CONTENTS

* PROJECT SETUP , 2
* ANNOTATIONS ,
* PROJECT IMP CODE DISCUSSION ,
* Stripe End-To-End Technical Integration Approach ,

PROJECT SETUP

* LOMBOK:

Add Lombok Maven dependency without version in POM.xml file

Add Lombok link into your IDE.

Steps-------

Apply settings at IDE level.

Help => Install new software

Work with: <https://projectlombok.org/p2>, click add,give name Lombok

Add all Trust it.

Apply these Then close & Restart.

* DevTool Dependency:

No need to re-start server every time after a change.

It’ll automatically save the changes and restore server.

* Install Sonar and SonarLint in IDE. It’s not mandatory
* GSON Library

We have to add GSON Dependency.

It’ll convert **java object to json** string.

**Interface and Implement class of Interface**

when one layer talks to another layer, always do it as interfaces & implement class.

Controller <=> service

Only this below two will be linked through interface,

Interface

Impl

Service <=> Dao

Only this below two will be linked through interface,

Interface

Impl

How to do that??

We have to create object only Interface. Below are the example to elaborate,

interface MyInterface {

m1(){}

m2(){}

}

class MyInterfaceImpl implements MyInterface {

@Override

m1(){}

m2(){}

m3()

m4()

m5()

}

MyInterface instance = new MyInterfaceImpl();

//we can call only this two

//this is **correct approach**

instance.m1

instance.m2

**//**if we create obj like this we can cl all method, wrong approach

MyInterfaceImpl instanceImpl = new MyInterfaceImpl();

instanceImpl.m1

instanceImpl.m2

instanceImpl.m3

instanceImpl.m4

**ANNOTATION OF SPRING**

**@Component**

Whenever you annotate any class with @Component, you let spring manage the lifecycle of that class. That class will become Spring Bean class.

Example:

@Component

Class MyServiceImpl {} //service class has interface, those interface’s implement class this one

**@RestController** (Internally its @Component only)

It’ll help to connect, the Dependent object with the controller class.

But it’ll **not inject** the dependent object to controller class object.

Like this,

@RestController

**public** **class** TestController {

**private** PaymentInterface obj; // This is **Field Injection**.

String val= obj.paymentProcess(payment);

System.***out***.println(val); //it’ll give error,cause PaymentInterface object is linked but not injected

}

For that we need @Autowired annotation

**@Autowired**

**@Configuration** (Internally its @Component only)

Spring Configuration annotation indicates that the class has [@Bean](https://www.digitalocean.com/community/users/bean) definition methods. So Spring container can process the class and generate Spring Beans to be used in the application.

**@Bean**

**@Primary**

While creating one object of a component class in AppConfiguration.

If we created two different method, to create object of a same class.

Then springboot run will fail,it’ll ask youto which one to choose. To choose only one,

We have to write @primary annotation on that method. So spring will choose that,not the other method. Example,

@Configuration

**public** **class** AppConfig {

@Bean

**public** Gson makeGson() {

**return** **new** Gson();

}

@Bean

@Primary

**public** Gson makeGsonOfGsonClass() {

**return** **new** Gson();

}

}

**@Qualifier**

It’s same like @primary. we can **eliminate the issue of which bean needs to be injected**.

Example,  
@Autowired

@Qualifier("makeGson ")

**private** Gson gson;

The main Difference between @Primary vs @Qualifier,  
 @Primary annotation is useful when we want to **specify which bean of a certain type should be injected by default or pick By Spring Application by deafult**, while start running the Springboot Application.

In a scenario,

If we require **the other bean at some injection point**, we would need to specifically indicate it via the @Qualifier annotation.

It’s worth noting that **if both the @Qualifier and @Primary annotations are present, then the @Qualifier annotation will have precedence.** Basically, @Primary defines a default, while @Qualifier is very specific.

**@PatchMapping**

**Field Injection/Constructor Injection**

In Field, @Autowired annotation required.

But in constructor it’s not required. It’ll done by Springboot or IDE

**Dependency LookUp:**

Dependency lookup is where Target class itself looks up for dependent class object and chooses appropriate implementation. It’ll not happen by Spring Boot automatically.

In Spring, dependency lookup is often used in conjunction with the ApplicationContext, which provides methods like getBean() to retrieve beans from the container.

# [Difference between @Bean and @Autowired](https://stackoverflow.com/questions/34172888/difference-between-bean-and-autowired)

* @Bean tells Spring 'here is an instance of this class, please keep hold of it and give it back to me when I ask'.
* @Autowired says 'please give me an instance of this class, for example, one that I created with an @Bean annotation earlier'.

# [Difference between @Bean and @Component](https://stackoverflow.com/questions/34172888/difference-between-bean-and-autowired) \*\*\*\*\*

# @Component (and @Service and @Repository) are used to auto-detect and auto-configure beans using classpath scanning.

Sometimes automatic configuration is not an option. **When?** Let's imagine that you want to wire components from 3rd-party libraries (**you don't have the source code so you can't annotate its classes with @Component**), so automatic configuration is not possible.

