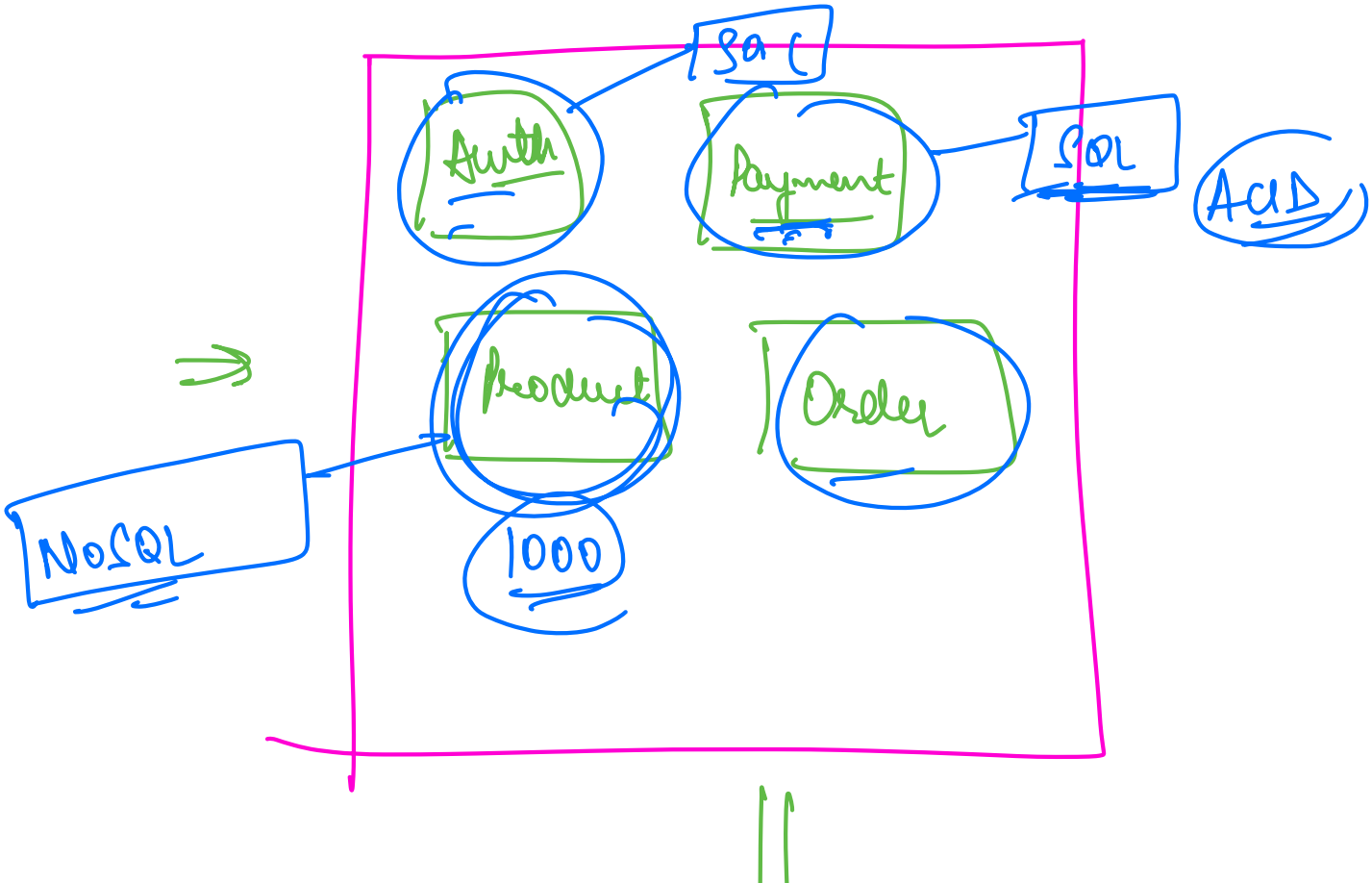
Agenda

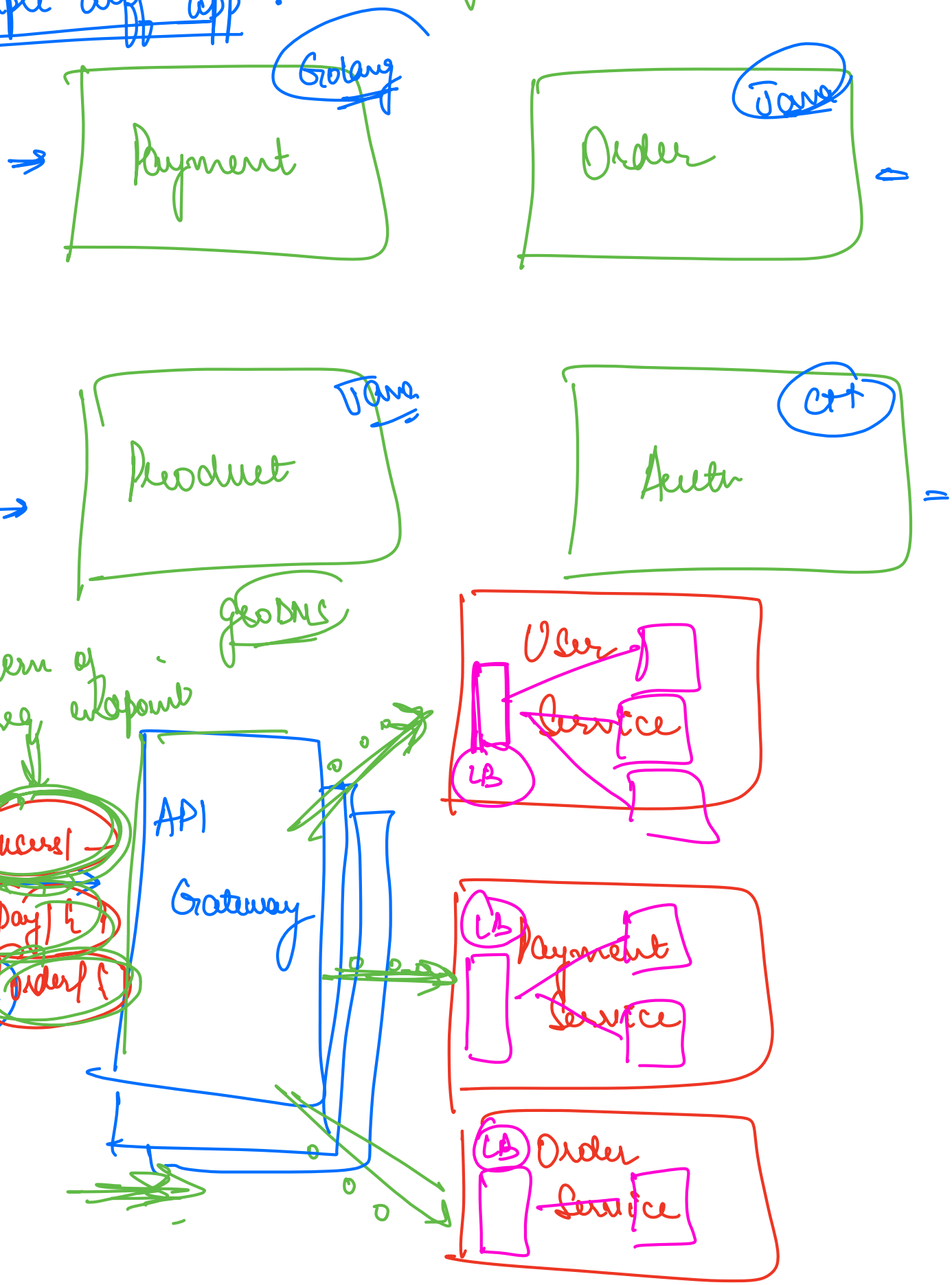
I
1. Project Req
2. HLD
3. Micro services

II
1. Introduction to Spring
2. Dependency Injection
3. Spring Boot
4. Start Project
5. Dependency Injection in Action



