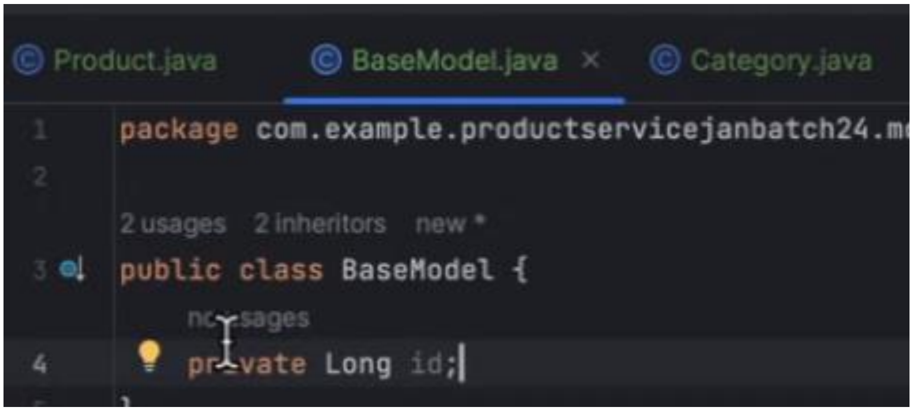
currently basemodel is a class if I don’t want to create obj of basemodel I would have create it abstract..

when Someone wants to assign value when create a product with current structure… no. constructor is there. As getter setter are missing.

Lombok documentation: https://projectlombok.org/

We cover spring boot based application..

When we first start creating service first we create **MODELS**. For models I will create a package first. I will need below models such as product [id,], we use Long instead of long.. why wrapper?? Convert string to Long and Long to String, for sterilization and put it out of an API in a JSON (serilized string) .. or null case, it contains many methods. OOPS is not violated. Long is taken from DB perspective not from HLD perspective..



```
package com.example.productservicejanbatch24.m

public class BaseModel {
    private Long id;
```

Use Lombok for getter setter.

@Data: which action/ annotation to be used always go to documentation. Donty use data…

Check lombok @data..

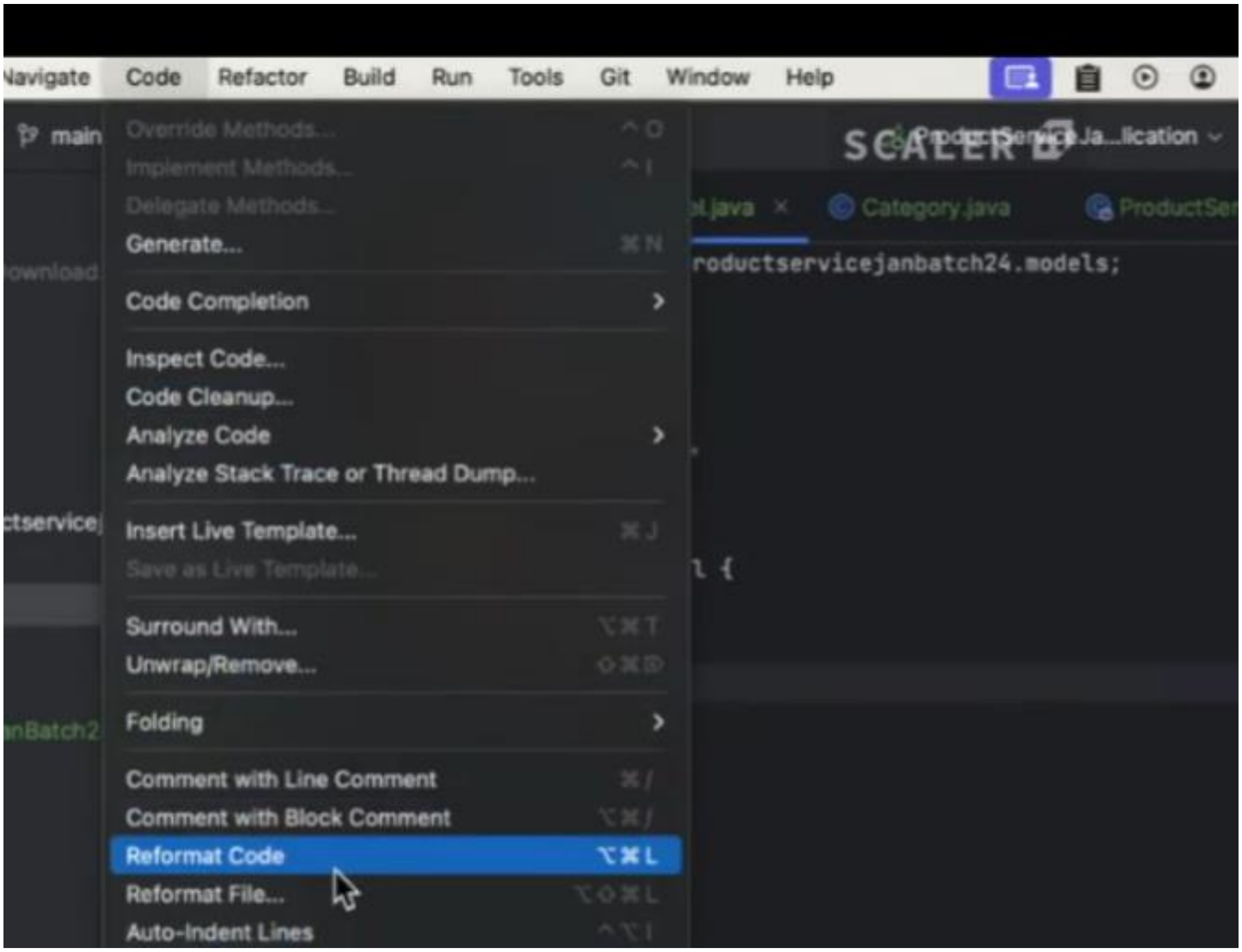
```
package com.example.productservicejanbatch.models;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class BaseModel{
    private Long id;
}
```

```
package com.example.productservicejanbatch.models;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
```

```
public class Product extends BaseModel {
    private String title;
    private String description;
    private Long price;
    private Category category;
}
```

```
package com.example.productservicejanbatch.models;

import lombok.Getter;
import lombok.Setter;

@Getter
@Setter
public class Category extends BaseModel{
    private String name;
}
```

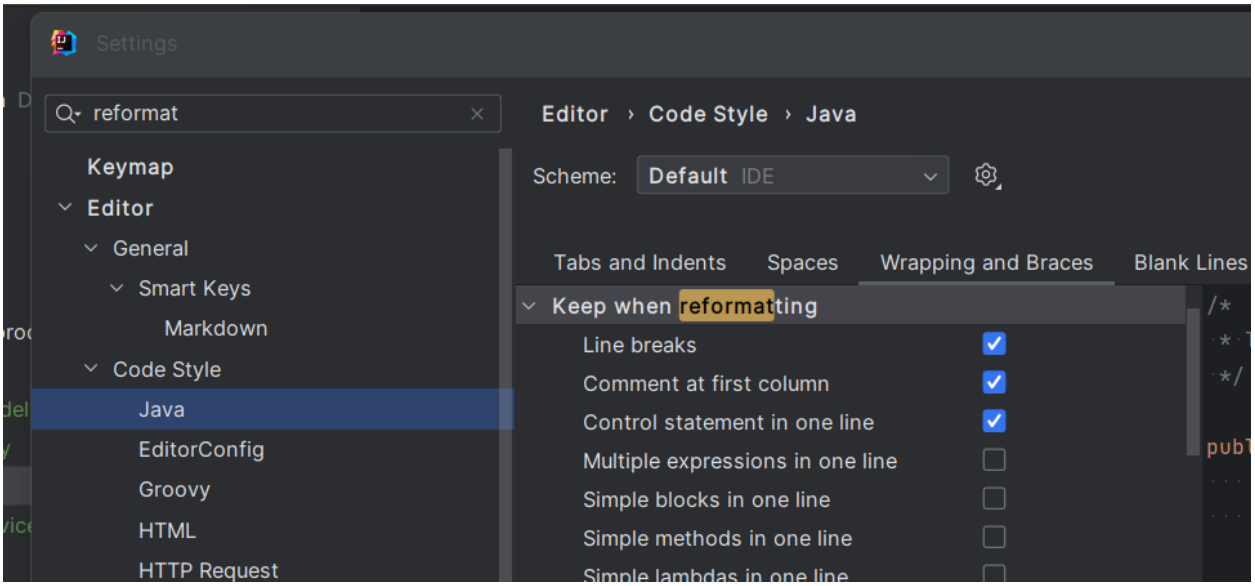
Reformat code: will make sure that my code is in good format. Every code has a checkstack associate to it.



```
roductservicejanbatch24.models;
l {
```

It will format the code in more readable manner. Generally people use default one from intellj.

Always reformat and push… perfect line space will be taken care.. help with review comment on file is not formatted.. etc.



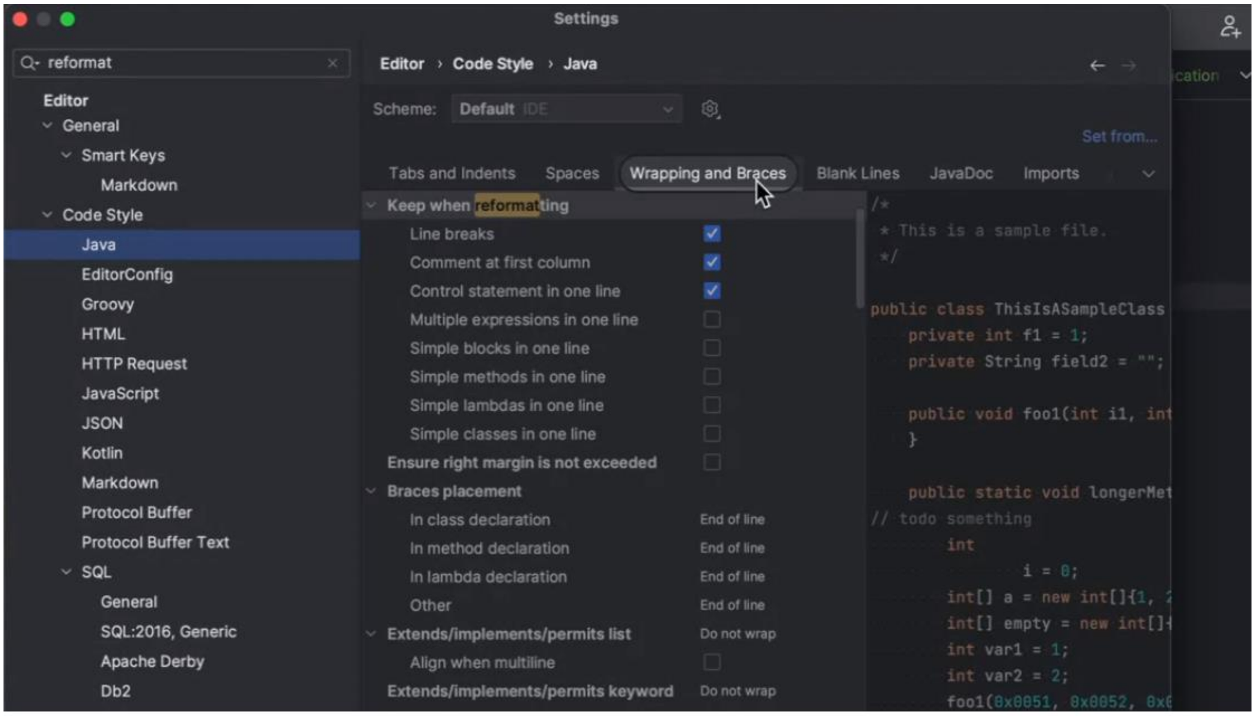
Editor › Code Style › Java

Scheme: Default IDE

Keep when reformatting

- Line breaks ☑
- Comment at first column ☑
- Control statement in one line ☑
- Multiple expressions in one line ☐
- Simple blocks in one line ☐
- Simple methods in one line ☐
- Simple lambdas in one line ☐

Make it readable format… every project will have check stack… people use default one from IDE..



Editor › Code Style › Java

Scheme: Default IDE

Keep when reformatting

- Line breaks ☑
- Comment at first column ☑
- Control statement in one line ☑
- Multiple expressions in one line ☐
- Simple blocks in one line ☐
- Simple methods in one line ☐
- Simple lambdas in one line ☐
- Simple classes in one line ☐

Ensure right margin is not exceeded ☐

Braces placement

- In class declaration: End of line
- In method declaration: End of line
- In lambda declaration: End of line
- Other: End of line

Extends/implements/permits list: Do not wrap

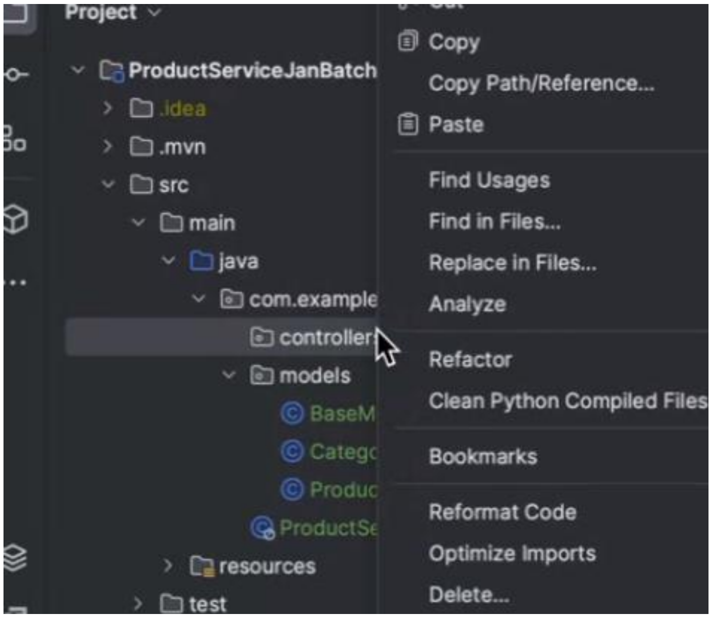
- Align when multiline ☐

Extends/implements/permits keyword: Do not wrap

```
/*
 * This is a sample file.
 */

public class ThisIsASampleClass
    private int f1 = 1;
    private String field2 = "";

    public void foo1(int i1, int
    }

    public static void longerMet
// todo something
        int
                i = 0;
        int[] a = new int[]{1,
        int[] empty = new int[]
        int var1 = 1;
        int var2 = 2;
        foo1(0x0051, 0x0052, 0x
```

You can export your setting. Every company follow their own.

Once models are created, next thing is controller.



We are done with models.. This is a product service we will connect with product so I will create productController.. when someone with product controller… they will connect with a path….

When we connect to a controller we connect to a path.. such as www.xyz.com/api/...

This is how end point look like. Endpoint is a combination of Domain name + path of entity api.

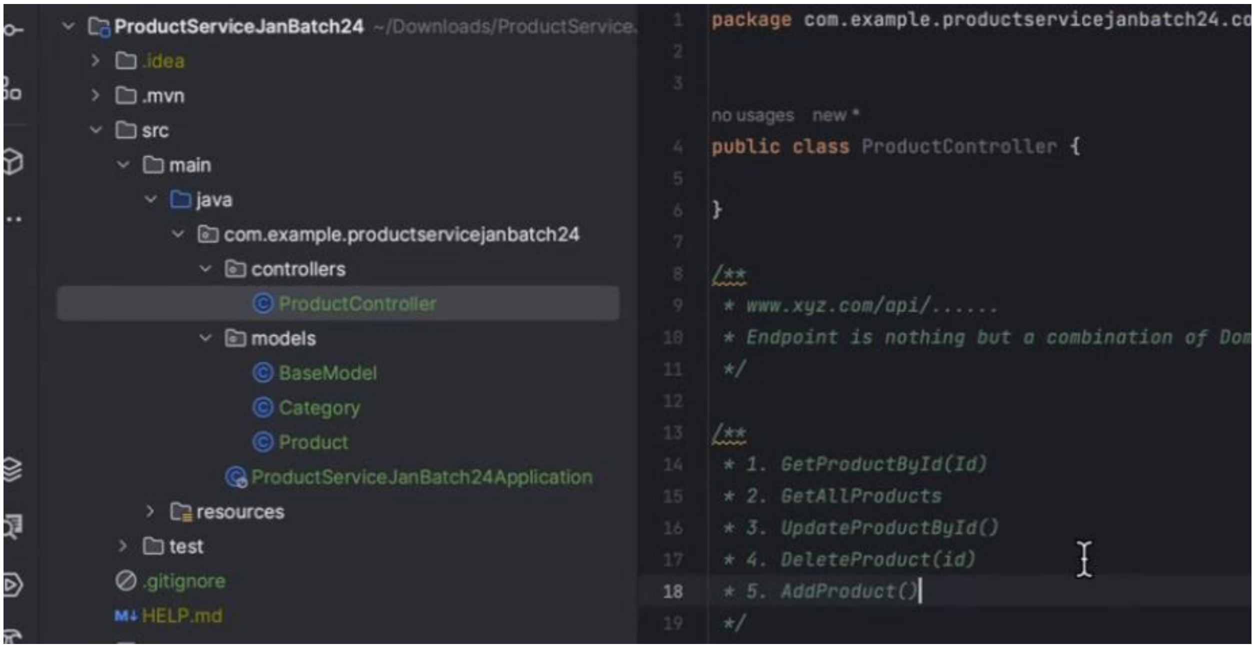
End point is a combination of domain name + path of API or entity api..

In API we don’t put domain name. taken care by app ppty. We take care everything after DOTCom /.com..

What API product service should have???

