Agenda

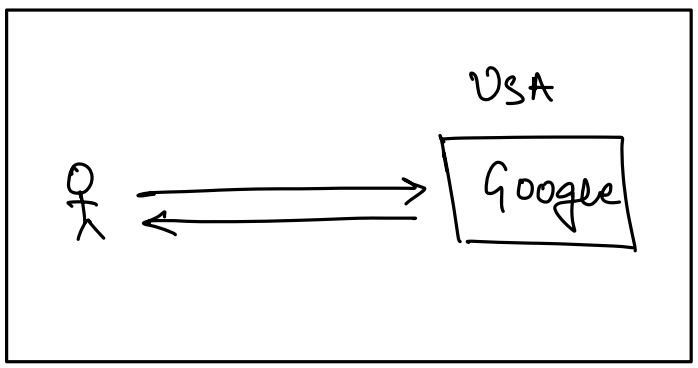
- → Monolithic (vs) Microservices
- → What are API's
- → REST
- → Stateful (vs) Stateless API's
- → MVC architecture
- → Login & SignUP API's
- → How payment infrastructure works
- → Implementing Payments API.

Ecommerce.

⇒ Outcome :

Microservices based working project in your resume with lot of advanced concepts.

100M.



① ⇒ Bottleneck

Slowest part of your system.

waiting time ↑

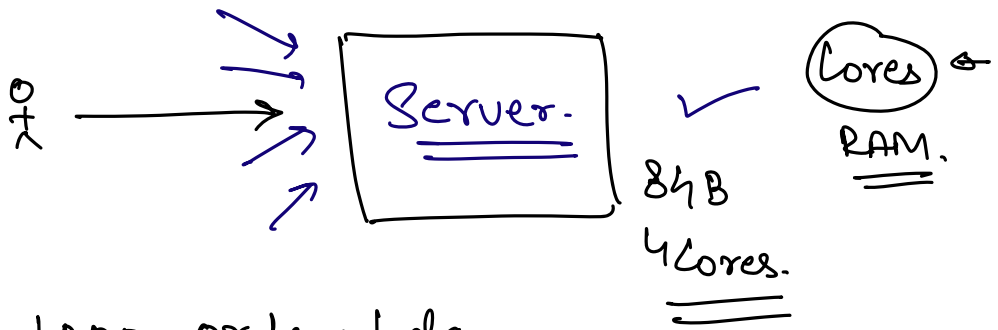
② SPOF. ⇒ Single point of failure.

⇒ If this m/c goes then complete YT will go down.

Flipkart (2008). (10-20 orders/day)

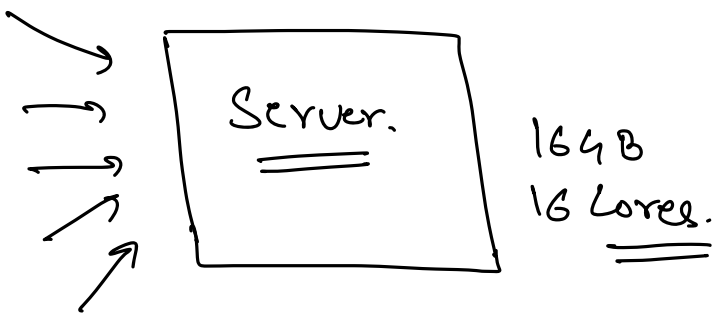
Sachin

⇒ 1 server to serve the traffic.



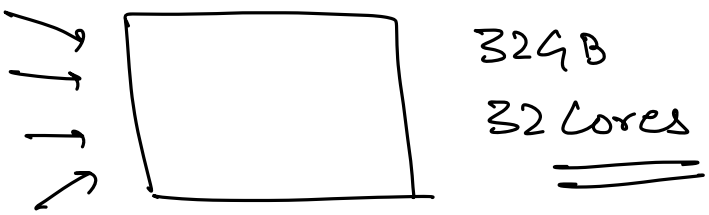
2010. ⇒ 1000 orders/day

⇒ Increase the capacity of m/c.



2012. 10000 orders/day

⇒ SPOF.