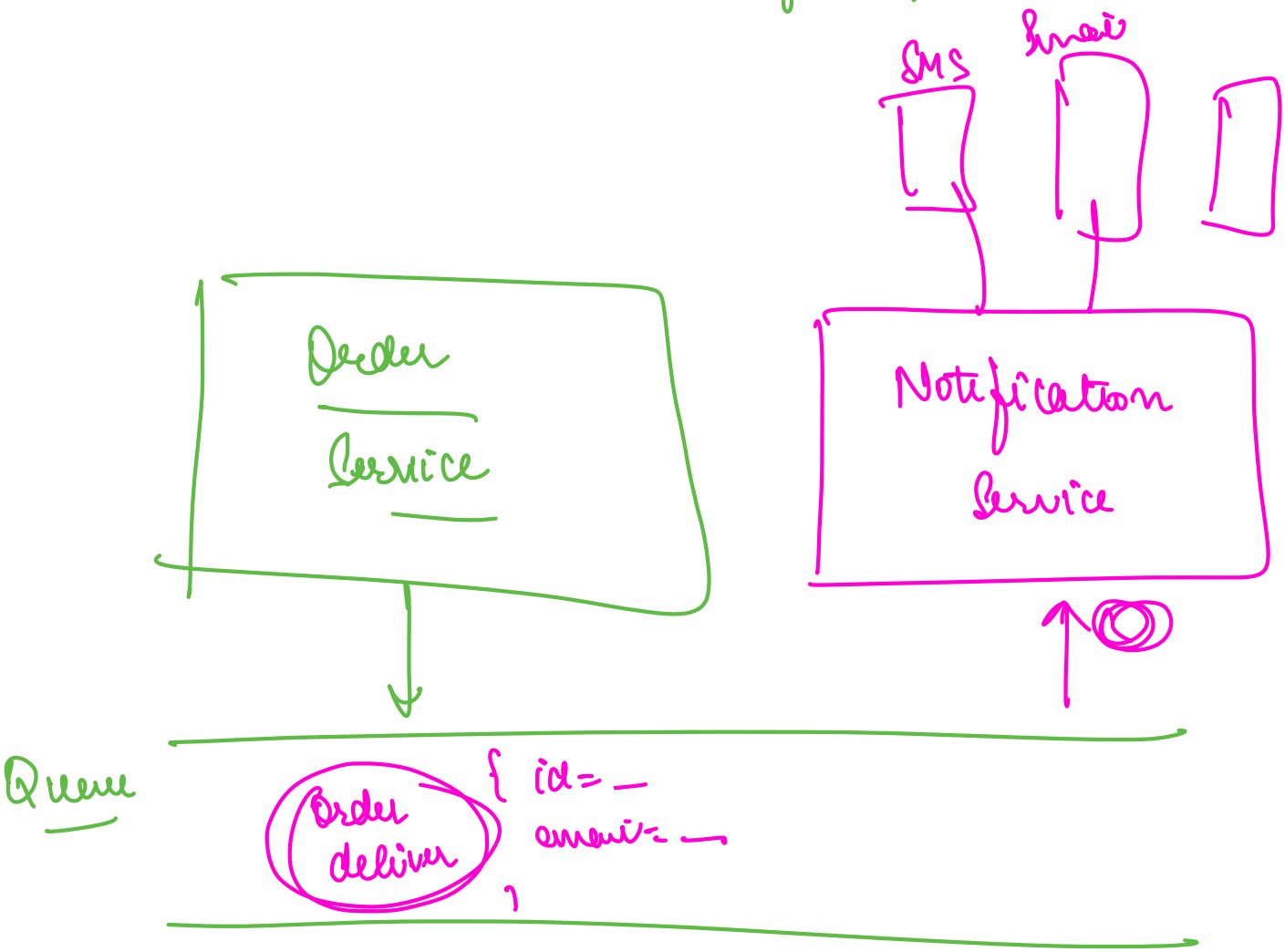
Divide into