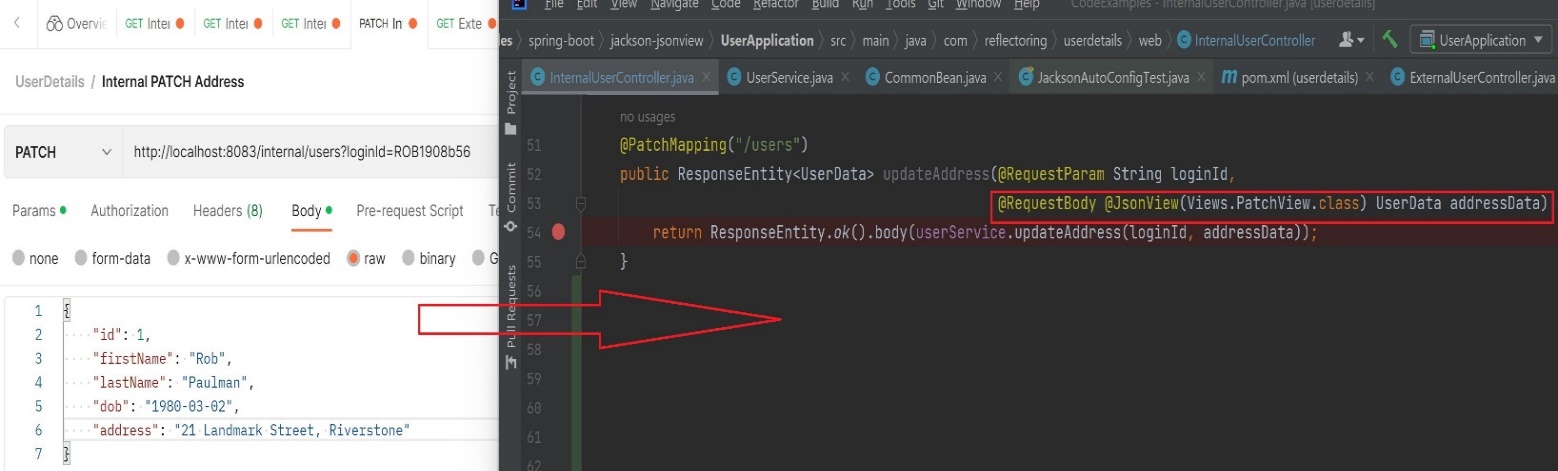
1. @Component **auto detects** and configures the beans using classpath scanning whereas @Bean **explicitly declares** a single bean, rather than letting Spring do it automatically.
2. @Component **does not decouple** the declaration of the bean from the class definition where as @Bean **decouples** the declaration of the bean from the class definition.
3. @Component is a **class level annotation** whereas @Bean is a **method level annotation** and name of the method serves as the bean name.
4. @Component **need not to be used with the @Configuration** annotation where as @Bean annotation has to be **used within the class which is annotated with @Configuration**.
5. We **cannot create a bean** of a class using @Component, if the class is outside spring container whereas we **can create a bean** of a class using @Bean even if the class is present **outside the spring container**.
6. @Component has **different specializations** like @Controller, @Repository and @Service whereas @Bean has **no specializations**.

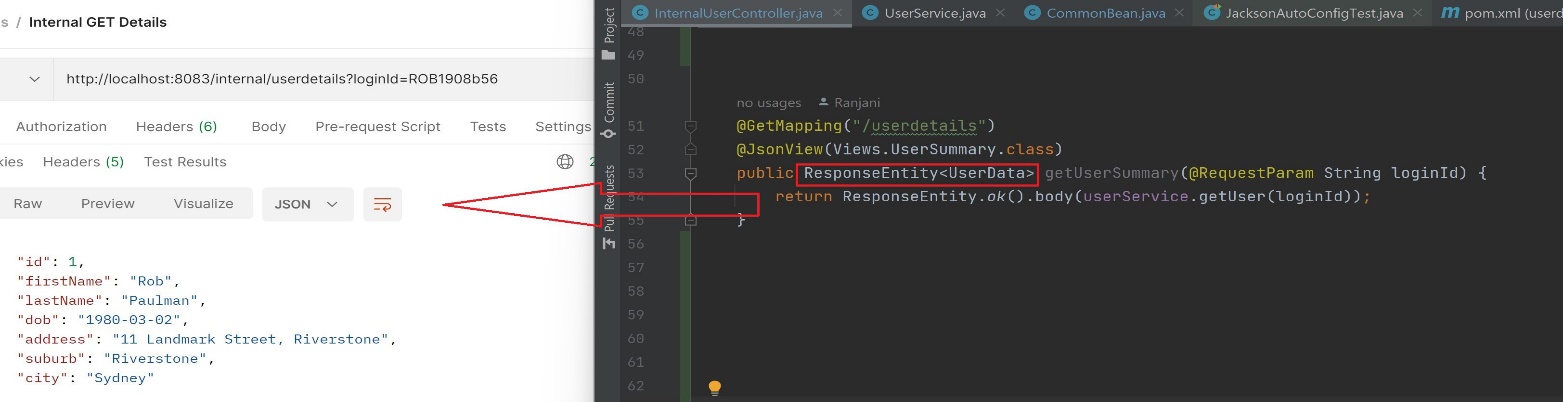
## Serialize and Deserialize objects in a Spring Boot Application

Serialization and deserialization form the core of REST APIs. **Spring Boot internally uses Jackson’s ObjectMapper class to perform serialization and deserialization.**

The Jackson API is one of the best JSON parsers in Java. SpringBoot **@RequestBody** annotation implicitly integrates well with Jackson and convert **JSON Data into Java Object**.