CRUD, getProductById(), addProduct(), deleteProduct(),

So I will add these apis.


ProductServiceJanBatch24 ~/Downloads/ProductService.
.idea
.mvn
src
main
java
com.example.productservicejanbatch24
controllers
ProductController
models
BaseModel
Category
Product
ProductServiceJanBatch24Application
resources
test
.gitignore
HELP.md
no usages new *

```
package com.example.productservicejanbatch24.co


public class ProductController {

}

/**
 * www.xyz.com/api/......
 * Endpoint is nothing but a combination of Dom
 */

/**
 * 1. GetProductById(Id)
 * 2. GetAllProducts
 * 3. UpdateProductById()
 * 4. DeleteProduct(id)
 * 5. AddProduct()
 */
```

1st API:

I need getproduct… I need get product of multiple types..

getProductByID(ID)

GetAllProducts

updateProductByID(ID)

deleteProductByID(ID)

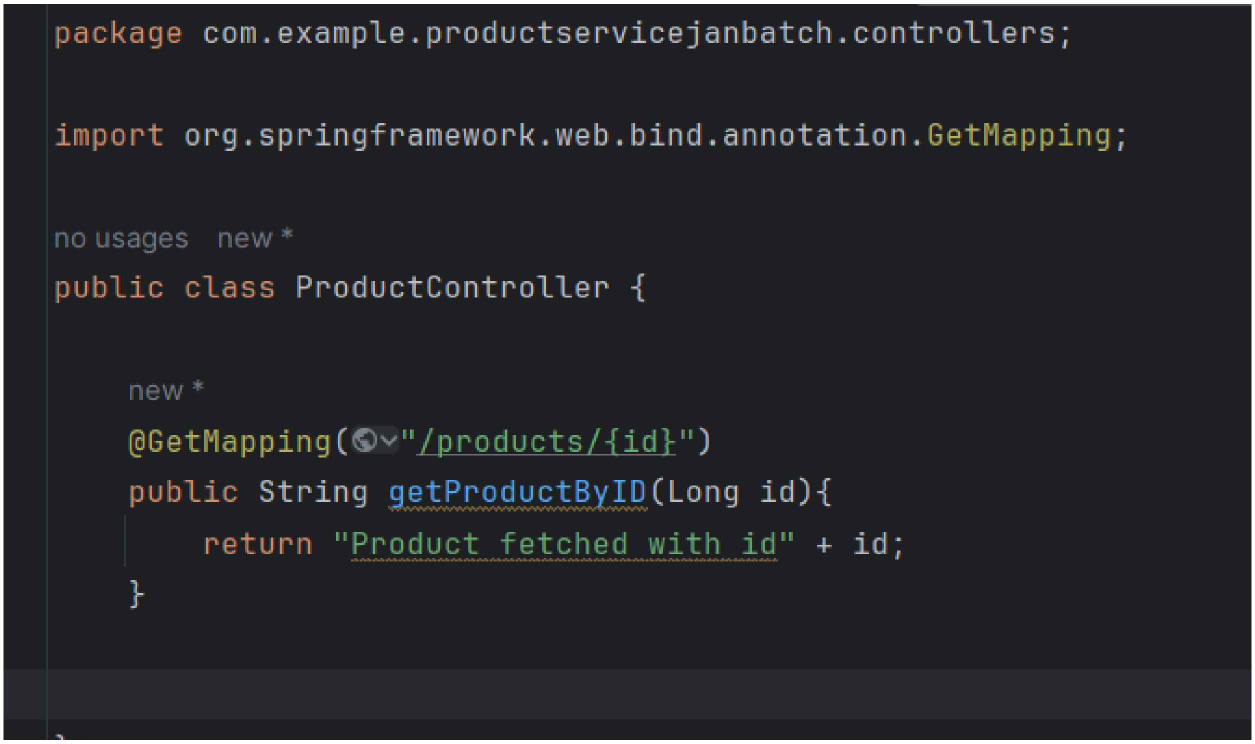
addProduct()

rest convention: Product should be plural format as we are searching from a list of products.

```
package com.example.productservicejanbatch.controllers;

public class ProductController {

    public String getProductByID(Long id);
}
```

it will be getmapping… it expects a path "/ entity name (products)/ how? By id

```
@GetMapping("/products/{id}")
public String getProductByID(Long id){

}
```



```
package com.example.productservicejanbatch.controllers;

import org.springframework.web.bind.annotation.GetMapping;

no usages   new *
public class ProductController {

    new *
    @GetMapping("/products/{id}")
    public String getProductByID(Long id){
        return "Product fetched with id" + id;
    }
```

Another API:

```
    @GetMapping("/products/")
    public List<String> getAllProducts(){

    }
}
```

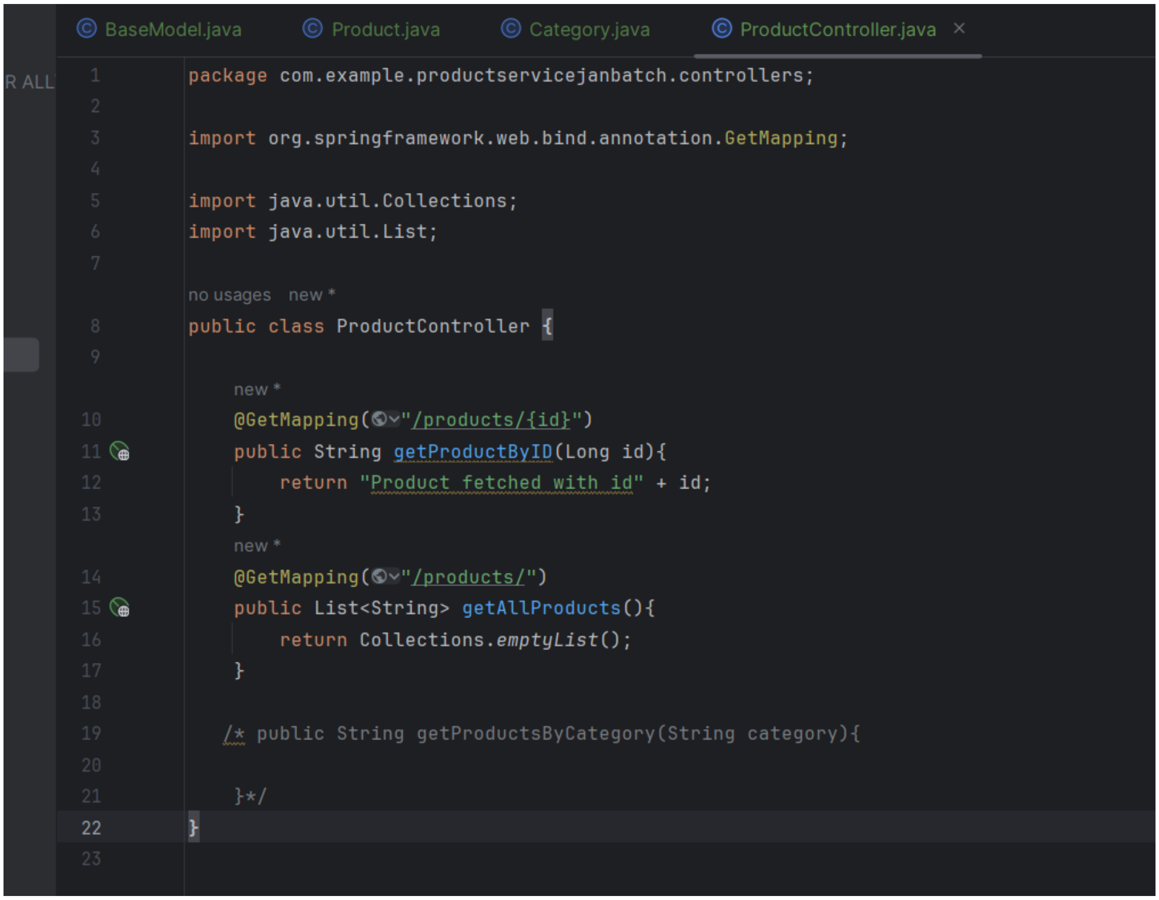
This is again an endpoint. So I need mapping getmapping.. I will need /products again.

```
@GetMapping("/products/")
public List<String> getAllProducts(){
    return Collections.emptyList();
}
```

All api naming convention me they return list of entities… so plural.

We do operation on multiple products.. when I search id = x I search on multiple products. Hence path of entity in plural format.

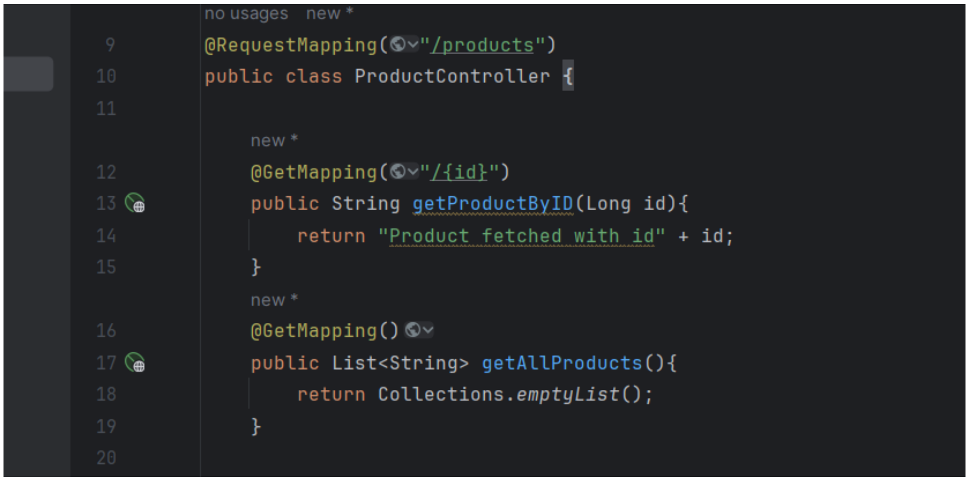
We don’t worry how **dispatcher** will map it to controller.. we will directly call the path.. the mapping of domain to this function will be taking care by dispatcher.



```java
package com.example.productservicejanbatch.controllers;

import org.springframework.web.bind.annotation.GetMapping;

import java.util.Collections;
import java.util.List;

public class ProductController {

    @GetMapping("/products/{id}")
    public String getProductByID(Long id){
        return "Product fetched with id" + id;
    }
    @GetMapping("/products/")
    public List<String> getAllProducts(){
        return Collections.emptyList();
    }

   /* public String getProductsByCategory(String category){

    }*/
}

```

Now as we know this controller will have mapping over products we can add requestmapping(/products)… and we don’t need to put product and remove the common at product controller.

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



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

    @GetMapping("/{id}")
    public String getProductByID(Long id){
        return "Product fetched with id" + id;
    }
    @GetMapping()
    public List<String> getAllProducts(){
        return Collections.emptyList();
    }

```

Request mapping is at class level, to define the common path. Extract out common path…

This is a controller.. who will create its object. I will have to tell spring… by saying @Restcontroller.

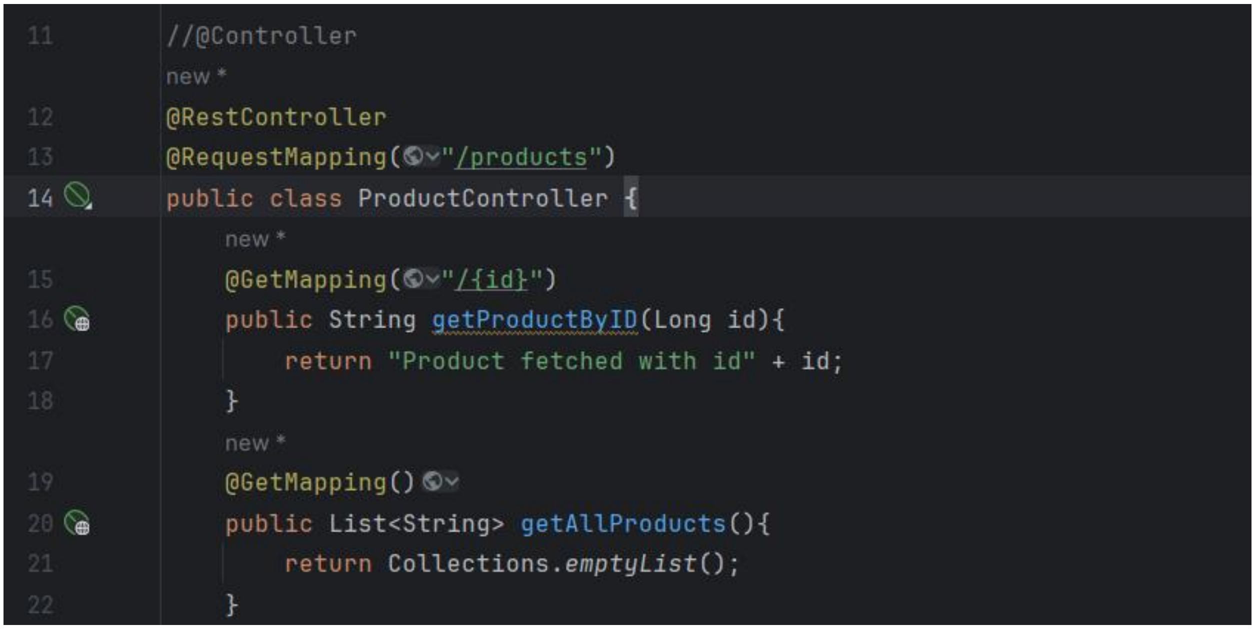
```
@Controller
@RestController
@RequestMapping("/products")
public class ProductController {
```

Another is @Controller… we use @REstcontroller. As we are saying that this controller is based out of rest guidline. Whatever compile time validation of a rest contoller, please do that.

Restcontroller will internally have a response body.. this RB wil do some additional validation that this controller is flowing rest guidelines. Please do the validation which are applicable in rest based controller.

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    @AliasFor(
        annotation = Controller.class
    )
    String value() default "";
}
```

So we will only use @RestController.



```
//@Controller
new *
@RestController
@RequestMapping("/products")
public class ProductController {
    new *
    @GetMapping("/{id}")
    public String getProductByID(Long id){
        return "Product fetched with id" + id;
    }
    new *
    @GetMapping()
    public List<String> getAllProducts(){
        return Collections.emptyList();
    }
```

What is Controller?

A Controller is a Java class that is used to handle HTTP requests in a Spring or Spring Boot application. It is responsible for receiving incoming requests from a client, processing them, and returning an appropriate response. A Controller typically returns a View object, which is responsible for rendering a web page that can be viewed by the user.

What about RestController?

On the other hand, a RestController is a special type of Controller that is used to handle RESTful requests in a Spring or Spring Boot application. It is similar to a Controller in that it receives incoming requests from a client, processes them, and returns an appropriate response. However, instead of returning a View object, a RestController returns a ResponseBody object, which contains the data that is sent back to the client in a format such as JSON or XML.

What is the difference between two?

The main difference between a Controller and a RestController is in the type of response they return. A Controller returns a View object that is responsible for rendering a web page that can be viewed by the user. A RestController, on the other hand, returns a ResponseBody object that contains data in a format such as JSON or XML that can be consumed by other applications.

Another key difference between the two is in the annotations used to create them. A Controller is typically created using the @Controller annotation, while a RestController is created using the @RestController annotation. The @RestController annotation is a combination of the @Controller and @ResponseBody annotations, which are used to create a Controller that returns a ResponseBody object.

The primary difference between a Controller and a RestController in Spring and Spring Boot is in the type of response they return. A Controller returns a View object that is responsible for rendering a web page, while a RestController returns a ResponseBody object that contains data in a format such as JSON or XML. When building a web application that requires both web pages and RESTful APIs, it is important to understand the difference between these two components and use them appropriately.

If you’ve made it this far, thank you for reading and if you enjoyed reading this post, consider dropping a clap and a follow. I post articles which are interesting and beginner-friendly on Java,Python and Machine Learning so that everyone can understand.

All the api will be single responsibility.

The principle that each API should have a single responsibility is known as the Single Responsibility Principle (SRP), which is one of the SOLID principles of object-oriented design. However, when it comes to APIs, the idea is still relevant, although it might be expressed a bit differently.

In the context of APIs, the Single Responsibility Principle suggests that each API endpoint or function should have a clear and focused purpose. It should do one thing and do it well. This makes the API more maintainable, understandable, and flexible.