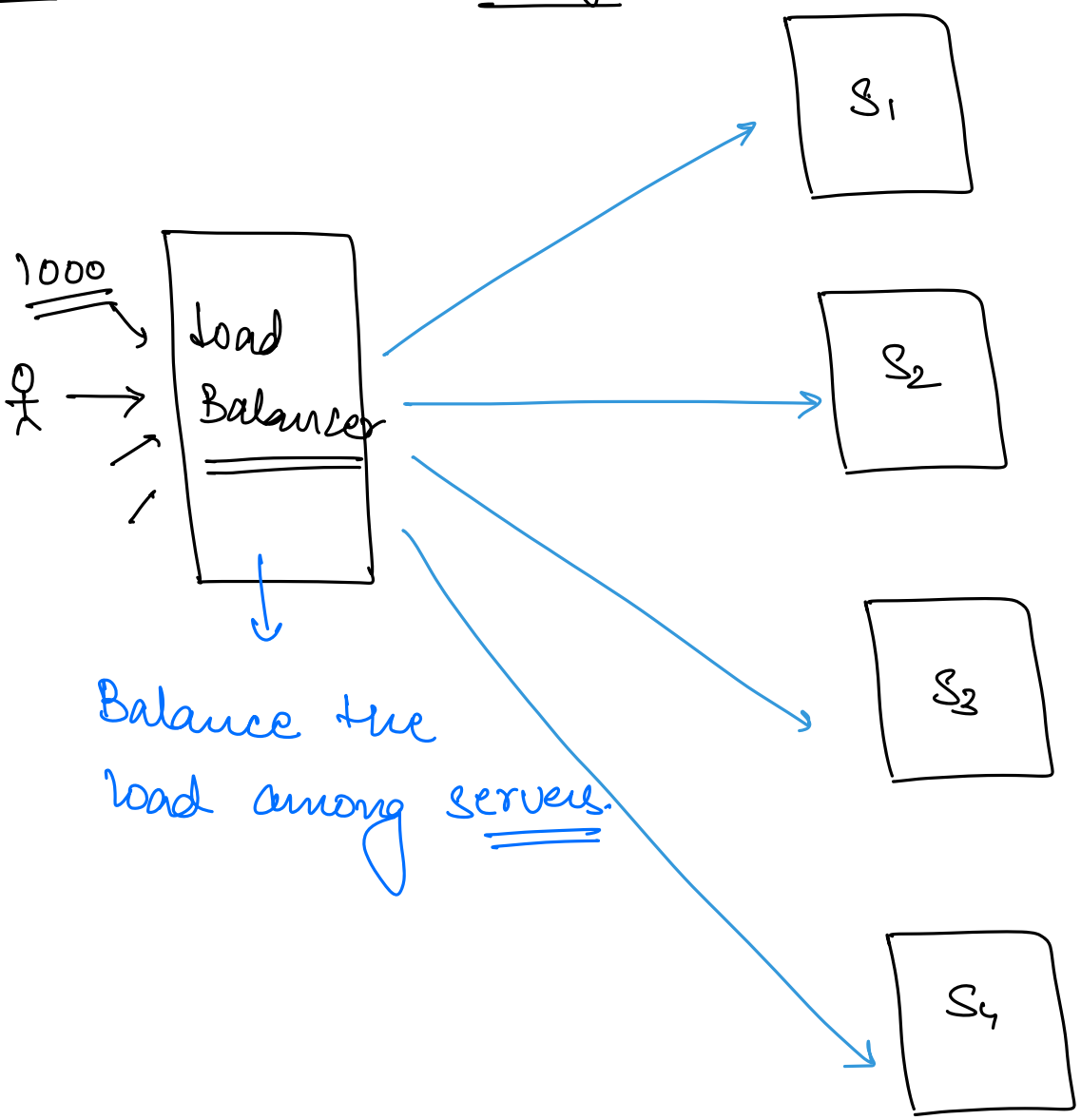
⇒ Can we increase the capacity of a single m/c infinitely. ⇒ NO.

⇒ Vertical Scaling: Increasing the capacity of same m/c

↳ has some limitation.

⇒ Soln: Add more machines. No SPOF

2015: 200,000 order/days.