Deserialization example:



Serialization example: 

**============================================**

PROJECT IMP CODE DISCUSSION

1. Core Payment Status System.

- Designing & Coding

2. Working with DB

3. Error Handling system

4. Standards/Debugging

**Core Payment Status System:**

Multiple statuses to represent every situation. Like,

CREATED

INITIATED

PENDING

SUCCESS

FAILED

Implementing in Code Diagram,

1. HookPoint

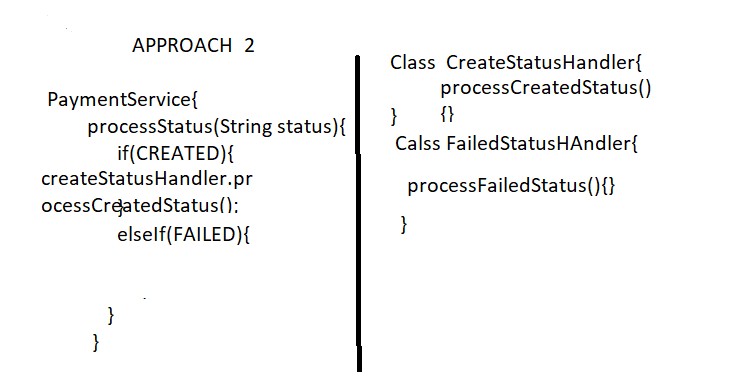
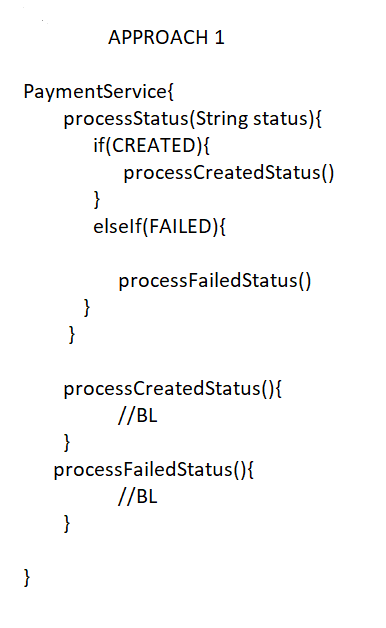
Means Something has changed, on that change we have to write BL. Like status has changed, based on status we have to write code.

1. In entire system, multiple hook points are there, where status needs to be changed.

2. Every hook point, needs to change status to some different value.

3. For every status change, more logic can come as system grows. Like send mail,send notification,DB manipulation.

Central System, that will manage all the Status HookPoint and their BL.



APPROACH 1 ->

Under class there will be one method, on that method we will pass the status,based on status it’ll call there Belonging Method from the same class.

Cons: More logic in one class. We have to write code a in a small logic break part. That we can reuse after without modifying the original or already developed code. As much as possible we have to break our code in different section.

APPROACH 2 ->

Different method we will write on different class. And we will call them in our if-else block using there object creation. Like,

CreateStatusHandler obj= new CreateStatusHandler();

Obj.processCreatedStatus();

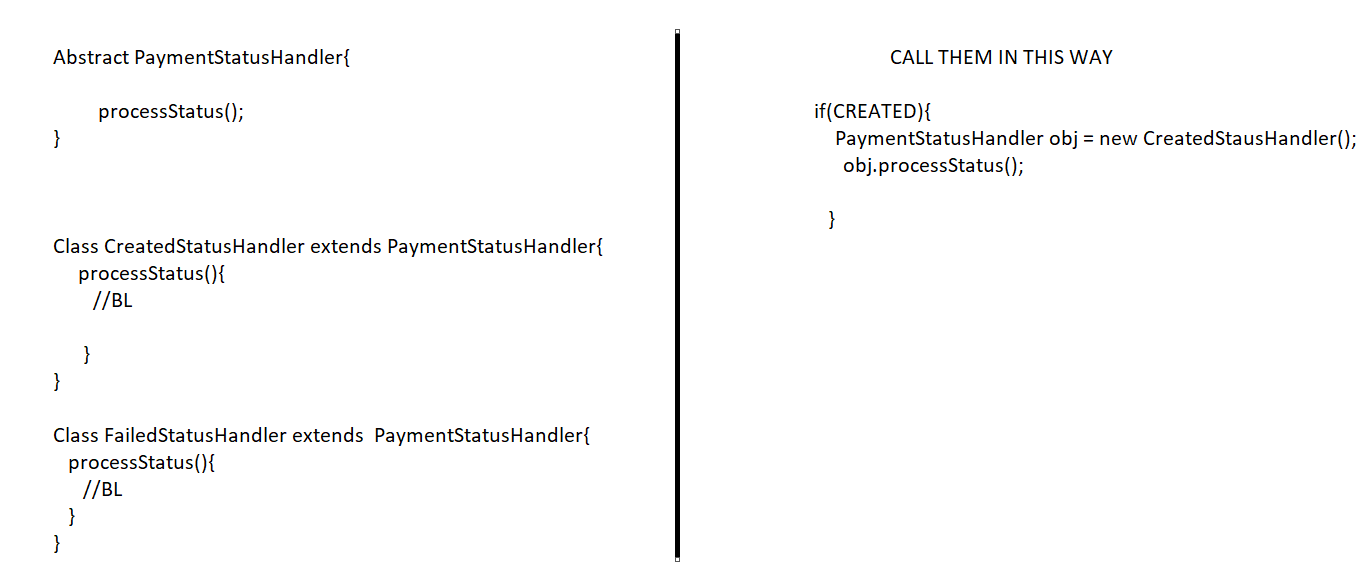
Cons: Both of the classes are on same pattern doing some operation on basic of status. **If more than one class logic and pattern are same, then in OOPs concept we have write them on Interface or Abstaraction Level**. It’s a rule. If they are not related then no need to bother.