Here are a few reasons why having a single responsibility for each API is beneficial:

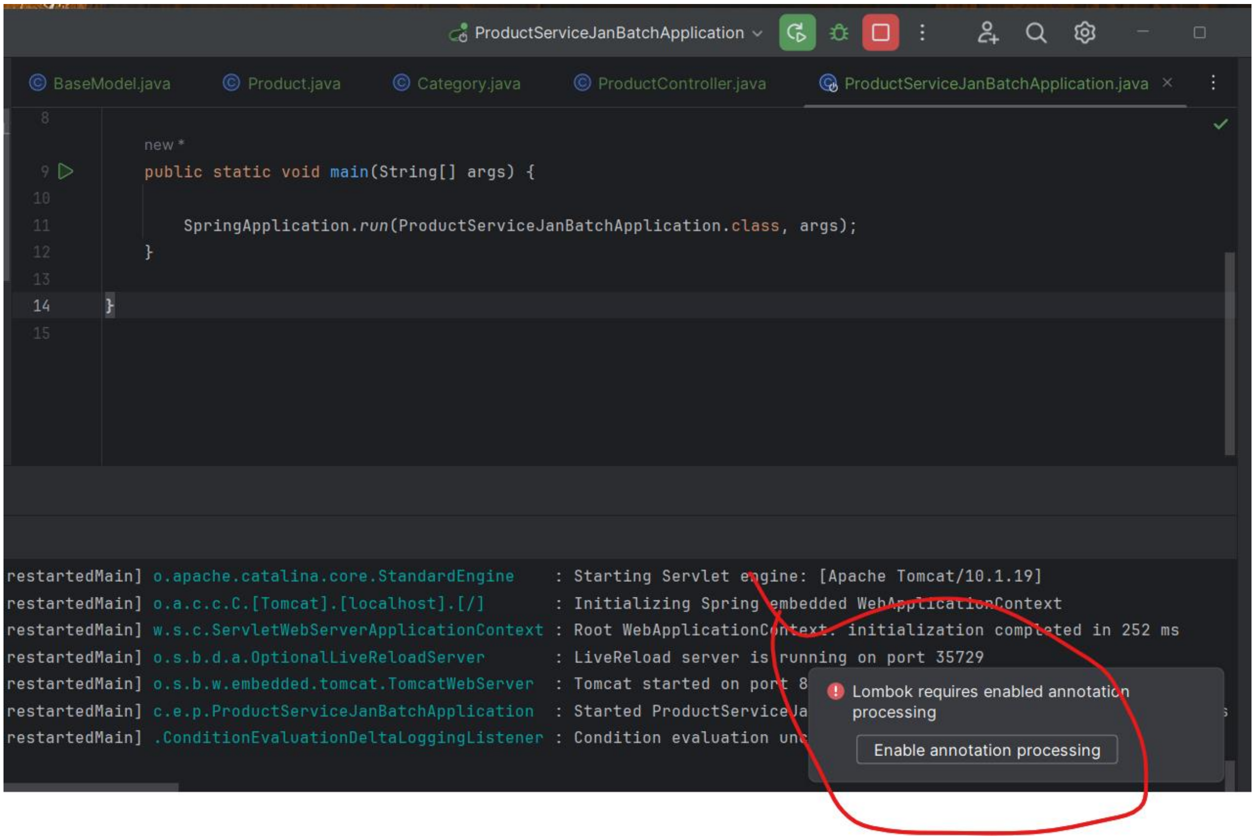
1. **Clarity and Understanding:** When APIs have a single responsibility, it becomes easier for developers to understand their purpose and use them correctly. This promotes better code readability and reduces the chances of misuse.
2. **Maintainability:** APIs with a single responsibility are typically easier to maintain. Changes to the functionality of one endpoint or function are less likely to impact others, reducing the risk of introducing bugs unintentionally.
3. **Flexibility and Reusability:** Single-responsibility APIs are more modular and can be reused in different contexts. Developers can compose these small, focused APIs to create more complex functionality while keeping each piece simple and manageable.
4. **Testing:** It is easier to write unit tests for APIs with a single responsibility. Testing becomes more straightforward, as you can isolate and verify the behavior of each API independently.
5. **Scalability:** When each API has a well-defined and narrow responsibility, scaling becomes more manageable. You can scale specific components of your system without affecting others unnecessarily.

```
@RestController
@RequestMapping("/products")
public class ProductController {
    @GetMapping("/{id}")
    public String getProductByID(Long id){
        return "Product fetched with id" + id;
    }
    @GetMapping()
    public List<String> getAllProducts(){
        return Collections.emptyList();
    }
```

In getProduct we have Long id… we will get an id from input. The function don’t know that there will be a input id… so we need to have a pathvariable.. please match the path variable id… to long id.

```
@GetMapping("/{id}")
public String getProductByID(@PathVariable("id") Long id){
    return "Product fetched with id" + id;
}
```

So lets run and see:

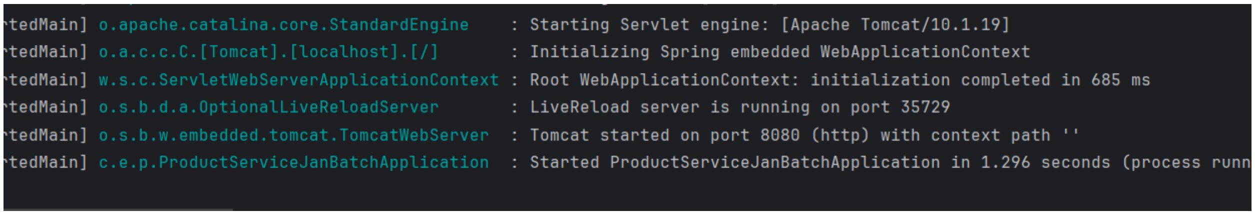

ProductServiceJanBatchApplication
BaseModel.java
Product.java
Category.java
ProductController.java
ProductServiceJanBatchApplication.java
new *
Enable annotation processing

```
    public static void main(String[] args) {

        SpringApplication.run(ProductServiceJanBatchApplication.class, args);
    }

}
```

```
restartedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 252 ms
restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8
restartedMain] c.e.p.ProductServiceJanBatchApplication  : Started ProductServiceJa
restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unc
```

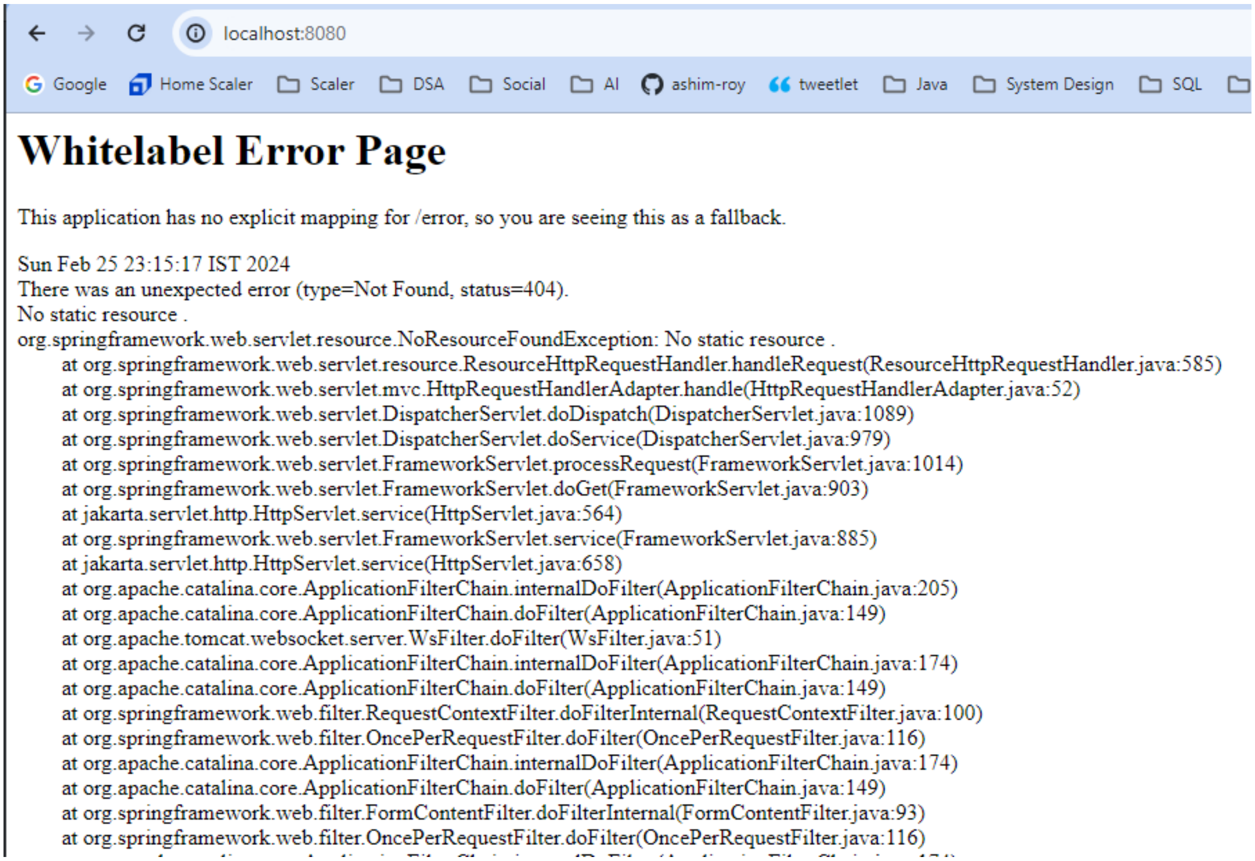
Lombok requires enabled annotation processing

This is normal. Just enable.



```
tedMain] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
tedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
tedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 685 ms
tedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
tedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path ''
tedMain] c.e.p.ProductServiceJanBatchApplication  : Started ProductServiceJanBatchApplication in 1.296 seconds (process runn
```

http://localhost:8080/



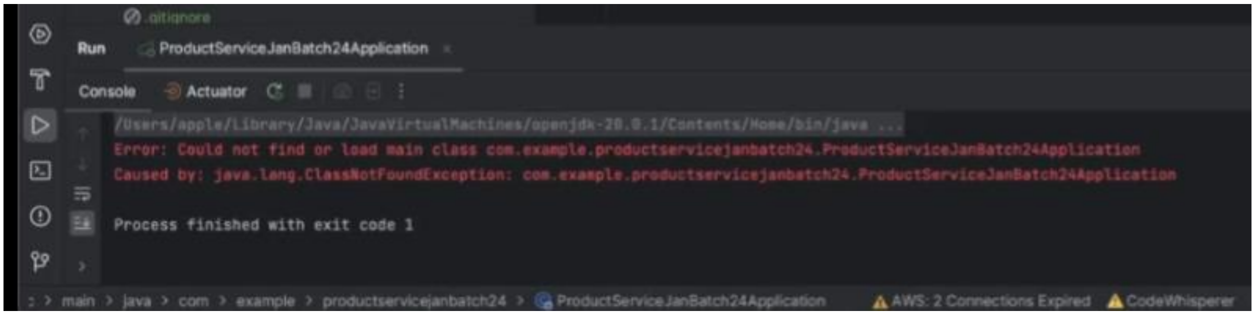
Whitelabel Error Page

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

```
Sun Feb 25 23:15:17 IST 2024
There was an unexpected error (type=Not Found, status=404).
No static resource .
org.springframework.web.servlet.resource.NoResourceFoundException: No static resource .
	at org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:585)
	at org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter.handle(HttpRequestHandlerAdapter.java:52)
	at org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1089)
	at org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:979)
	at org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1014)
	at org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:903)
	at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:564)
	at org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:885)
	at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:658)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:205)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:149)
	at org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:174)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:149)
	at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
	at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:116)
	at org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:174)
	at org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:149)
	at org.springframework.web.filter.FormContentFilter.doFilterInternal(FormContentFilter.java:93)
	at org.springframework.web.filter.OncePerRequestFilter.doFilter(OncePerRequestFilter.java:116)
```

The context path is the part of the URL that comes after the domain name and before the servlet path. It represents the base URL pattern to which the servlets are mapped.

But if u get such error: could not load or find main class…

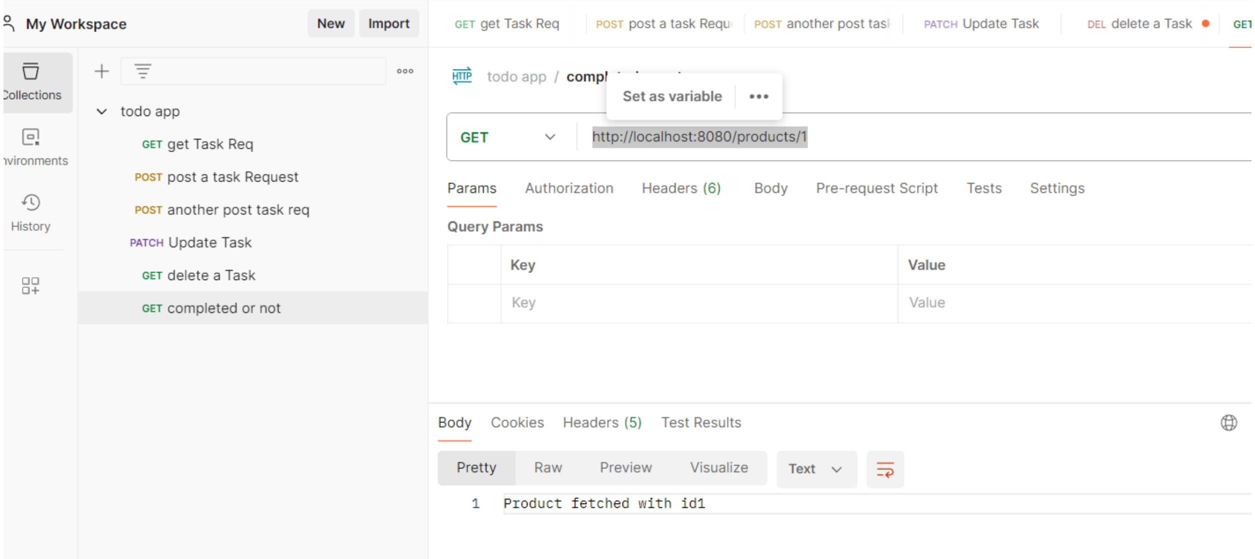

.gitignore
Run
ProductServiceJanBatch24Application
Console
Actuator
main > java > com > example > productservicejanbatch24 > ProductServiceJanBatch24Application
AWS: 2 Connections Expired
CodeWhisperer

```
/Users/apple/Library/Java/JavaVirtualMachines/openjdk-20.0.1/Contents/Home/bin/java ...
Error: Could not find or load main class com.example.productservicejanbatch24.ProductServiceJanBatch24Application
Caused by: java.lang.ClassNotFoundException: com.example.productservicejanbatch24.ProductServiceJanBatch24Application

Process finished with exit code 1
```

Just go to : the directory and remove .idea file.. it may get corrupted. Clean it up. Restart the intellij.

We can do it via postman.

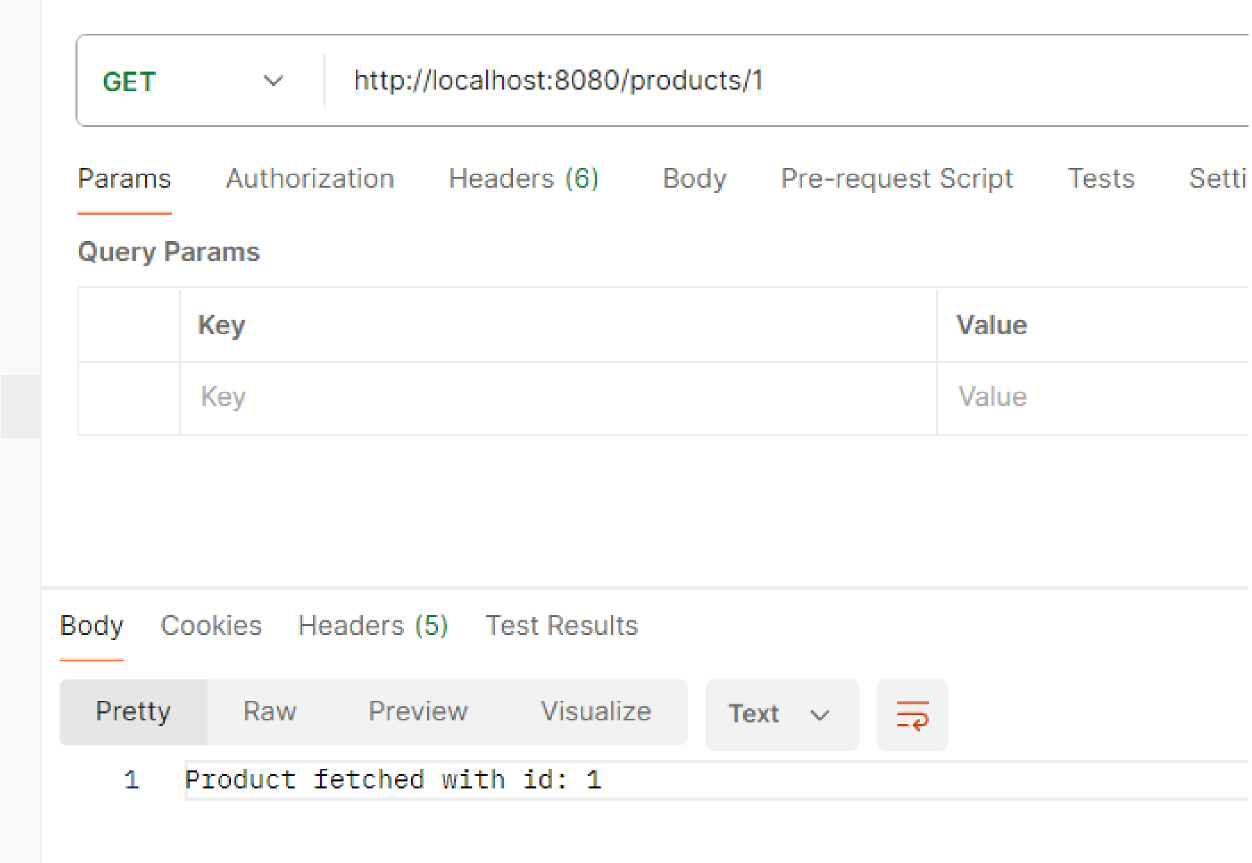
http://localhost:8080/products/1



GET http://localhost:8080/products/1

```
1   Product fetched with id1
```

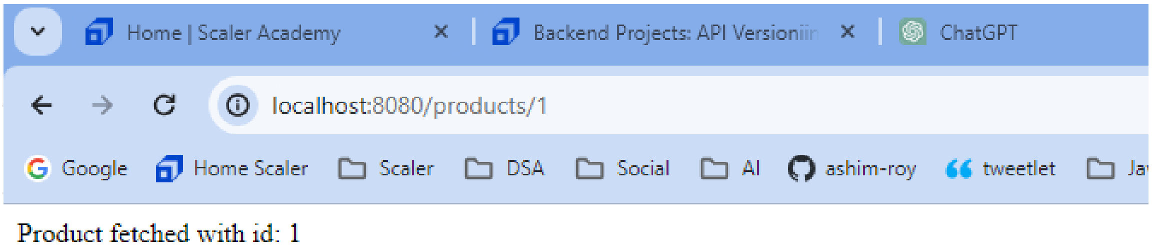
Change in code for space and rerun:



```
GET http://localhost:8080/products/1
```

```
Product fetched with id: 1
```