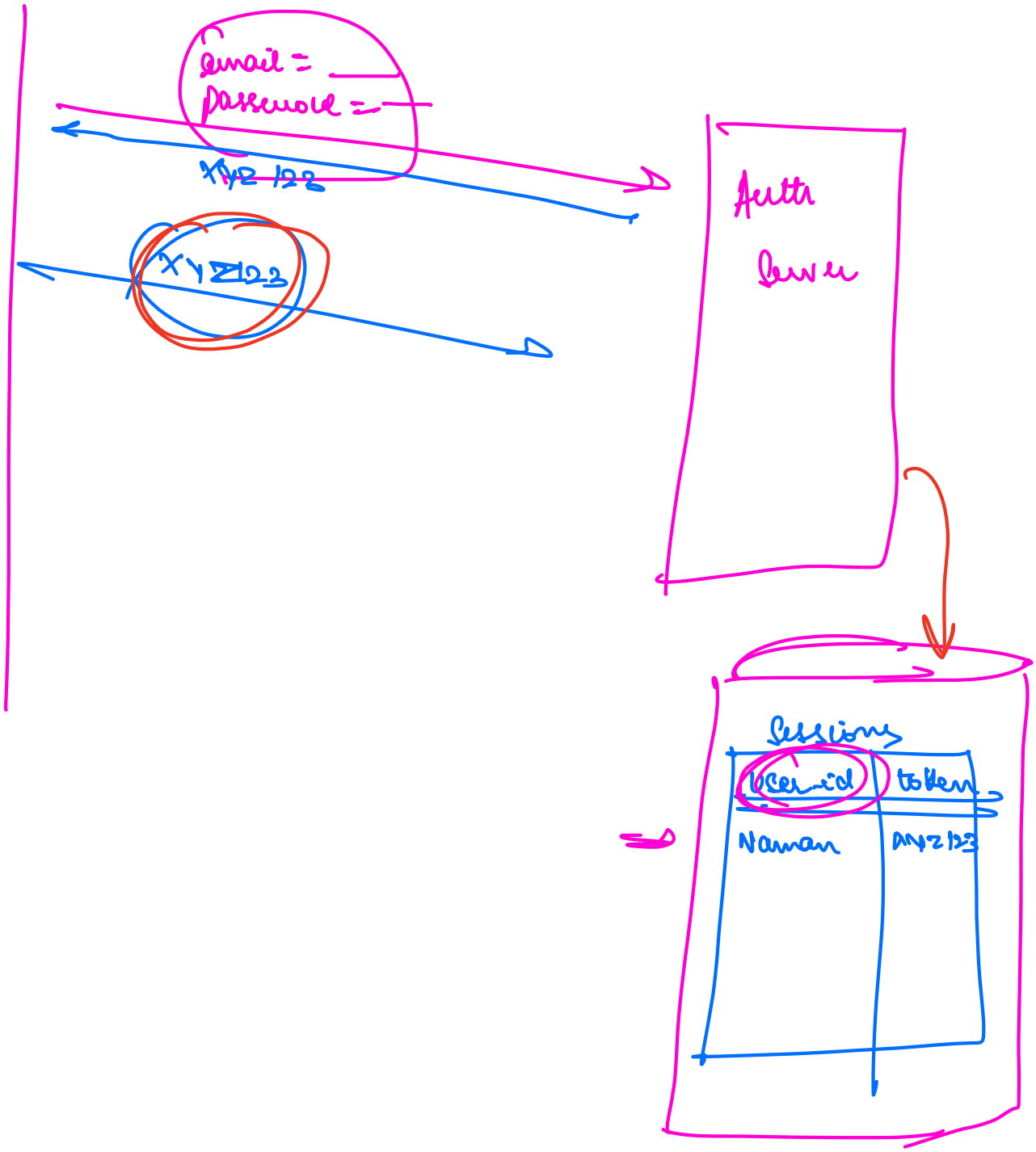
Multiple diff app^{n}:

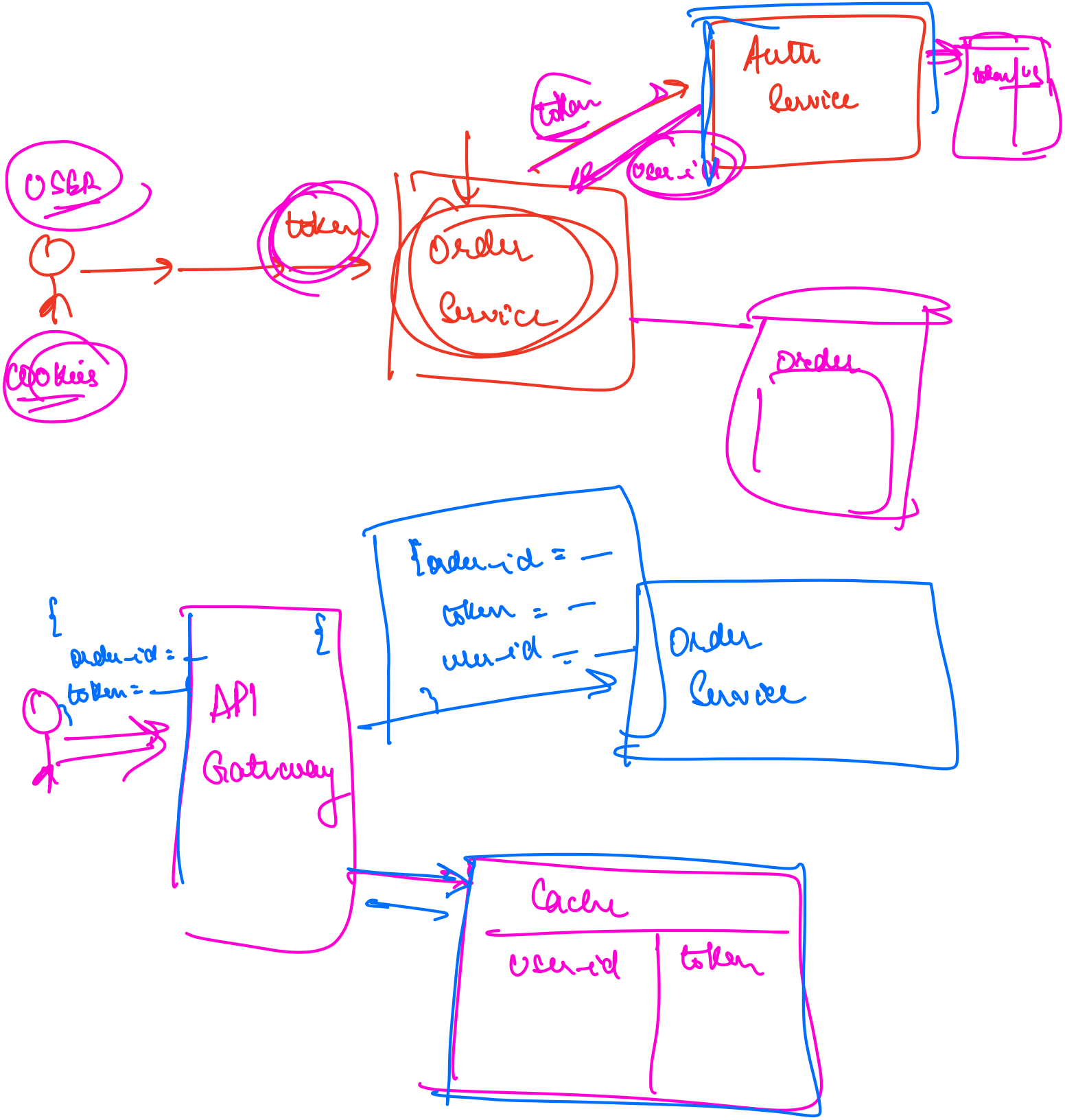


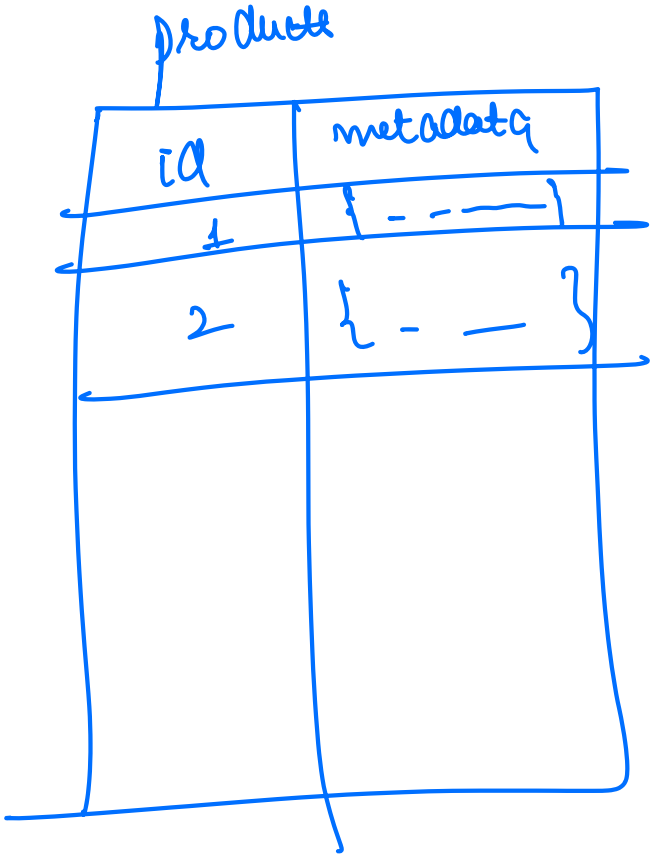
API Gateway ⇒ A system that forwards the request to correct micro service

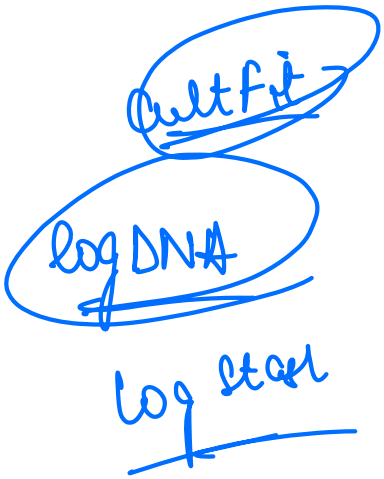
Load Balancer ⇒ A system that balances the load amongst app servers.




email =
password =
XYZ 123
XYZ123
Auth
Server
Sessions
user_id
token
Naman
XYZ123


Auth Service
token
user-id
USER
token
Order Service
Cookies
Order
{order-id = —
token = —
user-id = —
}
{
order-id =
token =
API Gateway
Order Service
Cache
user-id
token





Intro to Spring

J2EE ⇒ Java Enterprise Edition