APPROACH 3 ->

We will Use Interface or Abstract Class here. There a common method will be present. That method will be shared or overridden by every StatusHandler class. Here we will use Abstract,cause no return type required,and one common Method Sharing By different class.

**Why To Choose Abstract Not interface Here?**

* Use **abstract classes** when you need to share common state and behavior among related classes and enforce a single inheritance hierarchy.
* Use **interfaces** when you need to define capabilities or behaviors that can be implemented by multiple classes, allowing for multiple inheritance of type and greater flexibility.



2.Factory Design Pattern

A Method, which takes some parameters as input.Based on parameters decides which object to create (child class object). And return type of factory method, is same parent type.

We will implement this on HookPoint Logic. Like, for every condition we are creating obj and calling that class same name method. It’s repeatative Approach. So we will write a code that will check pattern of every condition. Use this factory in Service Layer Code.

public PaymentStatusHandler getStatusHandler(String status) {

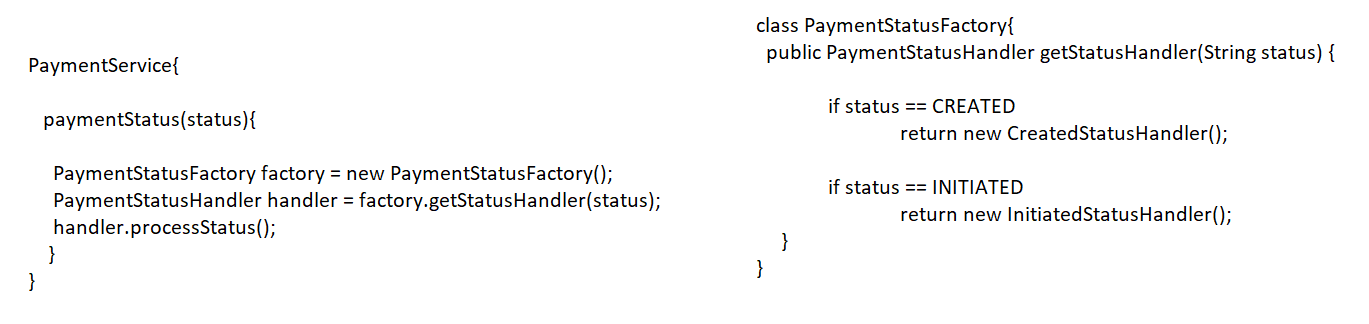
if status == CREATED

return new CreatedStatusHandler

if status == INITIATED

return new CreatedStatusHandler

}



**===================================================**

**IMP Required SpringBoot and JAVA Topic**

ModelMapper:

We use it in Springboot or Java class when complex **object conversions** are necessary.

Like, User class needs to convert in Employe class. And Users all Fields Data will be store on Employee Fields Data. How to do that normally in JAVA Class?

Employee emp= new Employee();

emp.setName(user.getName()); emp.setSalary(user.getSalary());

Using ModelMapper,

@Bean

public ModelMapper modelMapper() {

return new ModelMapper();

}

@Autowired

private ModelMapper modelMapper;