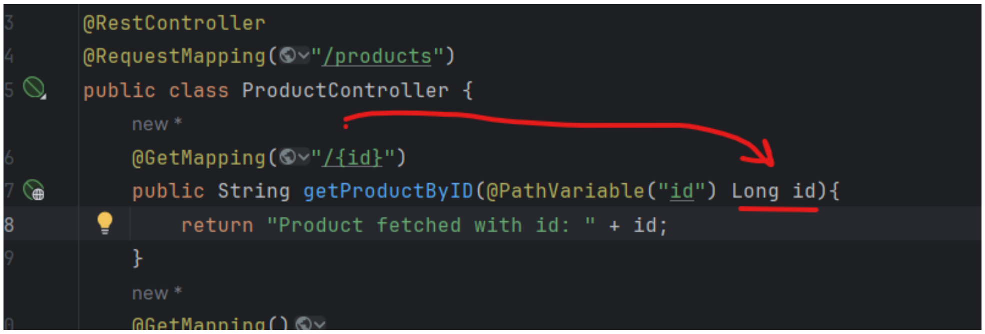
The get calls we can do via crome as well.



```
localhost:8080/products/1
```

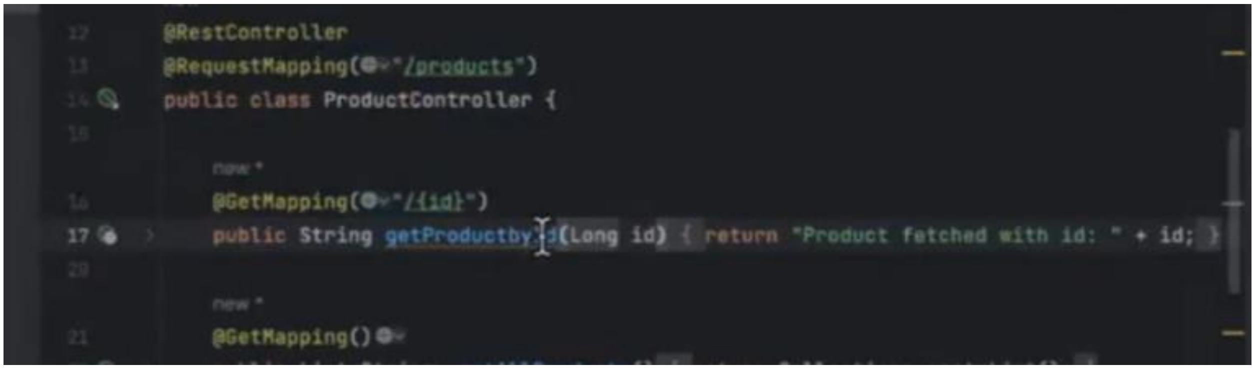
```
Product fetched with id: 1
```

If we havnt add the path variable.. the id data will never get mapped to the id below



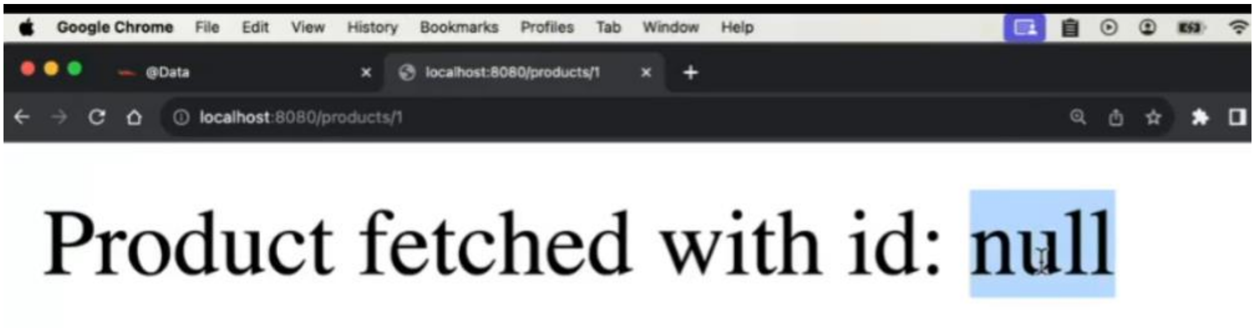
```java
@RestController
@RequestMapping("/products")
public class ProductController {
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id){
        return "Product fetched with id: " + id;
    }
    @GetMapping()
```

If I don’t specify the path variable it will take default value of Long = null.



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

    @GetMapping("/{id}")
    public String getProductbyId(Long id) { return "Product fetched with id: " + id; }

    @GetMapping()
```

Here in URL I am passing id=1. But its not mapping to the request / input parameter..



Product fetched with id: null

--

After controller we will create service..

Product controller will be hosting of the logic to Service..

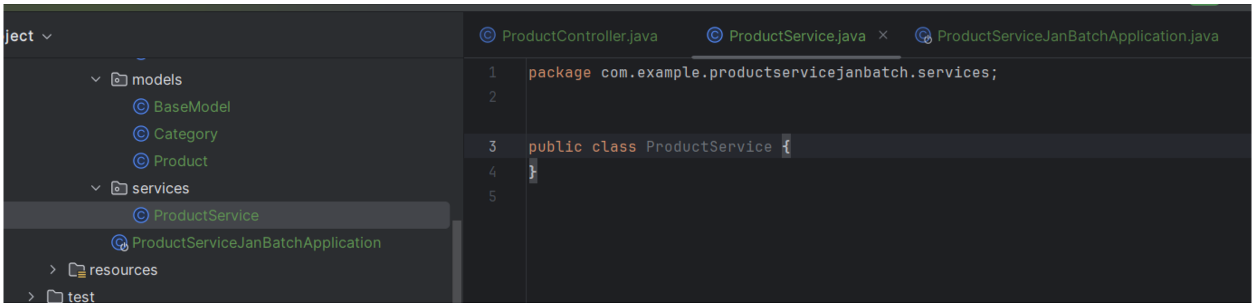
ProductContoller depend on product service. So PS will be class or interface…? Why?

Ans – class.

On general level a class is fine to have it here… generally u will have single logic. Its violates the D in SOLID. Dependency inversion. 2 concreate class wil not depend on each other.

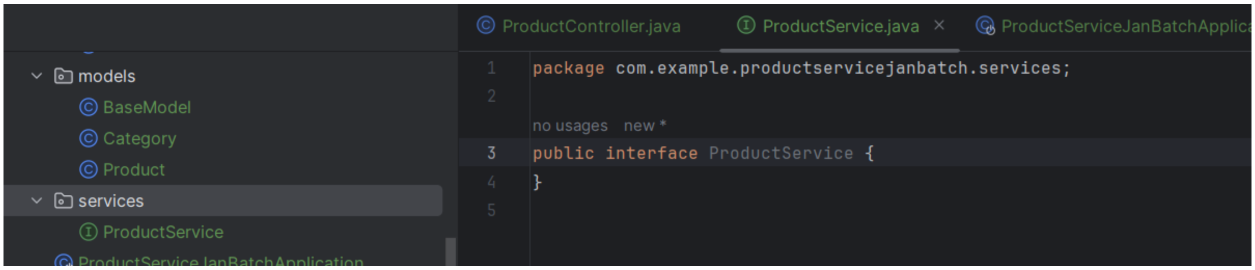
Ideally it should be an interface. So 2morrow if we change the logic of the service it will b easier..

In this class we say it interface.



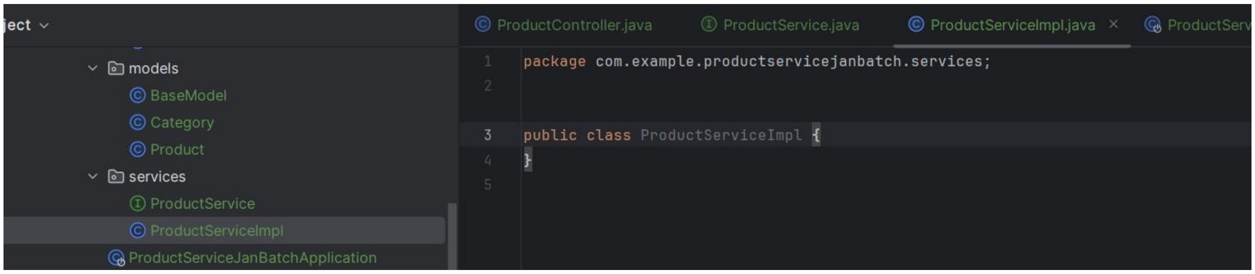
```java
package com.example.productservicejanbatch.services;


public class ProductService {
}
```

So change it to interface.



```java
package com.example.productservicejanbatch.services;

public interface ProductService {
}
```

Then we also need its implementation:



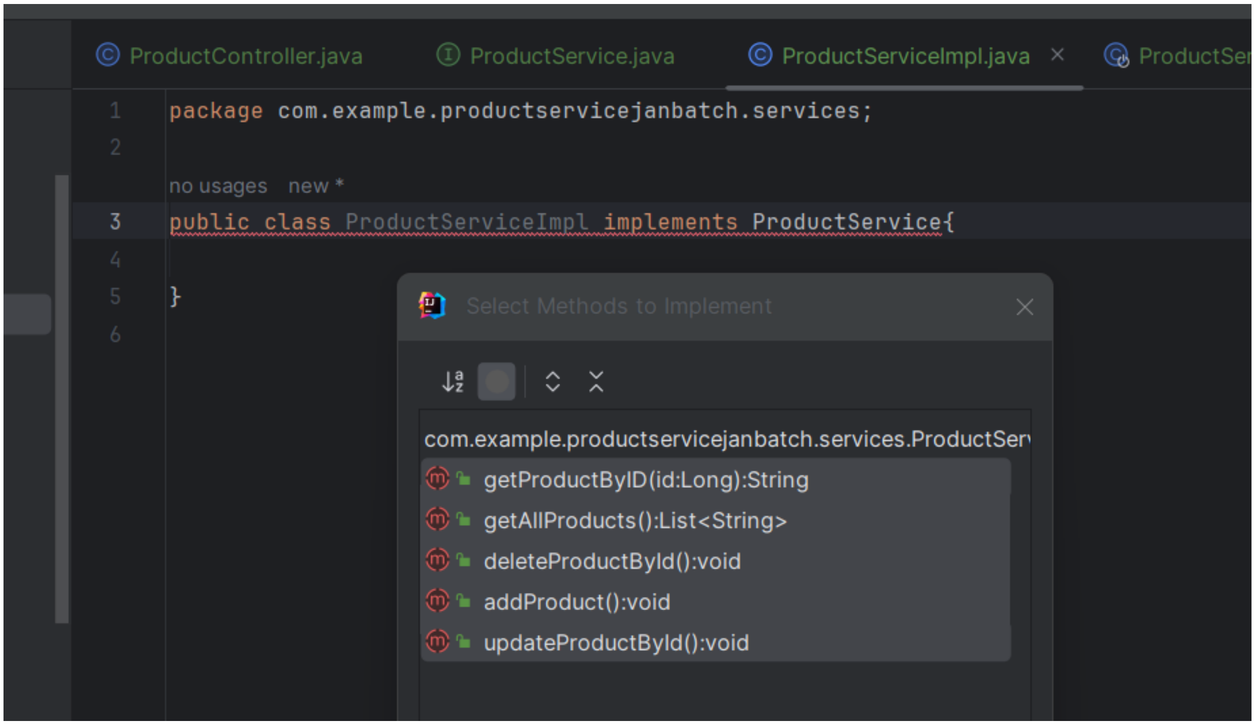
```java
package com.example.productservicejanbatch.services;


public class ProductServiceImpl {
}
```

They wanna follow DI. They will have an IF and implementation. Also naming can be IProductService and ProductService class.

The IF productservice will have the functions all the function which controller need to have..

```java
public interface ProductService {
    String getProductByID(Long id);
    List<String> getAllProducts();

    void deleteProductById();
    void addProduct();
    void updateProductById();
}
```

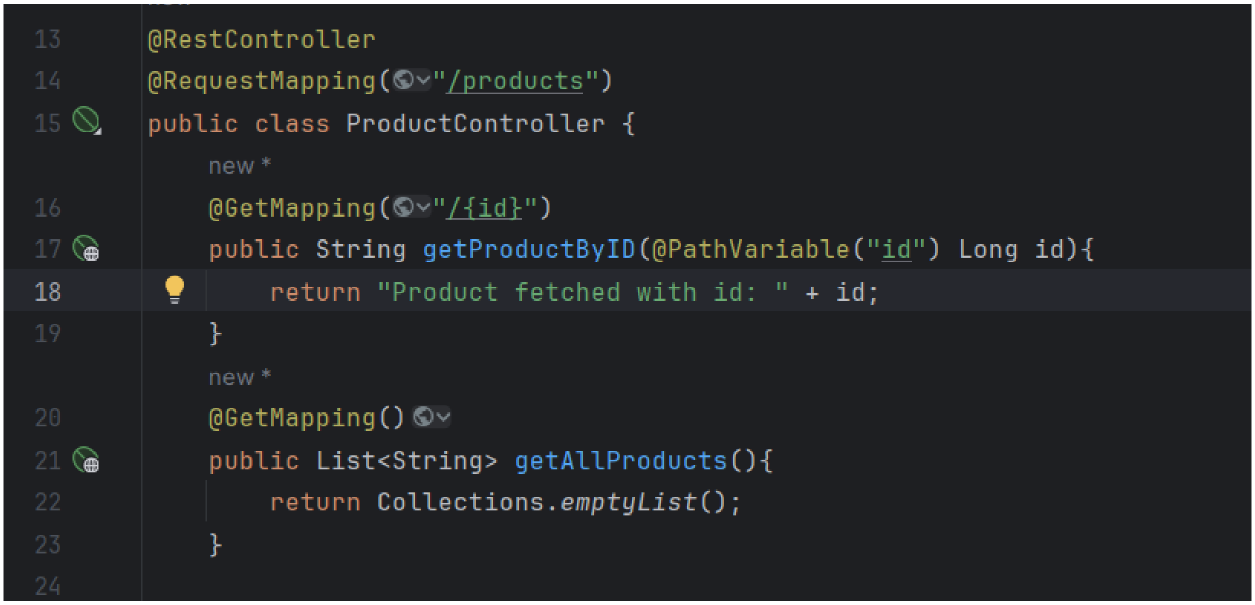
in productServiceImpl…. See for implements methods:



```java
package com.example.productservicejanbatch.services;

public class ProductServiceImpl implements ProductService{

}
```

```java
package com.example.productservicejanbatch.services;

import java.util.List;

public class ProductServiceImpl implements ProductService{

    @Override
    public String getProductByID(Long id) {
        return "product fetched from service, id: " + id;
    }
```

lets look at controller.



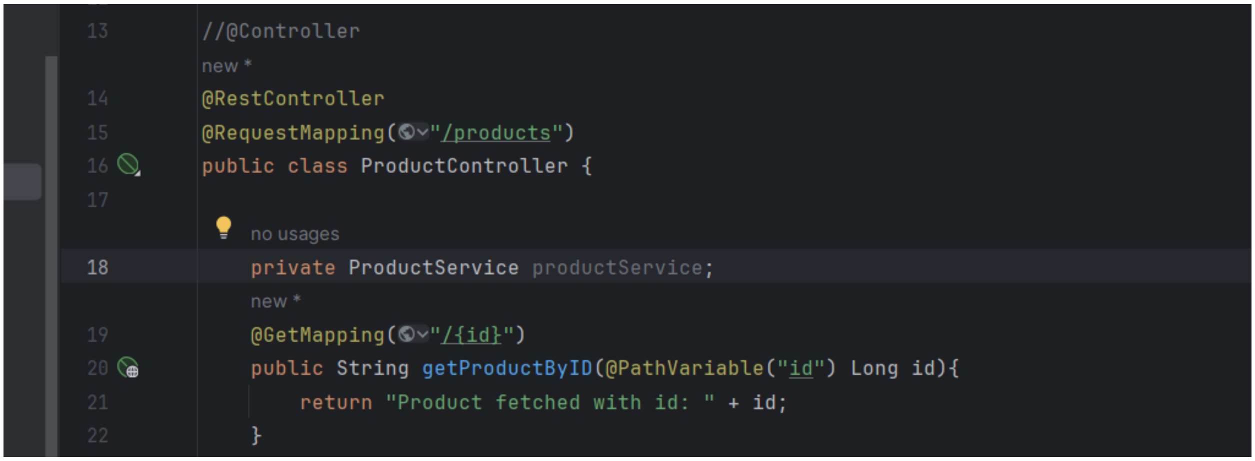
```
@RestController
@RequestMapping("/products")
public class ProductController {
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id){
        return "Product fetched with id: " + id;
    }
    @GetMapping()
    public List<String> getAllProducts(){
        return Collections.emptyList();
    }

```

The controller will get the data from?

It will reply on product service to get the data.

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

    private ProductService productService;
```