Framework for enterprises to build Java app^{n} in a better way ⇒ Spring

Spring framework

① Core Concept

- Dependency
- Dependency Inversion of Control
- Config Mgmt

② Projects (built upon Spring framework)

Spring Boot

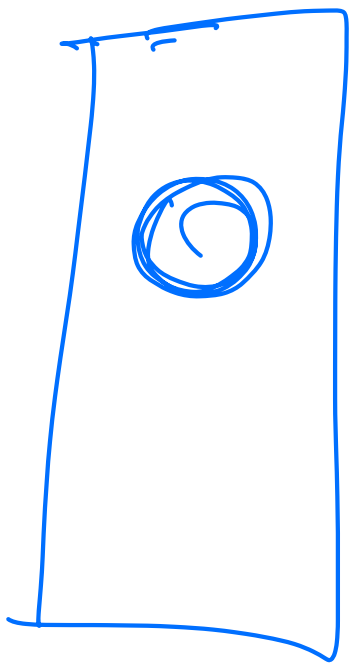
→ Spring MVC

→ Spring Cloud

→ Spring Auth Server

→ Spring Data JPA

→ Spring Security



① Dependency Injection

Order Manager

Payment gateway

onOrderPlaced () {

pg.createBill();

}

→ every class has dependencies

[yourselves creating obj of our dependencies

```
class OrderManager {
    RazorpayPaymentGW pg = new RazorpayPG();
}
```