public Employe convertToEmploye(User user) {

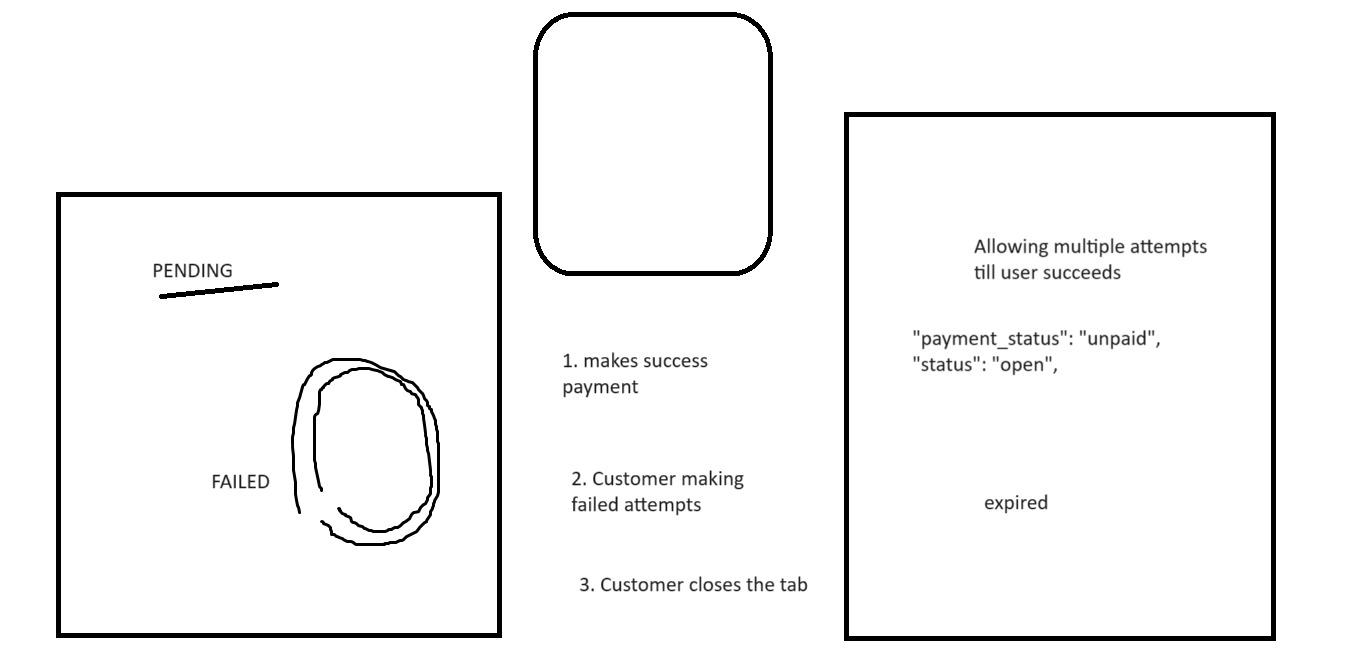
return **modelMapper.map(user, Employe.class); //source obj, DestinationClass.class**

}

DAO acts as a bridge between the database and the application. DTO acts as a data store from where the data is received and transferred to different layers i.e., to the DAO application

**============================================**

**Live Diagram:**

****

**Stripe End-To-End Technical Integration Approach**

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

Payment Status handling and Checking there log what is coming after API call.

Payment lifecycle:

CREATED - payment is valid and we have saved in DB

INITIATED - Before we call stripe/psp

PENDING - Success response with url. Means customer is on Payment page, and we are waiting for completion of payment.

SUCCESS - Payment has successfully happened

FAILED - Unable to complete payment

Every payment should end is final statuses (SUCCESS/FAILED)

If one payment reached final status, then status cannot change.

IN DB PENDING

1. *Successful Payment*

On successful completion of payment, checkout.session.completed event

other events

Two things should happen:

browser should redirect to success\_url (in createSession)

We should get notification "checkout.session.completed"

Test Expectations

Actually testing, in test mode

Actual Actions, with Success card details

Notifications coming as in Stripe CLI,

2024-07-25 10:16:53 --> checkout.session.completed [evt\_1PgJZoDKmZbp6ODmK5Q9KGJF]

2024-07-25 10:16:43 --> charge.succeeded [evt\_3PgJZdDKmZbp6ODm0qvuDy6X]

2024-07-25 10:16:43 --> payment\_intent.succeeded [evt\_3PgJZdDKmZbp6ODm0N5UkDQk]

2024-07-25 10:16:43 --> payment\_intent.created [evt\_3PgJZdDKmZbp6ODm0XxjfIRL]

2. *Failure Payment*

when failure happening.. did not get checkout.session.completed

however other events are still coming.

**customer is still sitting on the hosted page**.-> It’s a drawback,we have to implement logic here

- Customer is still on browser. With error message

2024-07-25 10:19:14 --> charge.failed [evt\_3PgJc4DKmZbp6ODm19yAeG1Q]

2024-07-25 10:19:14 --> payment\_intent.created [evt\_3PgJc4DKmZbp6ODm156EMAjJ]

2024-07-25 10:19:14 --> payment\_intent.payment\_failed [evt\_3PgJc4DKmZbp6ODm13FmHMfU]

Failed attempt2:

2024-07-25 10:20:28 --> charge.failed [evt\_3PgJc4DKmZbp6ODm1Z7rzt9e]

2024-07-25 10:20:28 --> payment\_intent.payment\_failed [evt\_3PgJc4DKmZbp6ODm1zwZKsPD]

1. If payment failed or customer decided to not to payment then,There have not straight way to coming back to ecomm/client site page back.

either browser back or close tab.

So in stripe psp there have two things that you can use for back,if customer don’t want to do payment.  
**return\_url- It’s for embedded page not for** **hosted**

**cancel\_url – it’s for hosted. We will pass like this,**

**cancel\_url :** https://example.com/cancel

1. *How do you finalized the field to be passed to PSP*?

---------

1. REQUIRED / MANDATORY Fileds have to pass

2. While testing.. & logically connecting the system...

To functionally complete the system, identify whichever fields are needed.. and apply...

3. Anything else, if you want to pass then pass... product owner

If some data is required, if you dont have it.. then what..

We have to build this system as feature...