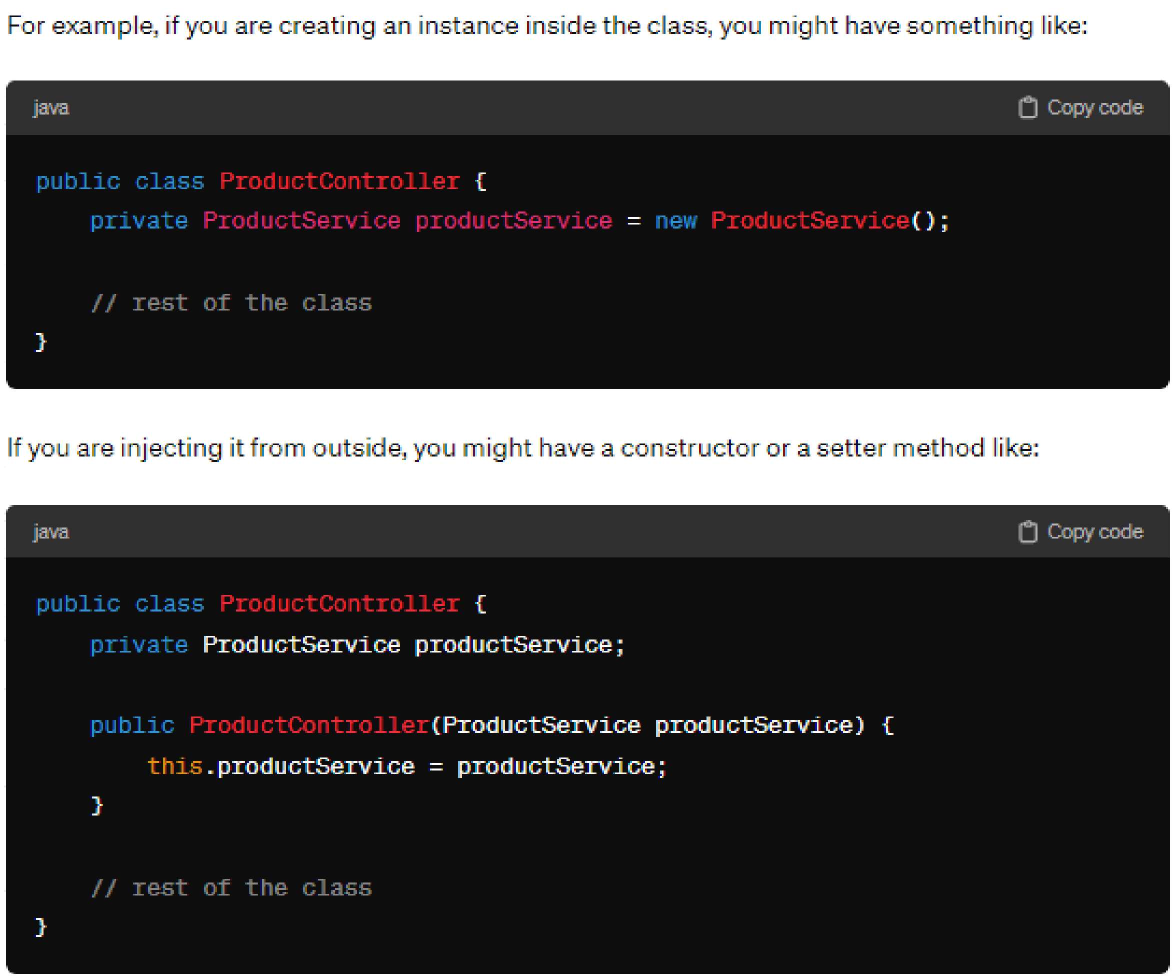
I need to inject the product service via contructor injection.



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

    private ProductService productService;
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id){
        return "Product fetched with id: " + id;
    }
```

ou are declaring a private field named `productService` of type `ProductService` in the `ProductController` class. This is not creating an object of `ProductService` or injecting it directly.

To use this field, you would typically need to either instantiate an object of `ProductService` within the `ProductController` class or inject it from outside, often through a constructor or a setter method. Without additional code, it's not clear whether you are creating an instance inside the class or injecting it from external code.



For example, if you are creating an instance inside the class, you might have something like:

java Copy code

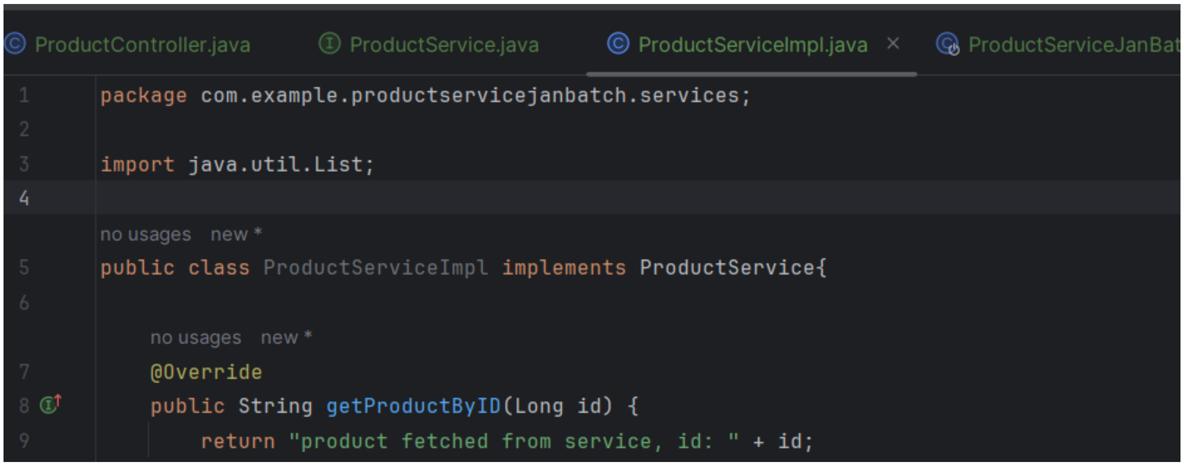
```java
public class ProductController {
    private ProductService productService = new ProductService();

    // rest of the class
}
```

If you are injecting it from outside, you might have a constructor or a setter method like:

java Copy code

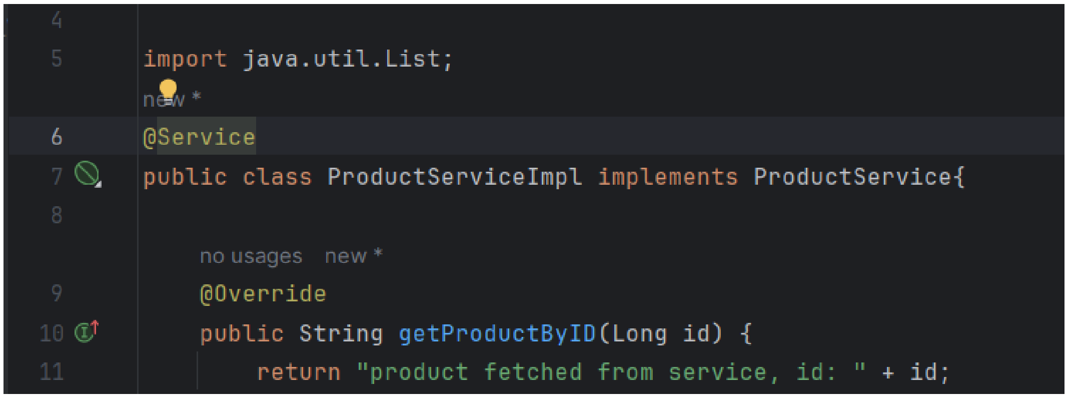
```java
public class ProductController {
    private ProductService productService;

    public ProductController(ProductService productService) {
        this.productService = productService;
    }

    // rest of the class
}
```

Lets go to productserviceImpl..



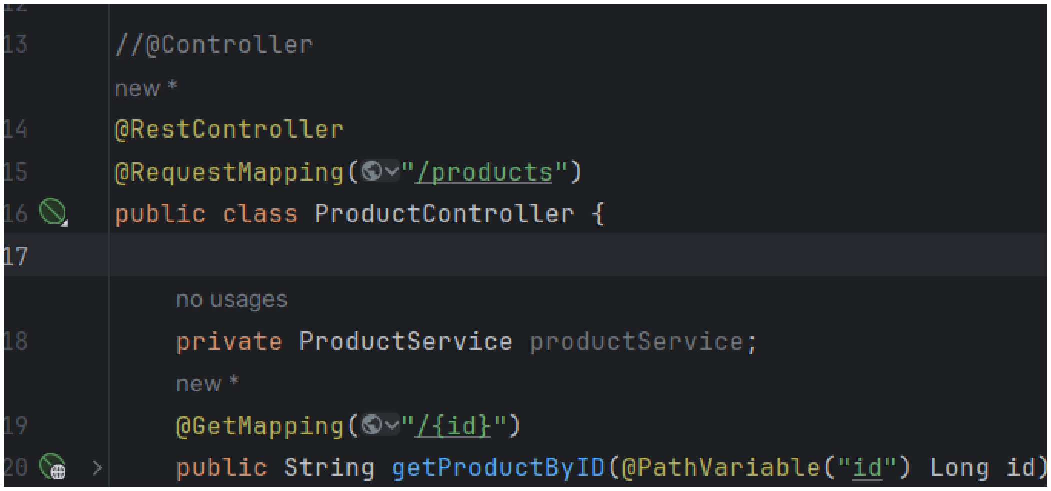
```
package com.example.productservicejanbatch.services;

import java.util.List;

public class ProductServiceImpl implements ProductService{

    @Override
    public String getProductByID(Long id) {
        return "product fetched from service, id: " + id;
```

We haven’t told spring that this object need to be created. So I can tell by @ service:

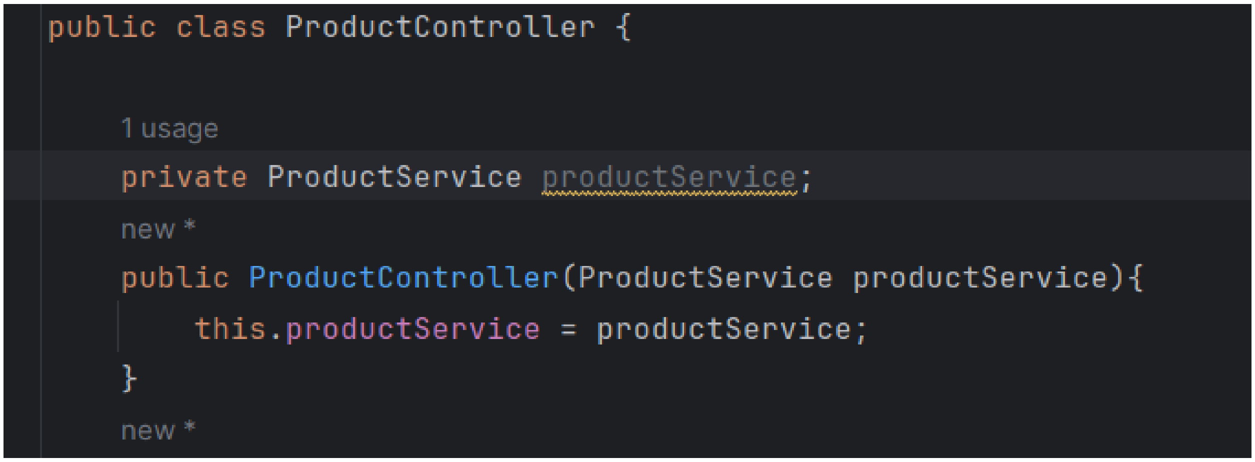


```

import java.util.List;

@Service
public class ProductServiceImpl implements ProductService{

    @Override
    public String getProductByID(Long id) {
        return "product fetched from service, id: " + id;
```

Now in productController I will inject the productService..



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

    private ProductService productService;
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id)
```



```
public class ProductController {

    1 usage
    private ProductService productService;
    new *
    public ProductController(ProductService productService){
        this.productService = productService;
    }
    new *
```

Here I am telling Spring these:

Create a object of product service:

```
@Service
public class ProductServiceImpl implements ProductService{
```

In ProductController I am saying that to create the ProductController, we need to have a object of product service. How can we get the product service here. In PC constructor we will inject the product service. Spring will inject here. Sping has created the object and then Autowire the object in the constructor for creating the object of product controller.

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

    private ProductService productService;
    @Autowired
    public ProductController(ProductService productService){
        this.productService = productService;
    }
```

Why spring will create the object of product controller?

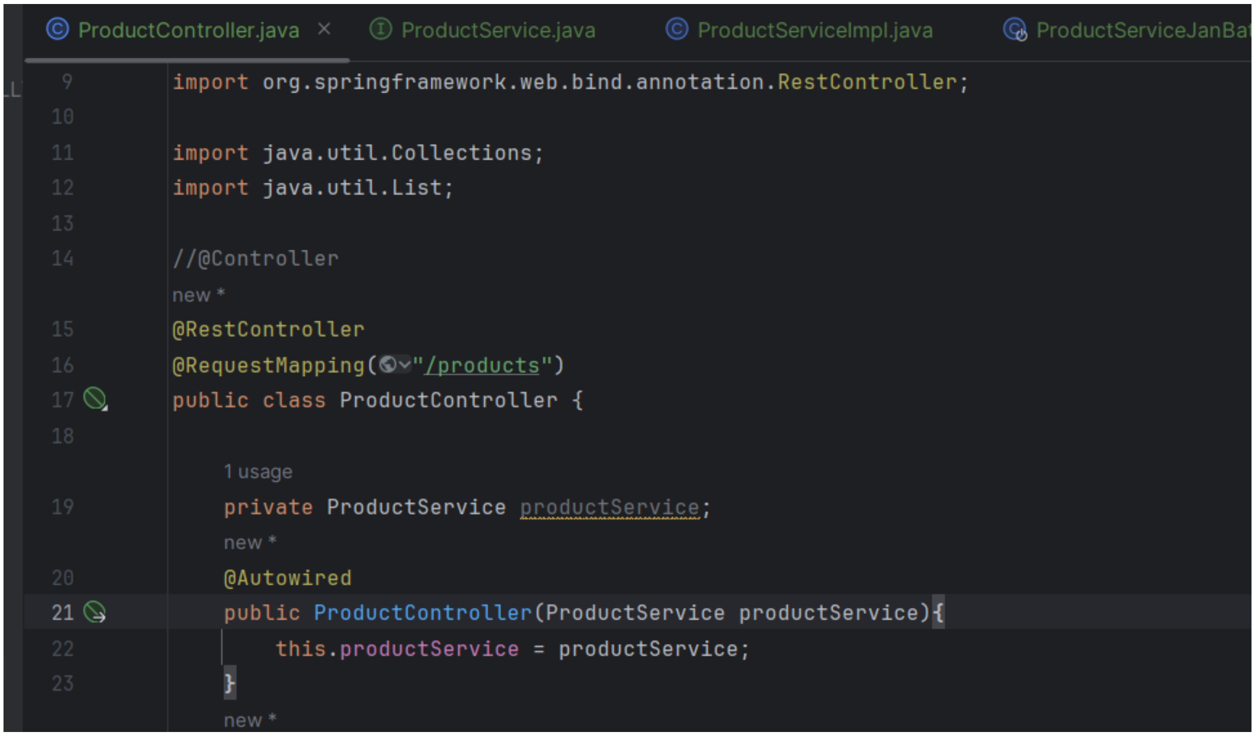
Because we annotated with @RestController.

`@RestController` annotation on the **`ProductController`** class. This annotation is part of the Spring MVC framework, which is designed for building RESTful web services.

When you annotate a class with **`@RestController`**, Spring will automatically create an instance (object) of that class when it initializes the application context. his instance is then managed by the Spring IoC (Inversion of Control) container. In the case of your **`ProductController`**, Spring will instantiate it and manage its lifecycle.

In your constructor, you are using the `@Autowired` annotation on the constructor parameter. This annotation is a way to tell Spring to inject the required dependencies, in this case, the `ProductService` dependency, when creating the `ProductController` bean. So, when Spring creates the `ProductController` bean, it will look for a bean of type `ProductService` in its context and inject it into the constructor.

Spring is responsible for creating and managing the objects (beans) annotated with `@RestController` or other relevant annotations. It uses dependency injection to provide the necessary dependencies to these beans, as configured through annotations like `@Autowired`. This is a fundamental aspect of the Inversion of Control (IoC) principle in the Spring framework.