Issues

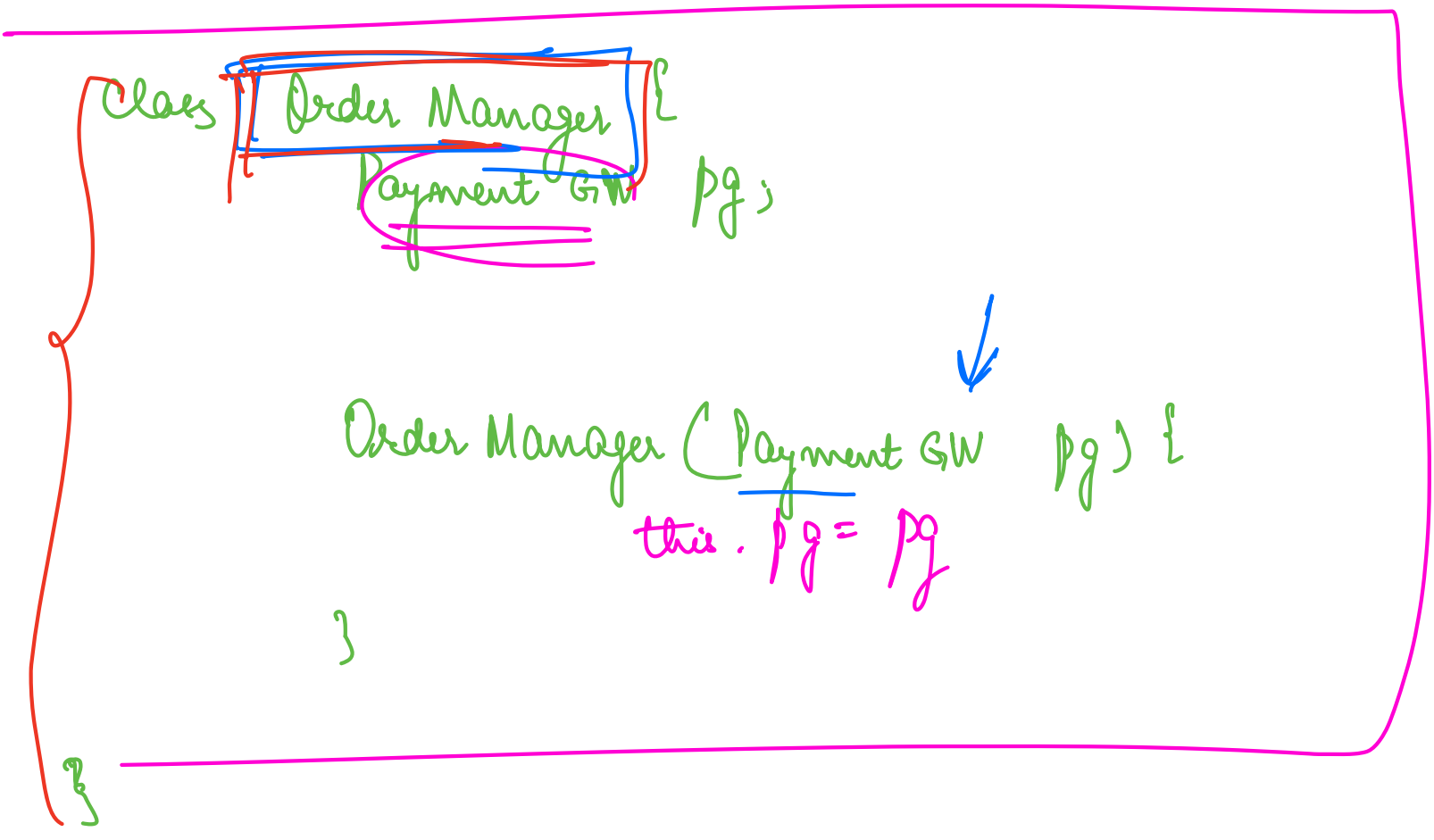
(1) Same obj may be needed somewhere else in my apps as well

⇓

Ideally wanted to make this a singleton

(2) violates dependency inversion.