HARDCODE - FIXED value

"XYZ"

Code the feature

Or ask the incoming ecommer team to pass the data

1. *Failure Payment after Added Back or Cancel button*

cancel url added.. and trying failed card...

even after adding cancel url, customer is still on stripe page, but failure notifications are coming.

On clicking cancel\_url (back option),

customer is redirected to cancel url.

**No notification received on Stripe CLI**

status: PENDING

We have to write in PENDING in DB if the status is not SUCCESS/FAILURE,

- We cannot change status in our DB, based on notifications that we receive from Stripe system. Like,

charge.failed

payment\_intent.payment\_failed

If got failure notification still we have to write every log in Db properly,but status will be in PENDING,cause success notification we didn’t get.

At what time we can make a FAILURE Status,

* When customer press Back button/ refresh the page / close the page.
* If card details is wrong then Stripe session will be still open and Customer can entry the correct Card Details and Can made a success Payment on the same session.
* Every Time before setting the status FAILURE, we have to call **ExpireSessionAPIs**

WE WILL CHECK HERE WHAT PARAMETER WE ARE GETTING IN RESPONSE AFTER HITTING APIs::::::

* When customer is sitting on Stripe Hosted page,

didn’t gave any card details, payment has not started, Hit getStatus API,

"payment\_status": "unpaid",

"status": "open",

* When customer clicks Back button/Close Tab:

Hit getStatus,

"payment\_status": "unpaid",

"status": "open",

* Fail payment & Hit GetstatusAPI

"payment\_status": "unpaid",

"status": "open",

* When payment is successfully done

"payment\_status": "paid",

"status": "complete",

* Our **payments session** **should close**... we cannot keep it open...

PENDING should convert to

SUCCESS

FAILED

* We will blindly update the payment as FAILED in our system after 30mins.. if we dont get any success notification...
* when we close, same time, stripe should also close.. so the links becomes invalid.. and both system are considering as ended...

Expire API::

on calling expire API of An ID,we are getting in response as,

"payment\_status": "unpaid",

"status": "expired",

If we call the same ID in getSessionStatus, getting response as

"payment\_status": "unpaid",

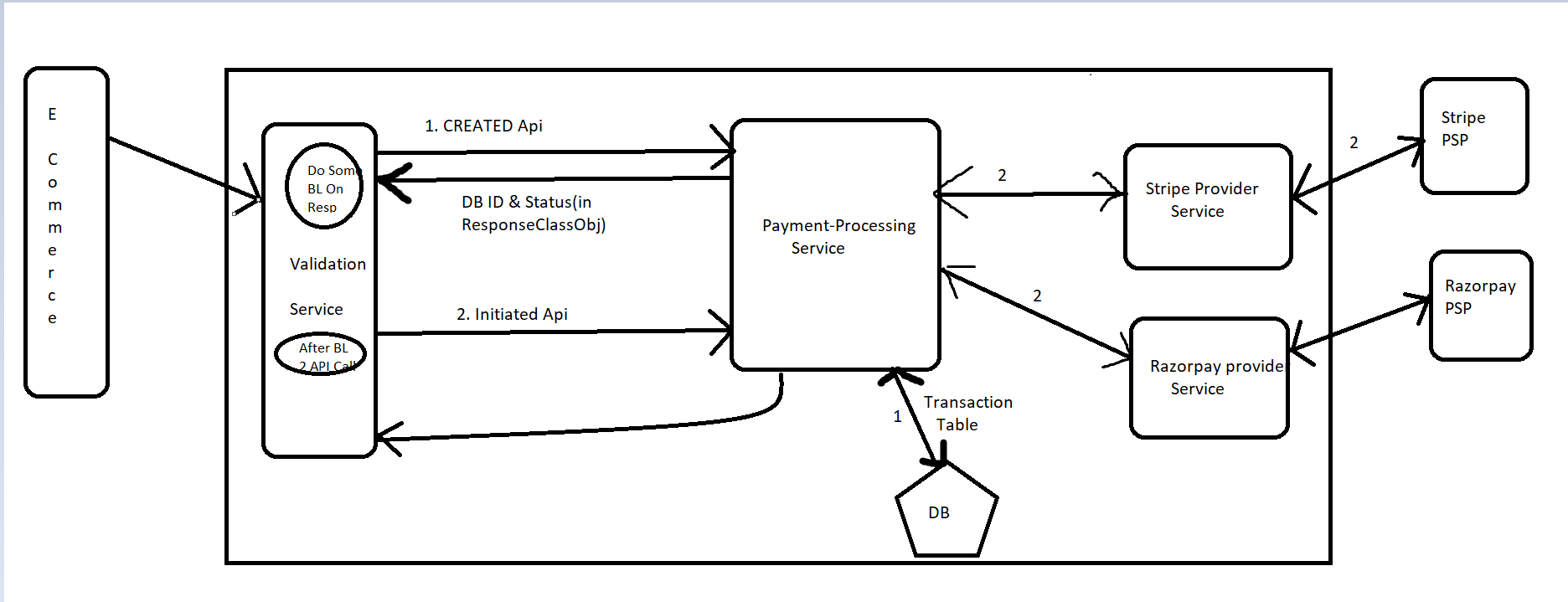
"status": "expired",

Customer is sitting on payment page and we call expire api in Backend.