```
import org.springframework.web.bind.annotation.RestController;

import java.util.Collections;
import java.util.List;

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

    private ProductService productService;
    @Autowired
    public ProductController(ProductService productService){
        this.productService = productService;
    }
```

Product service obj is created and autowired the ps in product controller using @restcontroller.

productservice is Interface.. how obj will be created?

It will use the implementation o fteh interface, which is

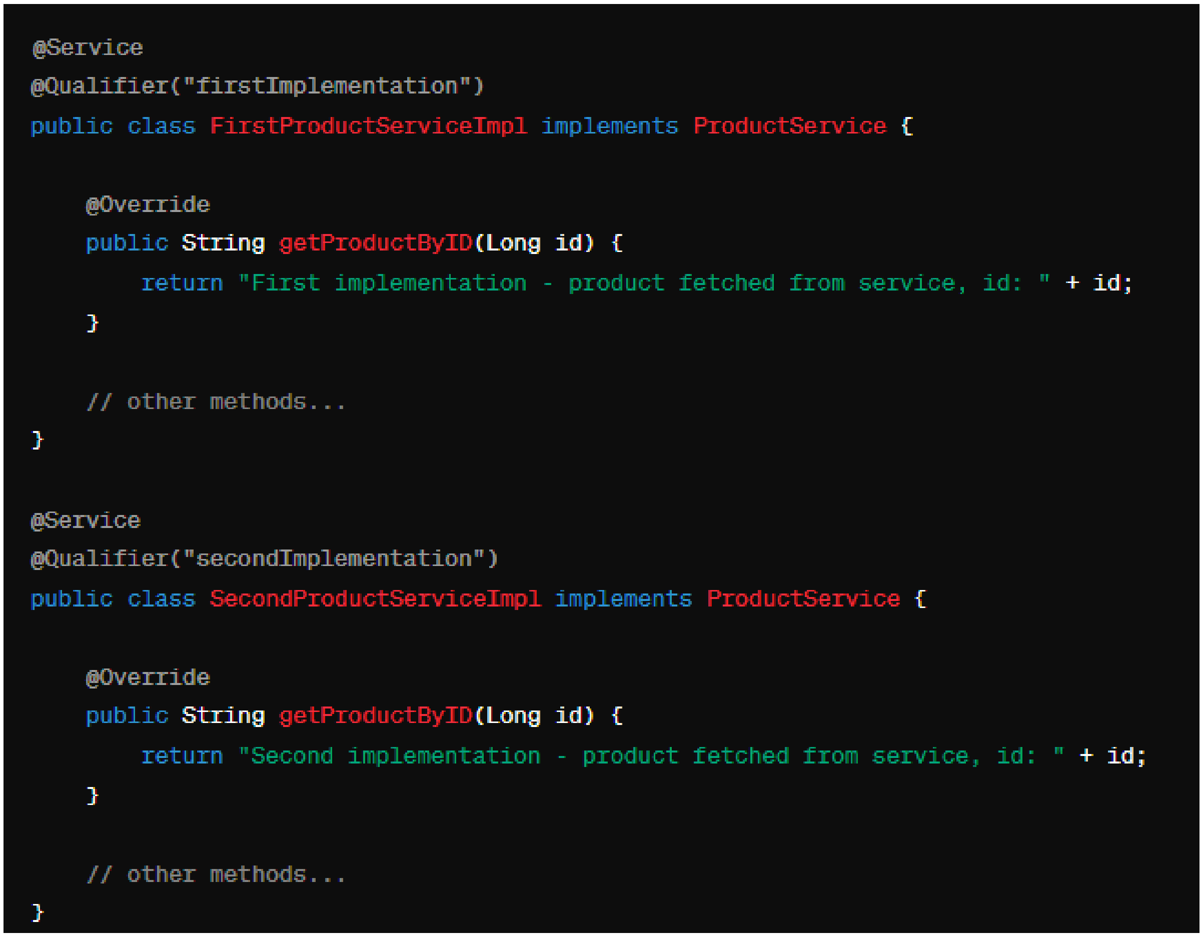
```
@Service
public class ProductServiceImpl implements ProductService{
```

We put annotation at implementation not in IF. We can refer the implementation by its parent.

if we had multiple impls? It will come..

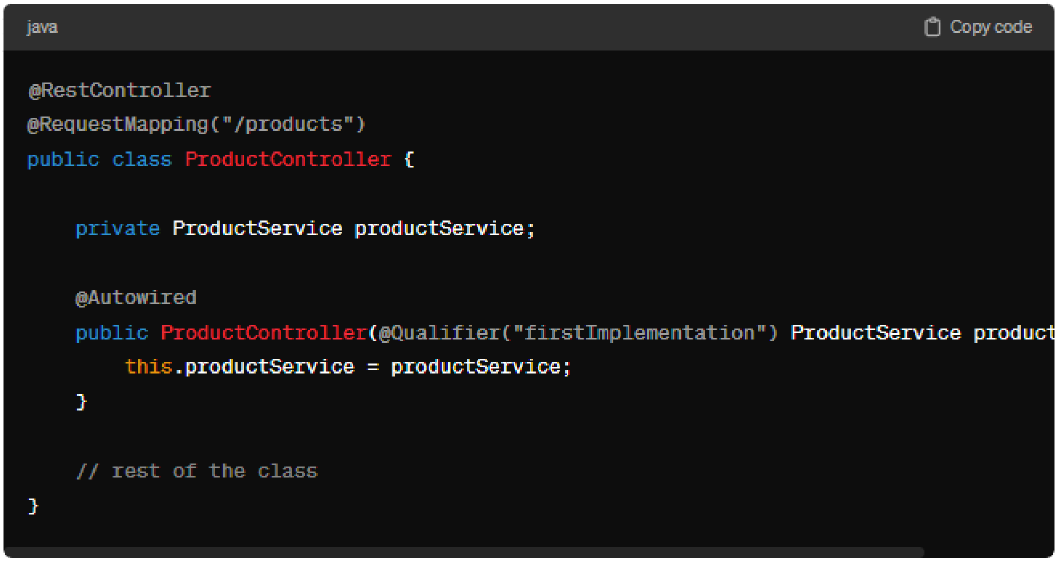
In Spring, when you have multiple implementations of an interface, you can use the `@Qualifier` annotation to specify which implementation should be injected. The `@Qualifier` annotation is used in conjunction with the `@Autowired` annotation to provide more fine-grained control over the dependency injection.

Let's assume you have two implementations of the `ProductService` interface:



```
@Service
@Qualifier("firstImplementation")
public class FirstProductServiceImpl implements ProductService {

    @Override
    public String getProductByID(Long id) {
        return "First implementation - product fetched from service, id: " + id;
    }

    // other methods...
}

@Service
@Qualifier("secondImplementation")
public class SecondProductServiceImpl implements ProductService {

    @Override
    public String getProductByID(Long id) {
        return "Second implementation - product fetched from service, id: " + id;
    }

    // other methods...
}
```

Now, in your `ProductController`, you can use `@Qualifier` to specify which implementation you want to inject:



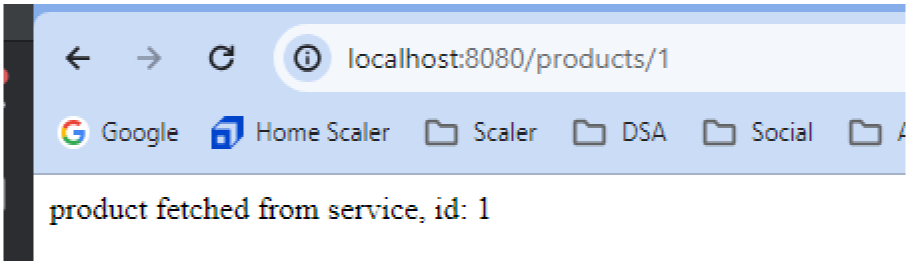
```java
@RestController
@RequestMapping("/products")
public class ProductController {

    private ProductService productService;

    @Autowired
    public ProductController(@Qualifier("firstImplementation") ProductService product
        this.productService = productService;
    }

    // rest of the class
}
```

By using **`@Qualifier("firstImplementation")`**, you are telling Spring to inject the bean with the name "firstImplementation" when creating the **`ProductController`**. If you had multiple implementations and you didn't specify the **`@Qualifier`**, Spring would throw an exception because it wouldn't know which bean to inject.

In summary, when you have multiple implementations of an interface and want to inject a specific one, use **`@Qualifier`** to specify the bean name or qualifier associated with the desired implementation.

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

    private ProductService productService;
    @Autowired
    public ProductController(ProductService productService){
        this.productService = productService;
    }
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id){

        return productService.getProductByID(id);
    }
```

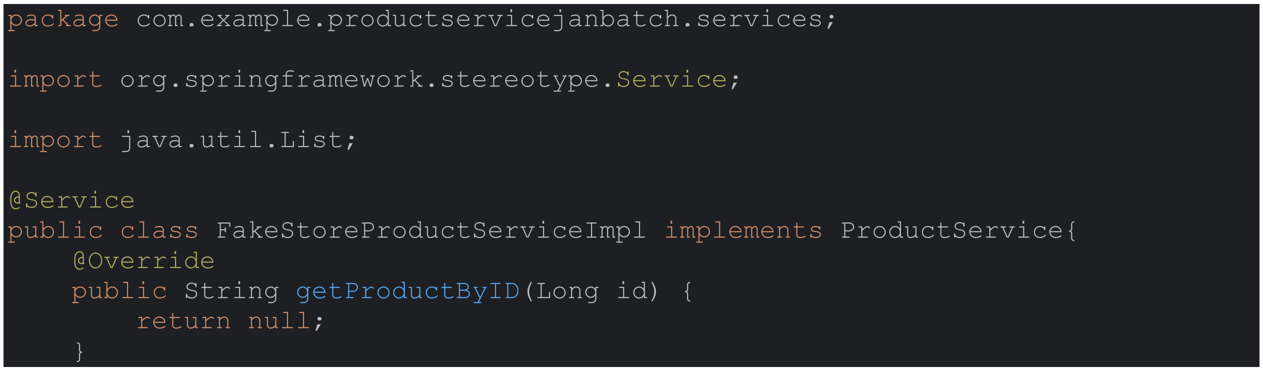
http://localhost:8080/products/1



product fetched from service, id: 1

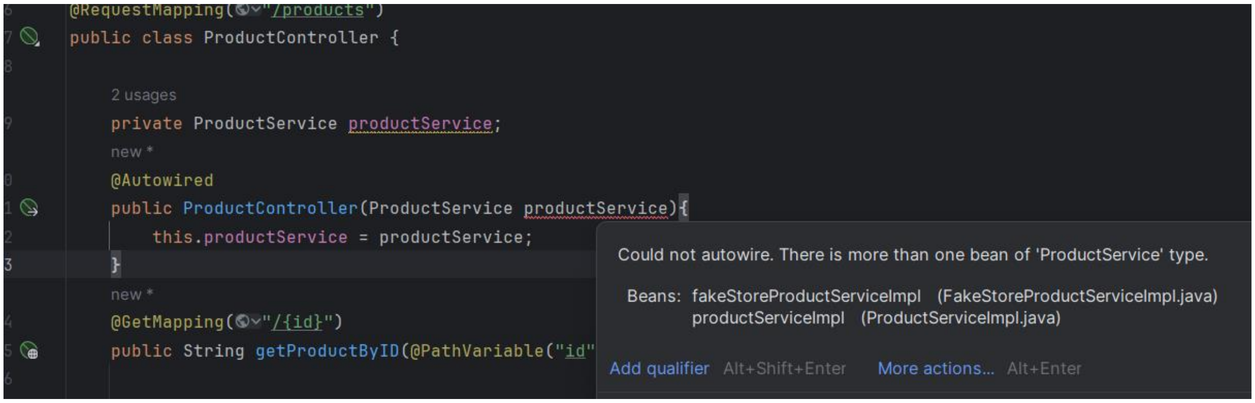
No 2 class should depend on each other… depend via interface.

Now will have 2 implementation



```
package com.example.productservicejanbatch.services;

import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class FakeStoreProductServiceImpl implements ProductService{
    @Override
    public String getProductByID(Long id) {
        return null;
    }
```

I can see error of couldn’t autowire

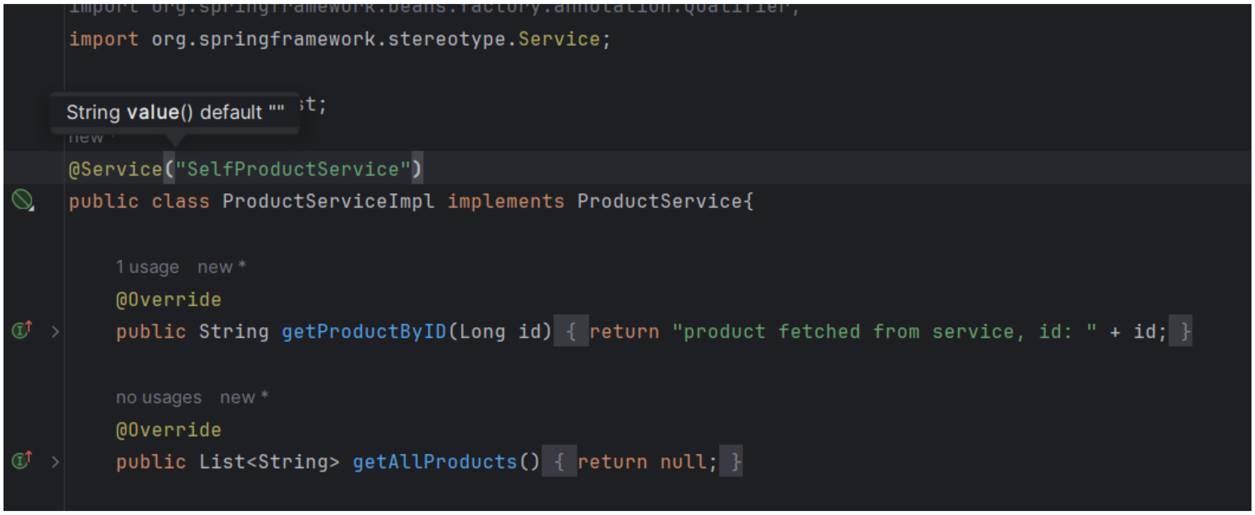


```
@RequestMapping("/products")
public class ProductController {

    2 usages
    private ProductService productService;
    new *
    @Autowired
    public ProductController(ProductService productService){
        this.productService = productService;
    }
    new *
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id"
```

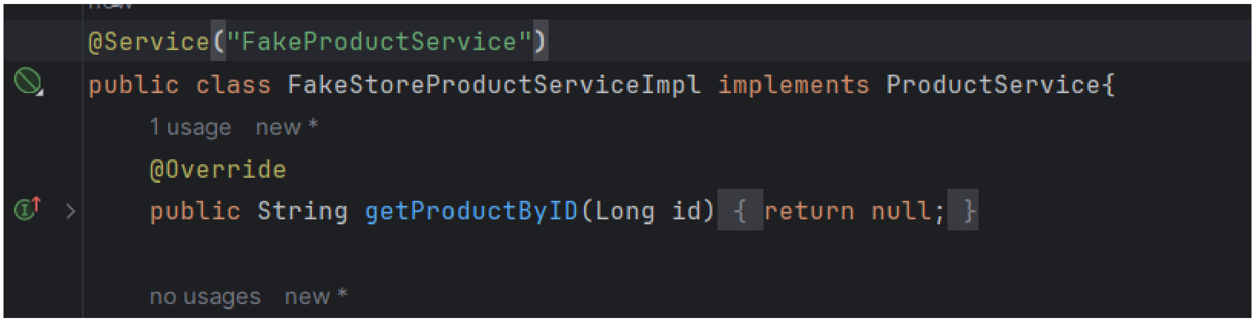
Could not autowire. There is more than one bean of 'ProductService' type.

Beans: fakeStoreProductServiceImpl (FakeStoreProductServiceImpl.java)
productServiceImpl (ProductServiceImpl.java)

There are more than 1 obj of product service.. it suggest me to add a qualifier..



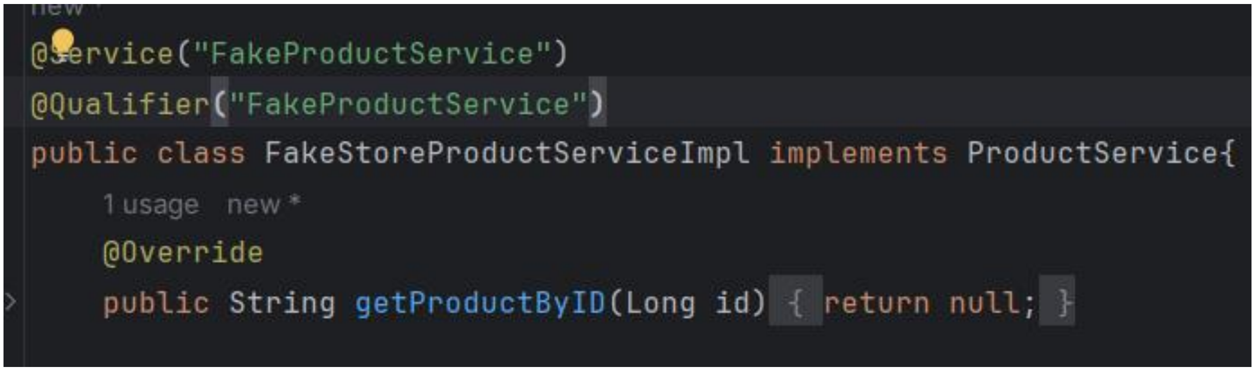
```
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

                                     t;
new *
@Service("SelfProductService")
public class ProductServiceImpl implements ProductService{

    1 usage  new *
    @Override
    public String getProductByID(Long id) { return "product fetched from service, id: " + id; }

    no usages  new *
    @Override
    public List<String> getAllProducts() { return null; }
```



```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    1 usage  new *
    @Override
    public String getProductByID(Long id) { return null; }

    no usages  new *
```

When to use Qualifier:

Below is wrong way of declaration…. just for demonstration



```
new *
@Service("FakeProductService")
@Qualifier("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    1 usage  new *
    @Override
    public String getProductByID(Long id) { return null; }
```

In the provided code snippet, you are using both `@Service("FakeProductService")` and `@Qualifier("FakeProductService")` annotations. However, it's important to note that these annotations serve different purposes.

1. `@Service("FakeProductService")`: This annotation is typically used to indicate that the class is a service component in Spring. The value inside the annotation can be

used as a name or identifier for the bean when it is registered in the Spring application context. It is not a qualifier annotation but is used for component scanning and bean registration.