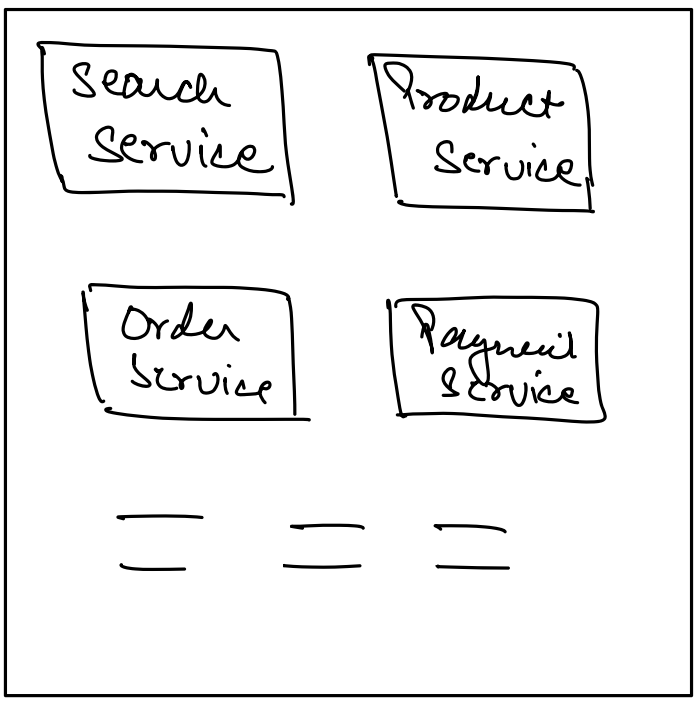


Horizontal Scaling

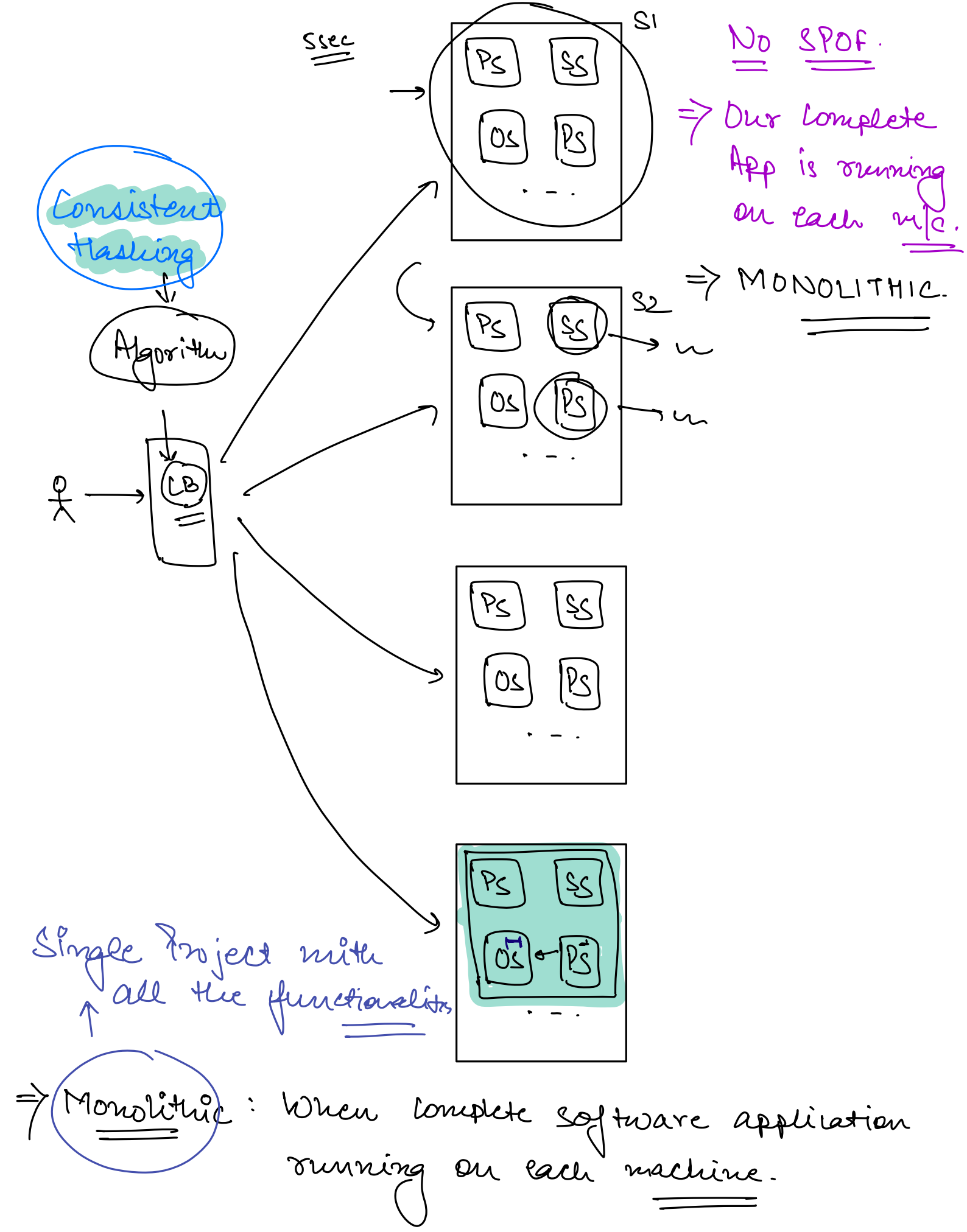
⇒ Keep on adding more & more m/c to the infra as required.