If after that Customer click Pay button on UI page,it’ll show Expired Link Page

**============================================**

**PROJECT - FLOW DIAGRAM**



\*\*\*\*\*\*\*\*

E-commerece side will call Payment Gateway Backend through APi. That Api will go to first Validation MicroService of PaymentGateway Backend. Validation Service layer will fetch all the data from api, and it’ll call First API(CREATED) to Payment-Processing MicroService Layer. In Payment-Processing MicroService Layer it’ll check that with that data we can create a transaction to make a payment in PSP or not. If everything is correct then it’ll save the details in DB and in return response it’ll return the new entry ID to Validation MicroService Layer. In validation it’ll perform some BL then with remaining data it’ll Second API(INITIATED) to Payment-Processing MicroService Layer.

Payment-Processing MicroService Layer will Call Stripe Microservice layer, and from that layer Third Party PSP will be called to make payment. After payment the URL and Status code will be return to Stripe Microservices layer,from their it’ll go to Payment-Processing MicroService Layer and Validation Layer. From validation it’ll go to E-Commerece Side.

That means only one call will happen from E-Com side and only One response will be return to E-Comm Side. In middle we will do this many API call to verify data first and then initiate payment.  
If we will not call 2 APi(CREATED<INITIATED) and every thing we pass on only one api,then the failure risk will be high and if validation fail for creation then without checking other implement BL logic time will be in vain.

**CREATED Api Call (First Api for Created a Transaction):**

What core info, every payment(Stripe,Razorpay,Payment) will need….

//this part data we have to pass from backend side, it’ll not come from E-Comm Side or not from PSP Side

1. paymentMethod: APM

2. paymentType: SALE

3. provider: STRIPE

//this part will come From E-Com Side(Total Amount will pass by E-Comm)

4. amount [total amount]

6. Currency

//this all three ID will not be same,and not share with Other System

7. merchantTxnRef : E-Comm System

8. txnRef : Our PaymentSystem

9. providerRef :

//come from Ecomm

10. userId : Customer Login ID

11. status

Related Class and DB names in code:  
Transaction Pojo class, Payments.Transaction DB Table

\*\*\* PaymentId logic:

1. Every system will generate its unique Id. (E-comm, PaymentSystem(us), PSP)

2. When PaymentSystem, call any PSP, then provide paymentSystemId, not of our client’s id.

3. PSP will have its own id, and give back to us.

4. We need to map all the 3 Ids together.

Create Session API Call:

==================>

Required Params are,

//pass this param value from Payment-Processing-Service,no need to pass from E-Comm Site

Mode : Payment

//this all needs to pass from E-Comm Side to Payment-Processing-Service

line\_items[0][quantity]

line\_items[0][price\_data][currency]

line\_items[0][price\_data][product\_data][name]

line\_items[0][price\_data][unit\_amount]

line\_items[1][price\_data][unit\_amount]

// Have below parameters configured in Payment-Processing-Service

system. If ecommerce don’t pass these urls, then load value from our configuration. Else pick from ecomm request.