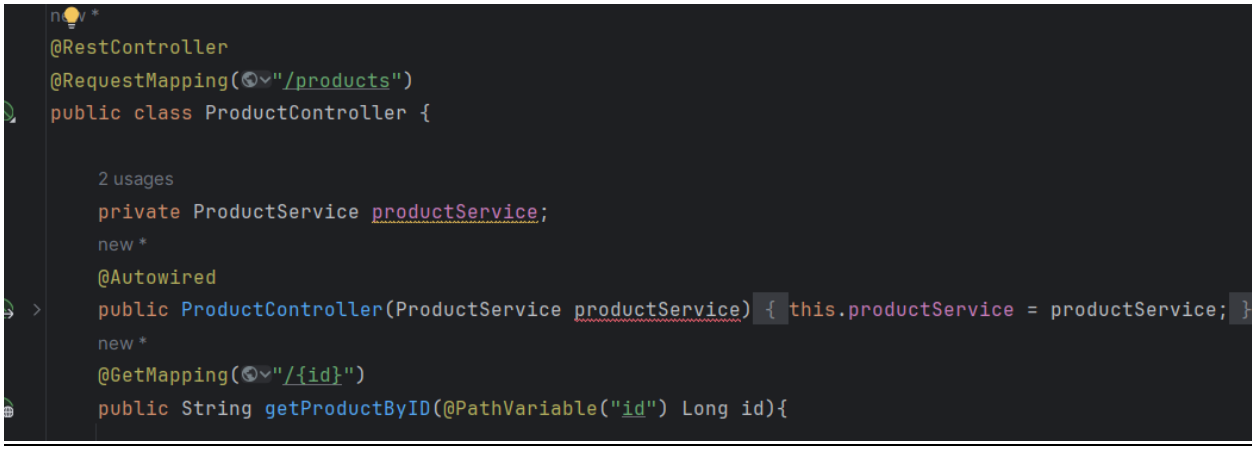
2. `@Qualifier("FakeProductService")`: This annotation is used to explicitly specify which bean should be injected when there are multiple beans of the same type. It helps resolve ambiguity when there are multiple candidates for autowiring.

If your intention is to use a qualifier to distinguish between multiple beans of the same type, you should use `@Qualifier("FakeProductService")` where the injection is happening, rather than on the service class itself.

if you are using `@Qualifier`, it should be used at the point of injection to specify which bean should be injected. The `@Service` annotation, on the other hand, is more about declaring the class as a service and providing an optional name for the bean in the Spring context.

We will add qualifier in ProductController where we inject.

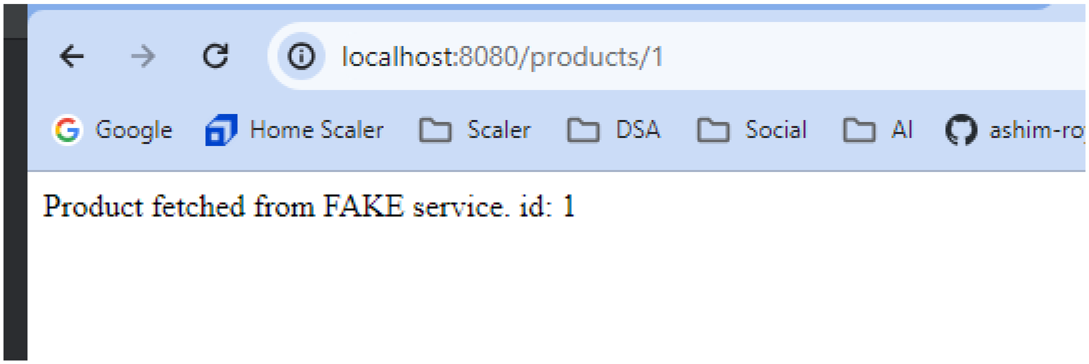
```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    @Override
    public String getProductByID(Long id) {

        return "Product fetched from FAKE service. id: " + id;
    }
```



```
new *
@RestController
@RequestMapping("/products")
public class ProductController {

    2 usages
    private ProductService productService;
    new *
    @Autowired
    public ProductController(ProductService productService) { this.productService = productService; }
    new *
    @GetMapping("/{id}")
    public String getProductByID(@PathVariable("id") Long id){
```

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

    private ProductService productService;
    @Autowired
    public ProductController(@Qualifier("FakeProductService") ProductService
productService){

        this.productService = productService;
    }
```

Qualifier will take name of service



Product fetched from FAKE service. id: 1

So qualifier is can identify which bean need to be used..

New concept:

So to keep our data will keep at DB. If someone maintains the DB for us. And we can use that for our advantage. There is a fakestore API.

https://fakestoreapi.com/

provide us a product service which we can use in our system. We use fakestore PAI to act as source of truth. Same way we use weather.com data.. when customer will hit our api they wil not notice. They see all data in our side.

How to integrate any 3rd party API..

Go through this: https://fakestoreapi.com/docs

they have her 20-25 products.

Why we should avopid field injection: https://stackoverflow.com/questions/39890849/what-exactly-is-field-injection-and-how-to-avoid-it

Injection types

There are three options for how dependencies can be injected into a bean:

1. Through a constructor
2. Through setters or other methods
3. Through reflection, directly into fields

You are using option 3. That is what is happening when you use @Autowired directly on your field

Injection guidelines

Edit: *These 3 links mentioned here are for Spring 4.2., for newer version documentation as per 2023 (6.09) see list below*

A general guideline, [which is recommended by Spring](#) (see the sections on [Constructor-based DI](#) or [Setter-based DI](#)) is the following:

- For mandatory dependencies or when aiming for immutability, use constructor injection
- For optional or changeable dependencies, use setter injection
- Avoid field injection in most cases

Field injection drawbacks

The reasons why field injection is frowned upon are as follows:

- You cannot create immutable objects, as you can with constructor injection
- Your classes have tight coupling with your DI container and cannot be used outside of it
- Your classes cannot be instantiated (for example in unit tests) without reflection. You need the DI container to instantiate them, which makes your tests more like integration tests
- Your real dependencies are hidden from the outside and are not reflected in your interface (either constructors or methods)
- It is really easy to have like ten dependencies. If you were using constructor injection, you would have a constructor with ten arguments, which would signal that something is fishy. But you can add injected fields using field injection indefinitely. Having too many dependencies is a red flag that the class usually does more than one thing, and that it may violate the Single Responsibility Principle.

Conclusion

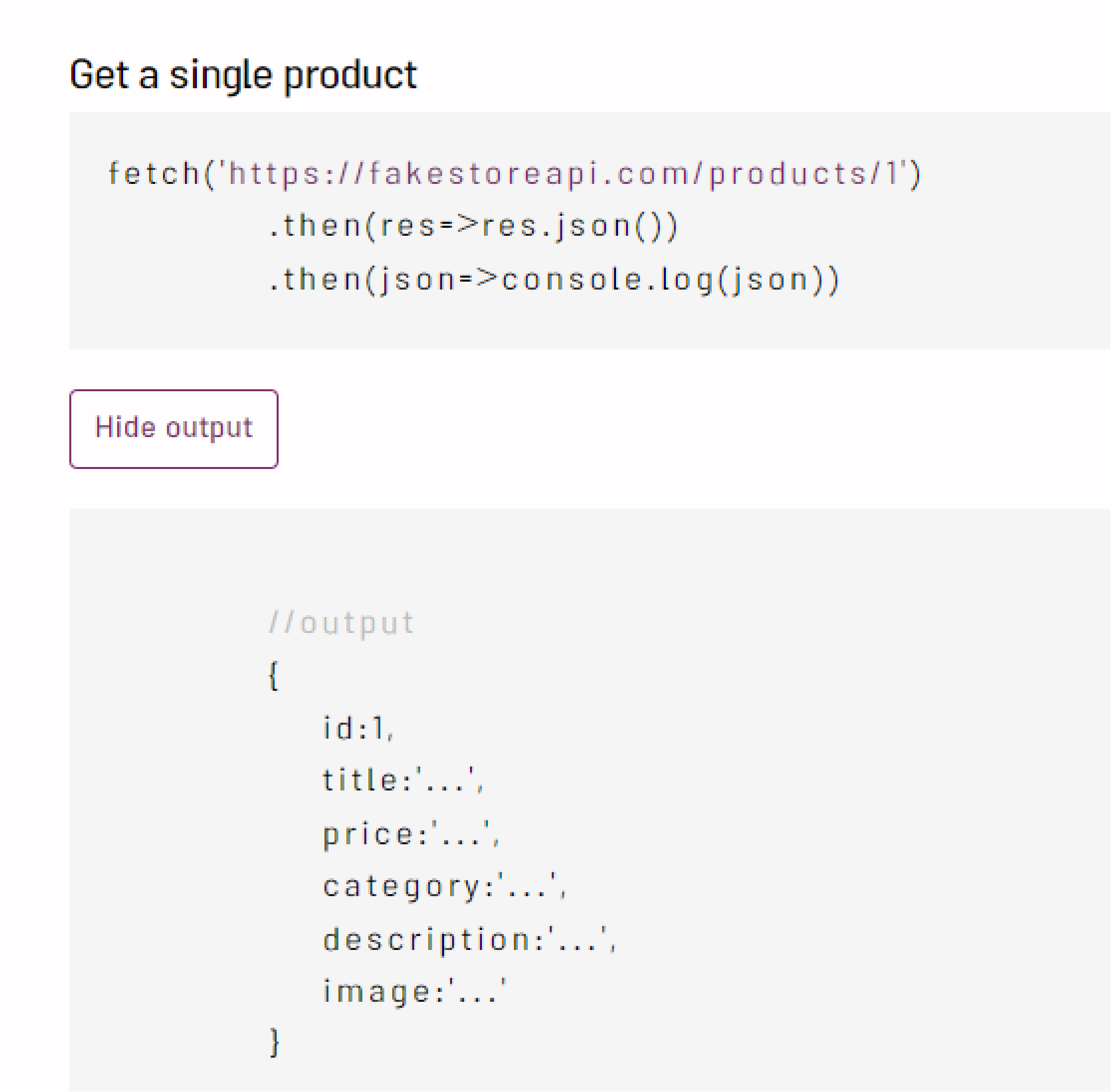
Depending on your needs, you should primarily use constructor injection or some mix of constructor and setter injection. Field injection has many drawbacks and should be avoided. The only advantage of field injection is that it is more convenient to write, which does not outweigh all the cons.

Further reading

I wrote a blog article about why field injection is usually not recommended: [Field Dependency Injection Considered Harmful](#).

To connect with 3rd party service we need a URL to connect..

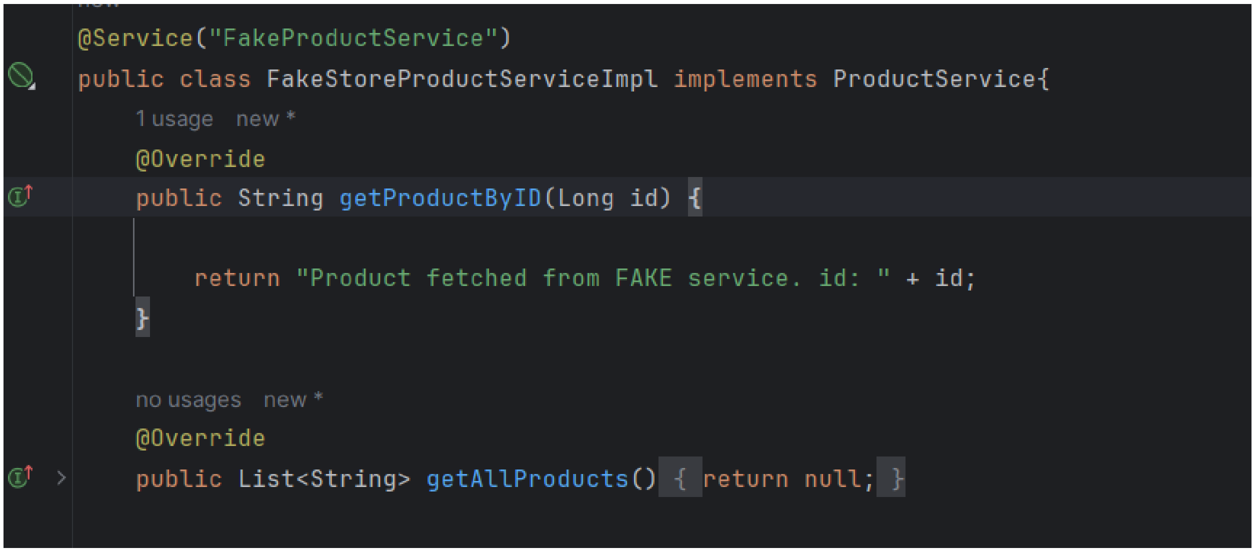
to get a single product:



Get a single product

```
fetch('https://fakestoreapi.com/products/1')
            .then(res=>res.json())
            .then(json=>console.log(json))
```

```
            //output
            {
                id:1,
                title:'...',
                price:'...',
                category:'...',
                description:'...',
                image:'...'
            }
```



```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    1 usage   new *
    @Override
    public String getProductByID(Long id) {

        return "Product fetched from FAKE service. id: " + id;
    }

    no usages   new *
    @Override
    public List<String> getAllProducts() { return null; }
```

Add the url:

```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    private String getProductURL = "https://fakestoreapi.com/products/1";
    @Override
    public String getProductByID(Long id) {
```

We need something to make a call. We need rest template builder…

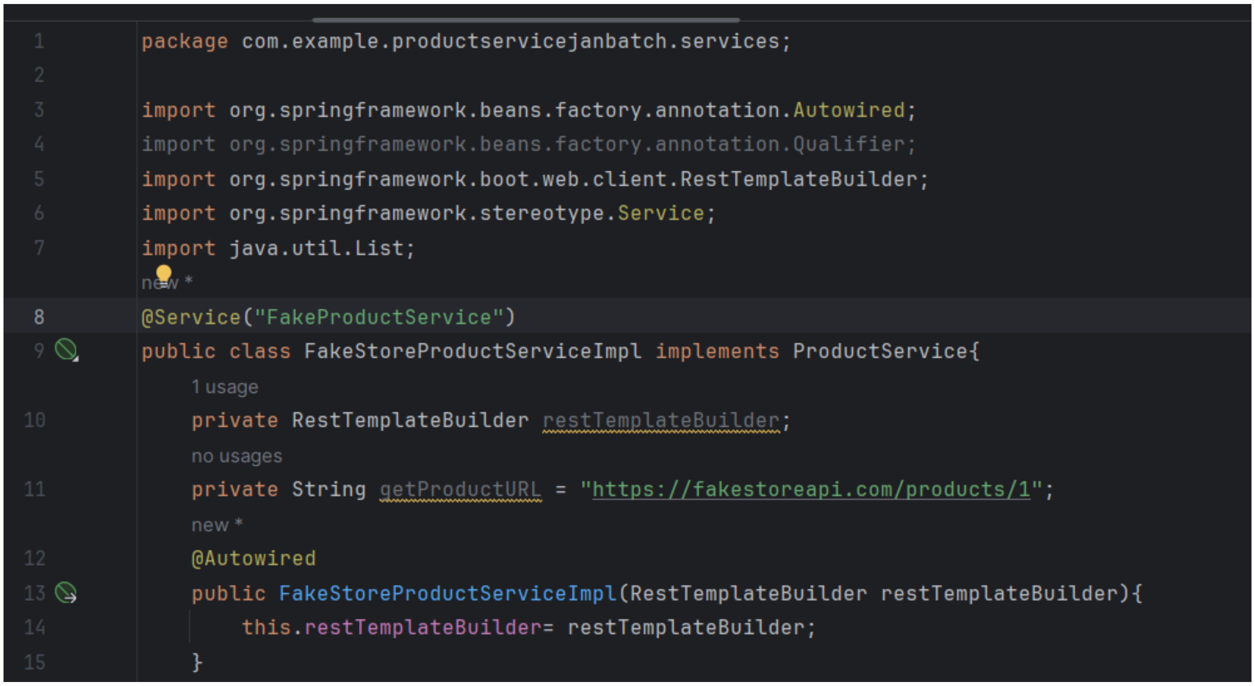
https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html#acceptHeaderRequestCallback(java.lang.Class)

```
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.stereotype.Service;

import java.util.List;

@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    private RestTemplateBuilder restTemplateBuilder;
```

I cant access the class… RestTemplateBuilder

So RestTemplateBuilder object will br created by SPRINg. So we will autowired.

```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    private RestTemplateBuilder restTemplateBuilder;
    private String getProductURL = "https://fakestoreapi.com/products/1";
    @Autowired
    public FakeStoreProductServiceImpl(RestTemplateBuilder
restTemplateBuilder){
        this.restTemplateBuilder= restTemplateBuilder;
    }
```



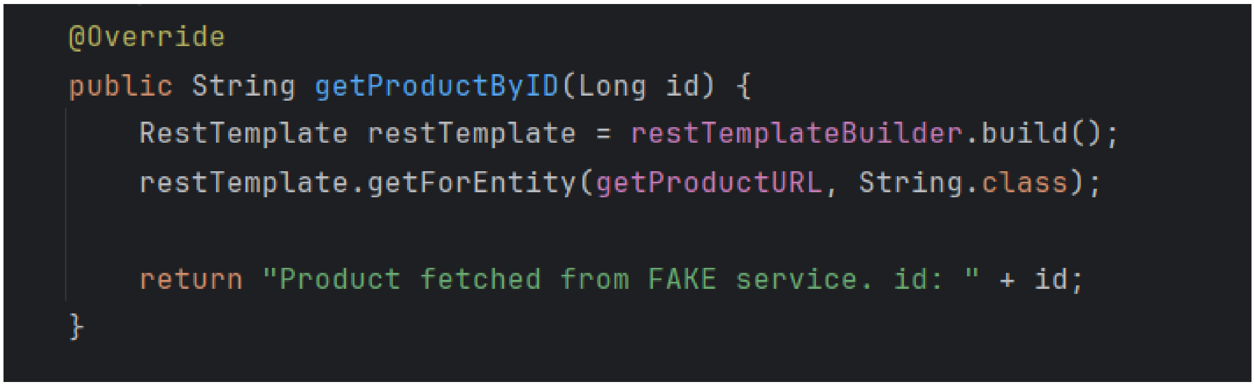
```
package com.example.productservicejanbatch.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.stereotype.Service;
import java.util.List;
new *
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    1 usage
    private RestTemplateBuilder restTemplateBuilder;
    no usages
    private String getProductURL = "https://fakestoreapi.com/products/1";
    new *
    @Autowired
    public FakeStoreProductServiceImpl(RestTemplateBuilder restTemplateBuilder){
        this.restTemplateBuilder= restTemplateBuilder;
    }
```

Now I need to get an entity.. for that I need to use resttemplate..

I will make a get call for an entity.

What does resttemplateBuilder provides?? When you are connecting to any URL. You try to make an HTTP call. You try to make a call to http endpoint with some details like get, post, put etc. if you create a http client and every time you need to specify that its better that we have a template already and let everything will be taken care by Spring.