⇒ Flipkart / Amazon

→ ProductService
→ UserService
→ SearchService
→ OrderService
→ PaymentService
→ Cart Service

→ 

⇒ Monolithic.



No SPOF.

⇒ Our complete App is running on each m/c.

⇒ MONOLITHIC.

⇒ Monolithic : When complete software application running on each machine.

Cons of Monolithic.

1) No selective scaling →

PaymentService ⇒ 2 m/c
OrderService ⇒ 3 m/c
SearchService ⇒ 10 m/c

2) No tech stack flexibility.

3) High Deployment time

4) A small bug can make the entire Appⁿ down.

Pros.

1) Single Deployment.

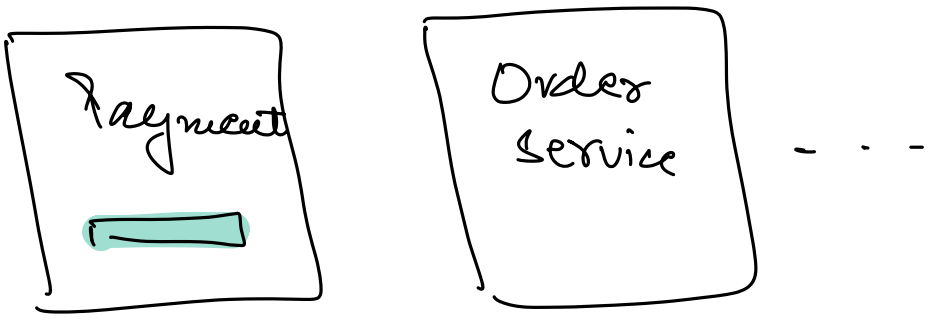
2) No n/w call, low latency.

⇒ MICROSERVICE.