→ no 2 conc classes should directly talk to each other. Talk via interface.



Dependency Inj ⇒ instead of creating dependencies yourself, getting the dependencies injected into you

```
new OrderManager(new RazorpayPG());
```

Spring Application Context

(Spring Container)

→ it avoids having 2 instances of the same class.

→ it creates all objects and their dependencies itself on your behalf and stores them in its container

(List<>)

Maps.

→ any utility class that you will create, Spring will create their obj as well as the objects of its dependencies auto.

Earlier

```
Order Manager {
    PaymentGW pg = new RazorpayPG();
}
```

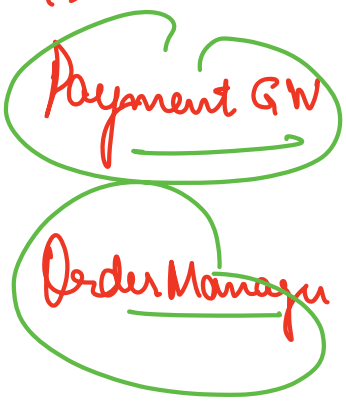
(V1)

```
Order Manager {
    PaymentGW pg;

    OrderManager(PaymentGW pg) {
        this.pg = pg;
    }
}
```

Main {

psvm() {



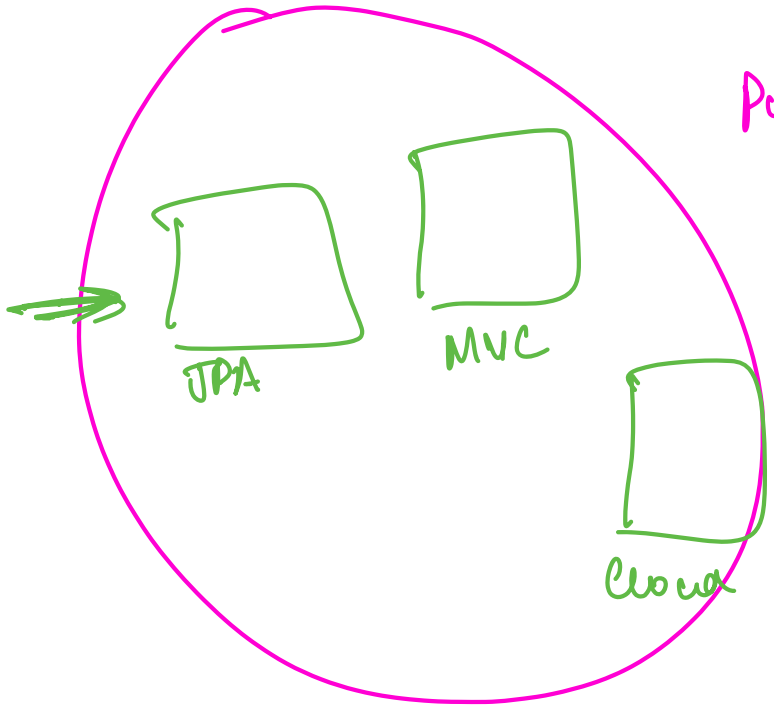
pg = new RazorpayPG();

om = new OrderManager(pg);

}

}

Inversion of Control ⇒ instead of you creating obj and their dependencies yourself, asking Spring to automatically create on your behalf.



Earlier, all spring proj that you used to add on Spring framework, you had to configure them yourself.

→ this config used to be very tedious

→ this used to happen via XML files!!!

↓ Spring Boot

→ I want to create a Spring app^{n} with support of

1. talking to SQL DB
2. ——— Kafka
3. ——— Security
4. ——— Microservices

Spring Boot will automatically create a ready to run app^{n} with each of these already configured with sensible defaults.

→ while providing a very easy way to override them.

Make it easy for people to work with Spring FW and its projects

Spring Boot W/O Spring ✗

Spring W/O Boot ✓

Scaler → RoR

H/W

→ Complete guide on creating first Spring Boot App's