success\_url

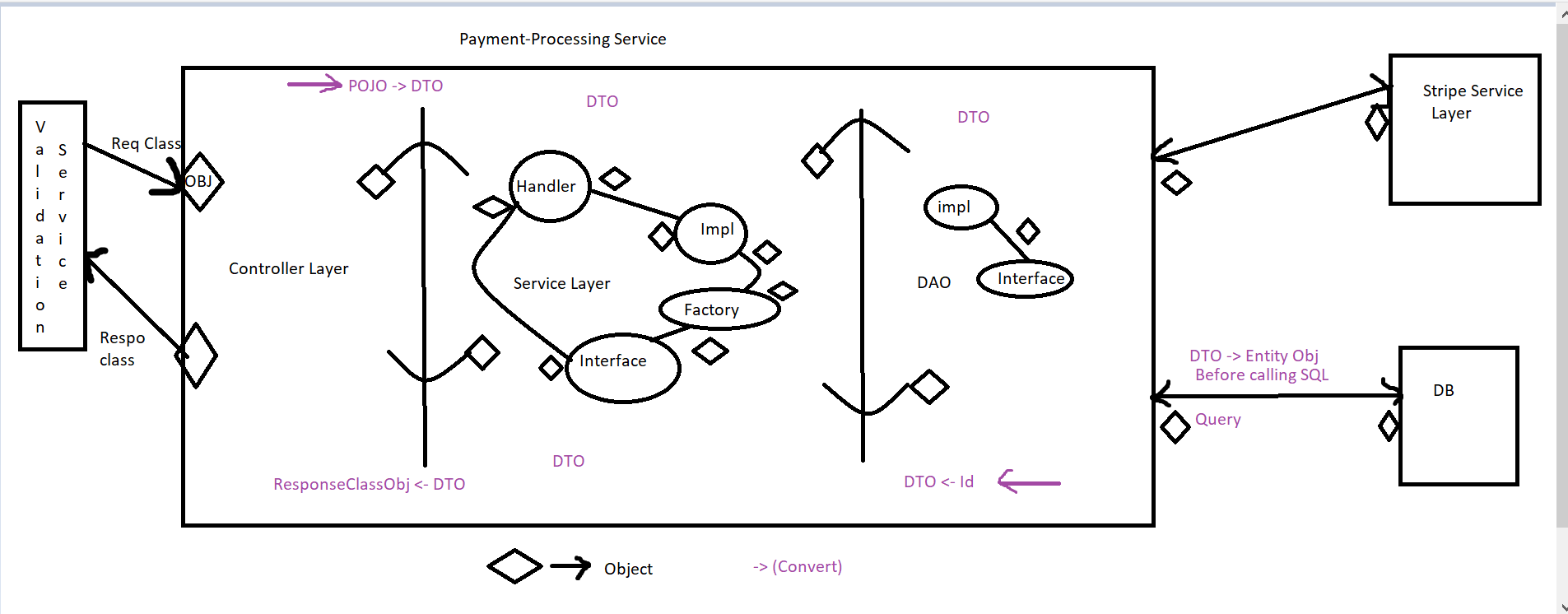
cancel\_url

payment-processing-service should expose 2 APIs to validation-service for invocation

1. CREATE transaction in DB

2. INITATE with Stripe PSP

**Create Session API Call** -data is required in INITIATE API(Second) call as a argument. Not in CREATED API(First) call.



**Package structure**

.controller

It’ll only call service layer and send the response back whatever coming from service layer

.service

- interface, implement, Handler,Factory ,other layer

.dao

- interface,impl, .. other layer

.constant

* enum,map,constant calss

.config

AppConfig

.utils

date conversation,

.pojo

(All incoming req/res) from validation service

All Outsider will contact this class,they can’t access other class

.dto

controller talking to other(=>) controller

controller => service

service => service

service => DAO

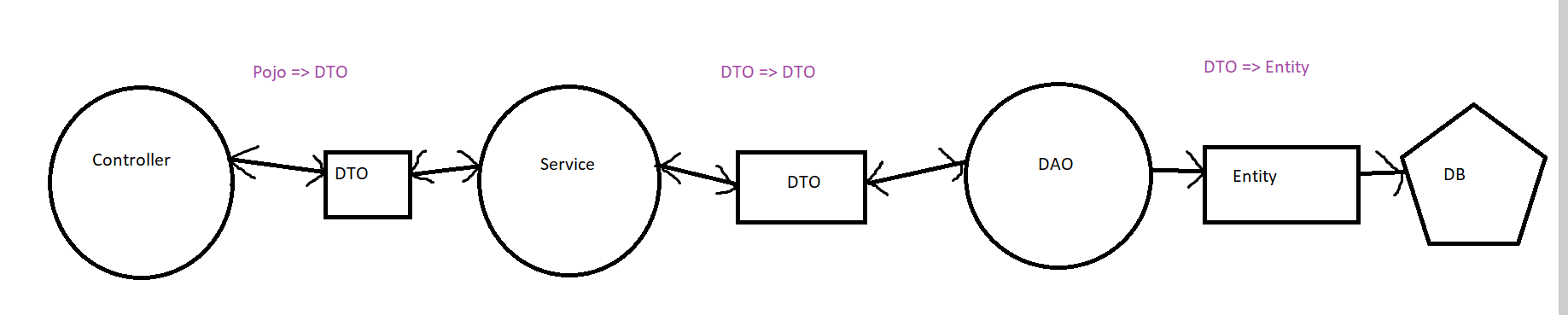
DAO => DAO

.entity

Java classes that can be mapped to database tables

.<Other service Layer name>

Their Req/Res



Rule For a Project

1. Pojo should never directly go to DB.

2. DB should always have only Entity writing not other Object writing/access

3. Never leak the entity to service layer.

4. Entities never move outside your service.

Don’t make any external service/layer calls with those objects and don’t return those objects as response.

* Pojo - not going to DB
* Entity cannot be leaked outside

\*\*\*\*\*\*\*

* What ever outsiders Req/Res coming to a Microservice,it’ll not go beyond Controller Layer. Controller Layer will be Abstraction Layer for the whole system. Outsider can’t access Service/Dao or any other BL. That’s why we’ll not write any BL in Controller class. Controller layer will handle every request properly and it’ll send to service layer.
* Pojo will be the main class in controller layer. We will not pass Pojo class obj to Service layer. RequestClass and ResponseClass will be written here,that we’ll share to invoker.
* Cause, if we simply dump the pojo object into service or in DB layer, then project security will be less. If anything bad happen to Pojo object,we’ll loss all the data that are coming from invoker side. Third, Hacker can hack this type of Project. Fourth,if project will go to large scale,we’ll create different Microservices for a single Layer. That time project tracking will be hard to maintain.
* So every time when we communicate we’ll convert to Pojo obj to DTO object(DTO class will be written differently in DTO Package) before sending it to Service.
* Inside service layer communication or Inter Dao Layer communication or To communicating with other different Service Layer(Stripe-Provider Microservice,Validation) we will share DTO Object only
* Before storing data to Db or before Writing SQL Query in DAO Layer,we have to convert DTO Object in DAO layer into Entity Object. That Entity Object will be Stored in DB. Entity classes will be written in different Entity Package.