Each service is an individual Project

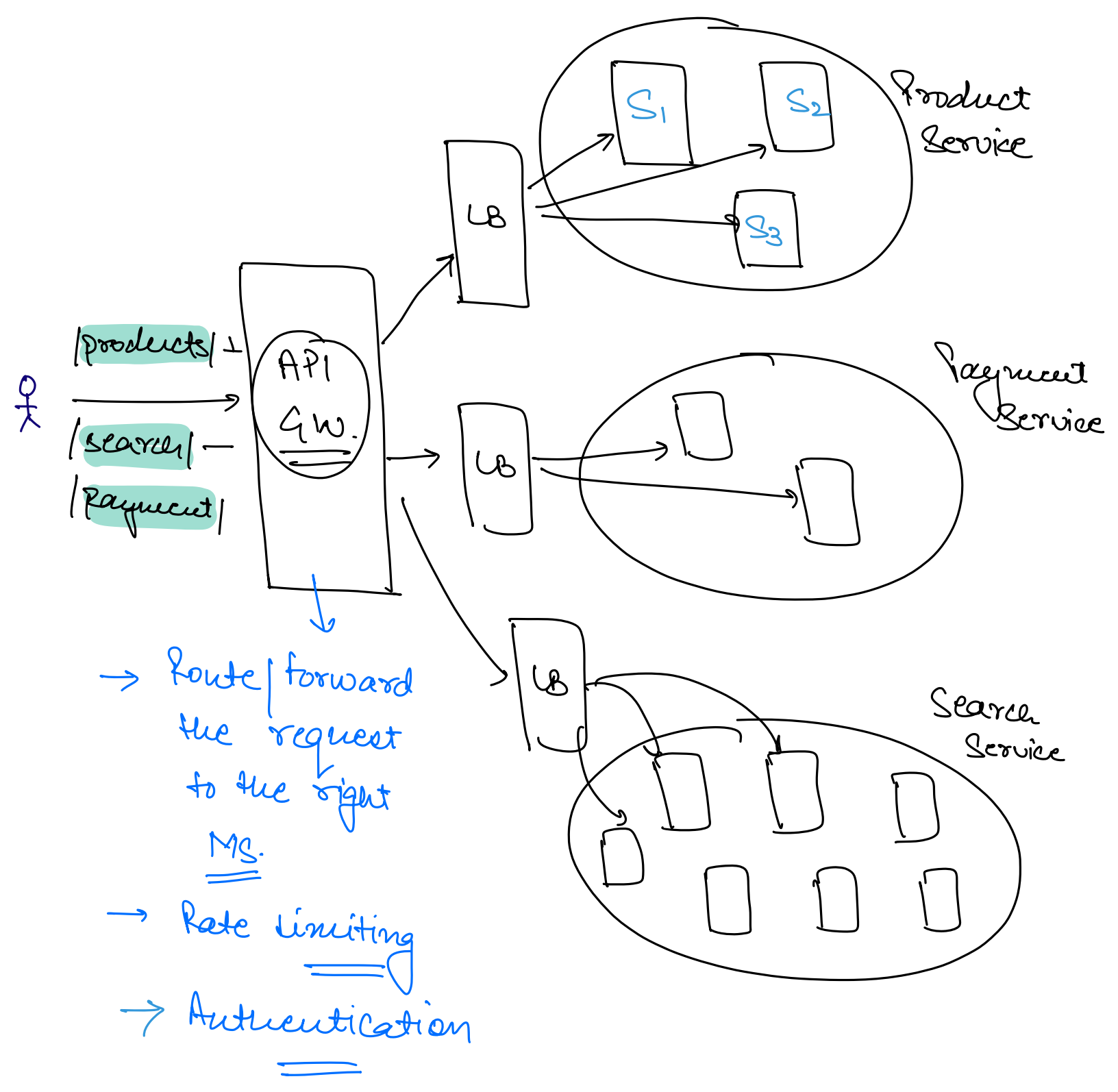
→ ProductService → 5 servers.
→ OrderService → 3 servers.
→ PaymentService → 2 servers.
→ —
—
—
—
—
—

Microservice : When the S/w system is broken down into individual services & each service runs individually.



⇒ SRP : Single Responsibility Principle

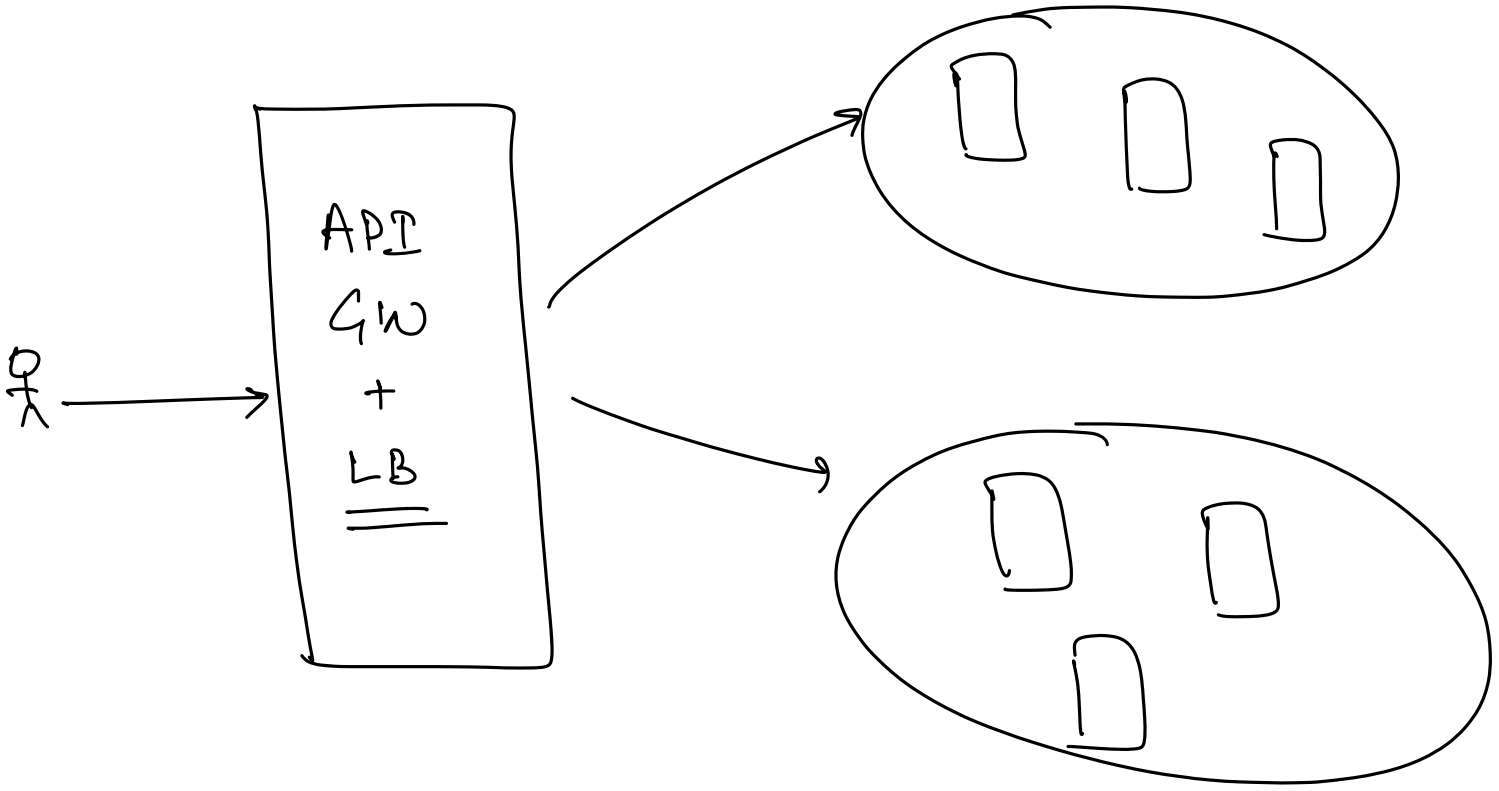
Microservices. : Each service should have a single responsibility.