```
@Override
public String getProductByID(Long id) {
    RestTemplate restTemplate = restTemplateBuilder.build();
    restTemplate.getForEntity(getProductURL, String.class);

    return "Product fetched from FAKE service. id: " + id;
}
```



```
@Override
public String getProductByID(Long id) {
    RestTemplate restTemplate = restTemplateBuilder.build();
    restTemplate.getForEntity(getProductURL, String.class);

    return "Product fetched from FAKE service. id: " + id;
}
```

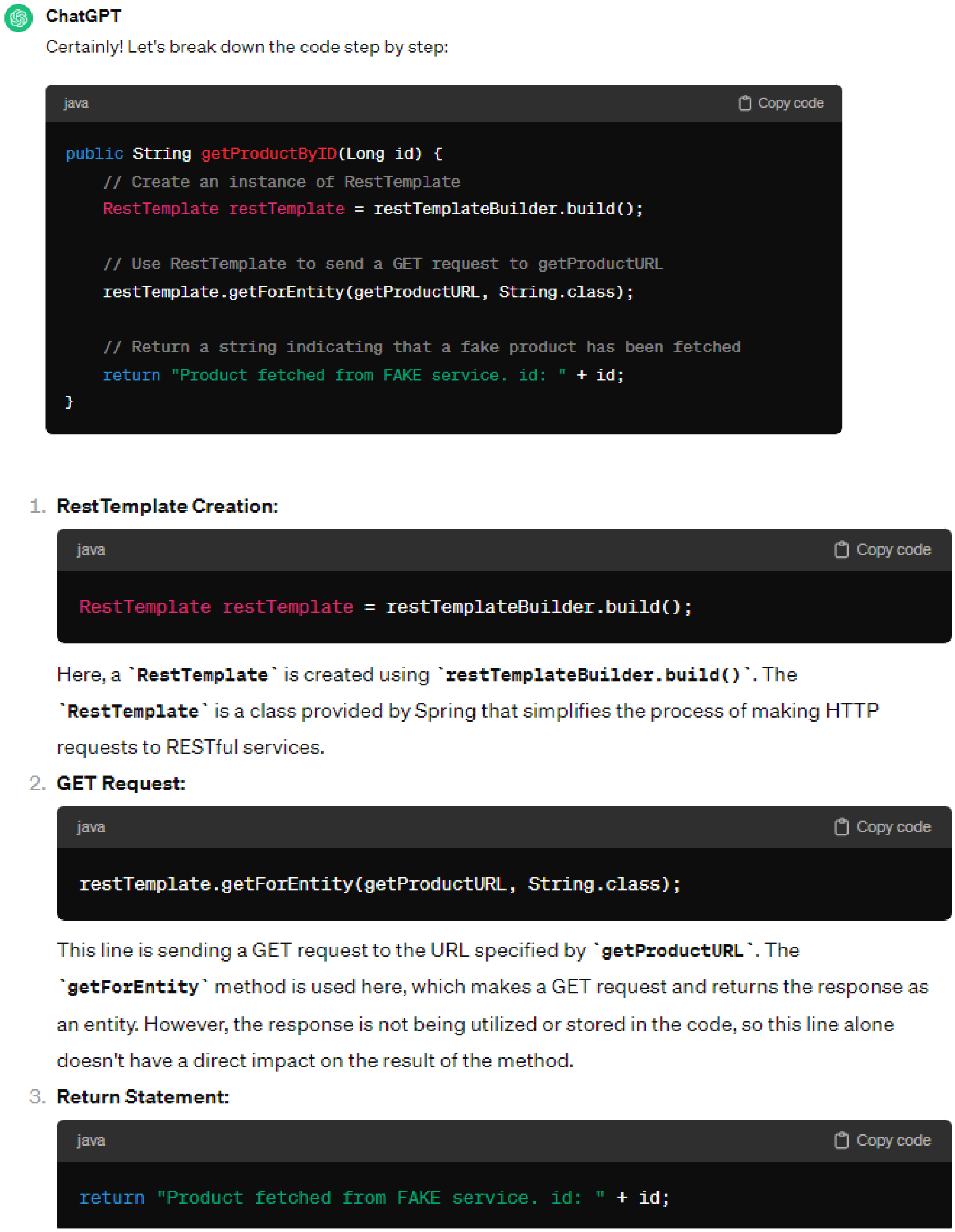
There are some fill in the blanks which I will fill and rest will be taken care by spring..

resttemplateBuilder will provide me a template framework, if I wanna do a get for an entity . I will just provide the URL, provide for which you wanna convert this.. [spring.class]

I wanna make a http call to this URL(getProduct). Get the product and convert into spring.class..

Whatever is requ to make an http call to this URL for the get mapping everything is taken care by spring.

Why I have build the resttemplateBuilder.build… it should be a same http call or new? Generally a new http call is used. Sometime in critical environment you wanna reuse the http. Then yu do. Otherwise 99% time you make a new thing.



ChatGPT

Certainly! Let's break down the code step by step:

```java
public String getProductByID(Long id) {
    // Create an instance of RestTemplate
    RestTemplate restTemplate = restTemplateBuilder.build();

    // Use RestTemplate to send a GET request to getProductURL
    restTemplate.getForEntity(getProductURL, String.class);

    // Return a string indicating that a fake product has been fetched
    return "Product fetched from FAKE service. id: " + id;
}
```

1. **RestTemplate Creation:**

   ```java
   RestTemplate restTemplate = restTemplateBuilder.build();
   ```

   Here, a `RestTemplate` is created using `restTemplateBuilder.build()`. The `RestTemplate` is a class provided by Spring that simplifies the process of making HTTP requests to RESTful services.
2. **GET Request:**

   ```java
   restTemplate.getForEntity(getProductURL, String.class);
   ```

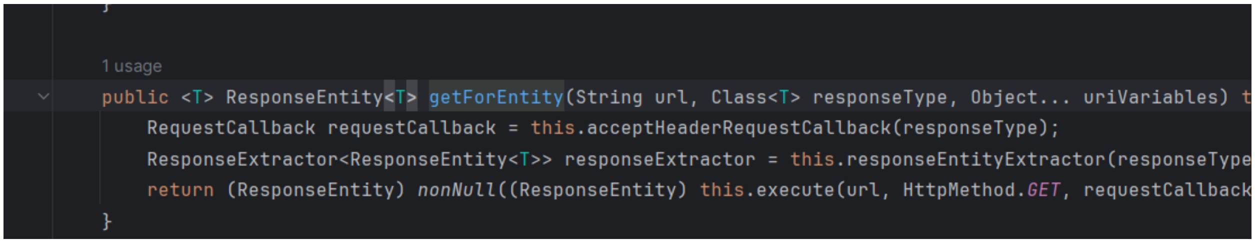
   This line is sending a GET request to the URL specified by `getProductURL`. The `getForEntity` method is used here, which makes a GET request and returns the response as an entity. However, the response is not being utilized or stored in the code, so this line alone doesn't have a direct impact on the result of the method.
3. **Return Statement:**

   ```java
   return "Product fetched from FAKE service. id: " + id;
   ```

1. Regardless of the result of the GET request, this line always returns a string indicating that a fake product has been fetched. The value of `id` is concatenated to the string, suggesting that the product with the specified ID has been fetched (even though the actual product data is not retrieved from the GET request).

In summary, the provided code seems to be a placeholder or a mock implementation of a method in a `ProductService` class. It creates a `RestTemplate`, sends a GET request (though not using the response), and then returns a hardcoded string indicating that a product has been fetched from a fake service. If you want to actually retrieve data from the service and use it, you need to handle the response returned by `getForEntity`.

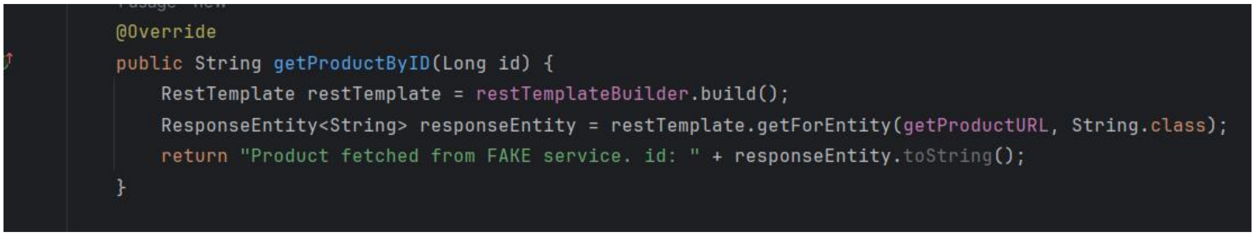
Here getforentity return an string



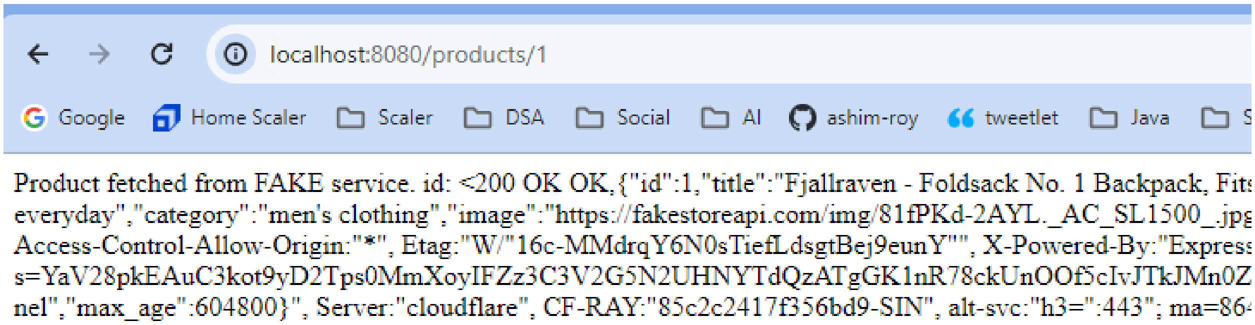
```
    }

    1 usage
    public <T> ResponseEntity<T> getForEntity(String url, Class<T> responseType, Object... uriVariables) t
        RequestCallback requestCallback = this.acceptHeaderRequestCallback(responseType);
        ResponseExtractor<ResponseEntity<T>> responseExtractor = this.responseEntityExtractor(responseType
        return (ResponseEntity) nonNull((ResponseEntity) this.execute(url, HttpMethod.GET, requestCallback
    }
```

Using generics here...

```
@Service("FakeProductService")
public class FakeStoreProductServiceImpl implements ProductService{
    private RestTemplateBuilder restTemplateBuilder;
    private String getProductURL = "https://fakestoreapi.com/products/1";
    @Autowired
    public FakeStoreProductServiceImpl(RestTemplateBuilder
restTemplateBuilder){
        this.restTemplateBuilder= restTemplateBuilder;
    }
    @Override
    public String getProductByID(Long id)  {
        RestTemplate restTemplate = restTemplateBuilder.build();
        ResponseEntity<String> responseEntity =
restTemplate.getForEntity(getProductURL, String.class);
        return "Product fetched from FAKE service. id: " +
responseEntity.toString();
    }
```

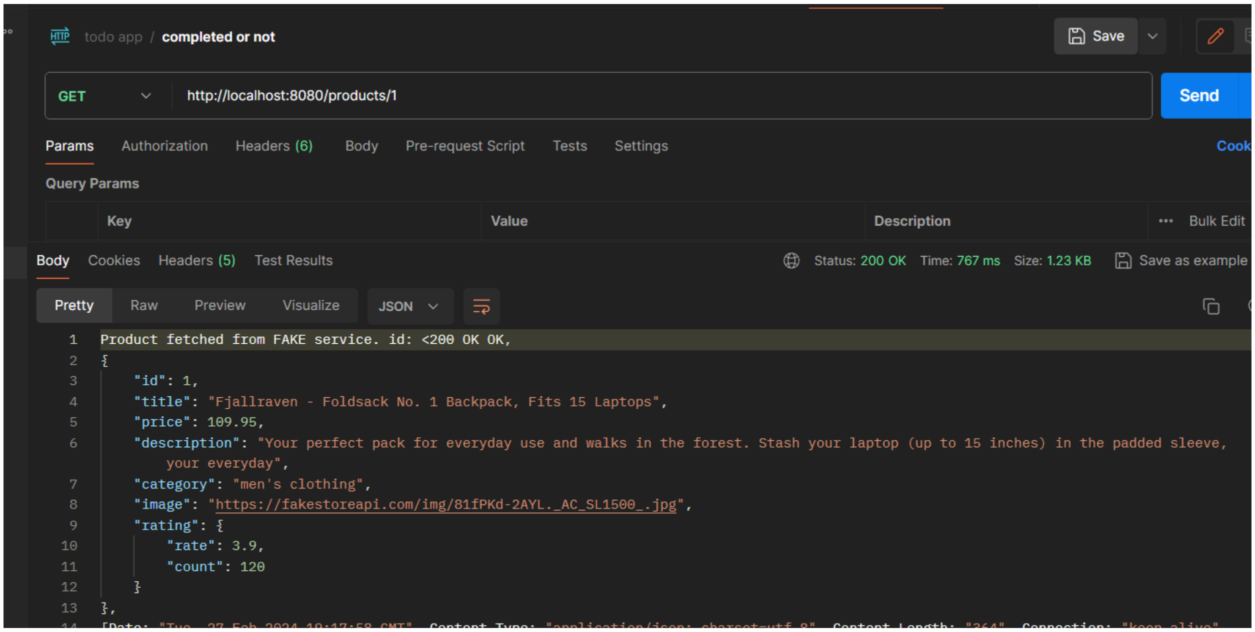


```
    @Override
    public String getProductByID(Long id) {
        RestTemplate restTemplate = restTemplateBuilder.build();
        ResponseEntity<String> responseEntity = restTemplate.getForEntity(getProductURL, String.class);
        return "Product fetched from FAKE service. id: " + responseEntity.toString();
    }
```



```
Product fetched from FAKE service. id: <200 OK OK,{"id":1,"title":"Fjallraven - Foldsack No. 1 Backpack, Fits
everyday","category":"men's clothing","image":"https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg
Access-Control-Allow-Origin:"*", Etag:"W/"16c-MMdrqY6N0sTiefLdsgtBej9eunY"", X-Powered-By:"Express
s=YaV28pkEAuC3kot9yD2Tps0MmXoyIFZz3C3V2G5N2UHNYTdQzATgGK1nR78ckUnOOf5cIvJTkJMn0Z
nel","max_age":604800}", Server:"cloudflare", CF-RAY:"85c2c2417f356bd9-SIN", alt-svc:"h3=":443"; ma=86
```

Lets look at postman: in JSON



```
Product fetched from FAKE service. id: <200 OK OK,
{
    "id": 1,
    "title": "Fjallraven - Foldsack No. 1 Backpack, Fits 15 Laptops",
    "price": 109.95,
    "description": "Your perfect pack for everyday use and walks in the forest. Stash your laptop (up to 15 inches) in the padded sleeve,
        your everyday",
    "category": "men's clothing",
    "image": "https://fakestoreapi.com/img/81fPKd-2AYL._AC_SL1500_.jpg",
    "rating": {
        "rate": 3.9,
        "count": 120
    }
},
```

The scope of the getProductByID, when I need to make a call I need to make a new http call. That's why it's a LOCAL>. If it would have been a global variable I should not create a new obj everytime…. I should not interfare.

As this RestTemplate restTemplate will be used across the functions.

We are using buolder pattern…

How have we used rest template? What its doing. Its using a rest template bulder. restTemplateBuilder provides a basic structure of a http call.

Go through the doc: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html#acceptHeaderRequestCallback(java.lang.Class)

`public class` **`RestTemplate`** `extends InterceptingHttpAccessor implements RestOperations`

Synchronous client to perform HTTP requests, exposing a simple, template method API over underlying HTTP client libraries such as the JDK `HttpURLConnection`, Apache HttpComponents, and others. RestTemplate offers templates for common scenarios by HTTP method, in addition to the generalized `exchange` and `execute` methods that support less frequent cases.

RestTemplate is typically used as a shared component. However, its configuration does not support concurrent modification, and as such its configuration is typically prepared on startup. If necessary, you can create multiple, differently configured RestTemplate instances on startup. Such instances may use the same underlying `ClientHttpRequestFactory` if they need to share HTTP client resources.

Whatever u do via an http method via a get, post, put post… first create a http client. Then u need to specify the type of call and then specify the URL .. all of this is taken crae by rest template.. everything you wanna specify in http cal perspective.

If not specify your parameter.

To build the obj of rest template we use rest template.build.. just like builder pattern..