→ Route / forward the request to the right MS.

→ Rate limiting

→ Authentication



⇒ MICROSERVICES.

- ProductService
- UserService
- PaymentService
  - Razorpay
  - Stripe.

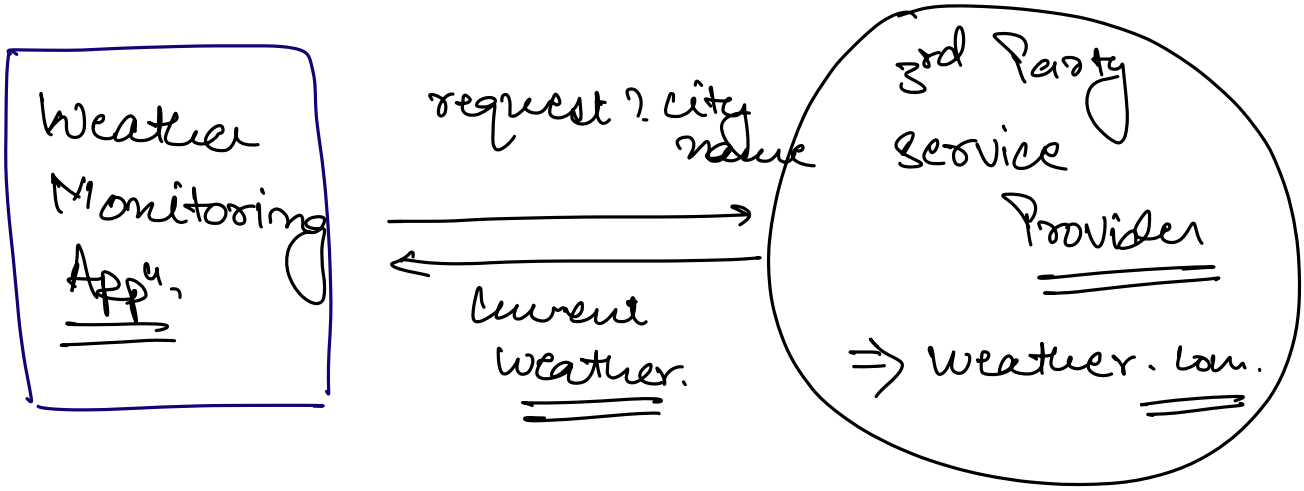
JAVA + SpringBoot.

API's.

↳ Application Programming Interface.

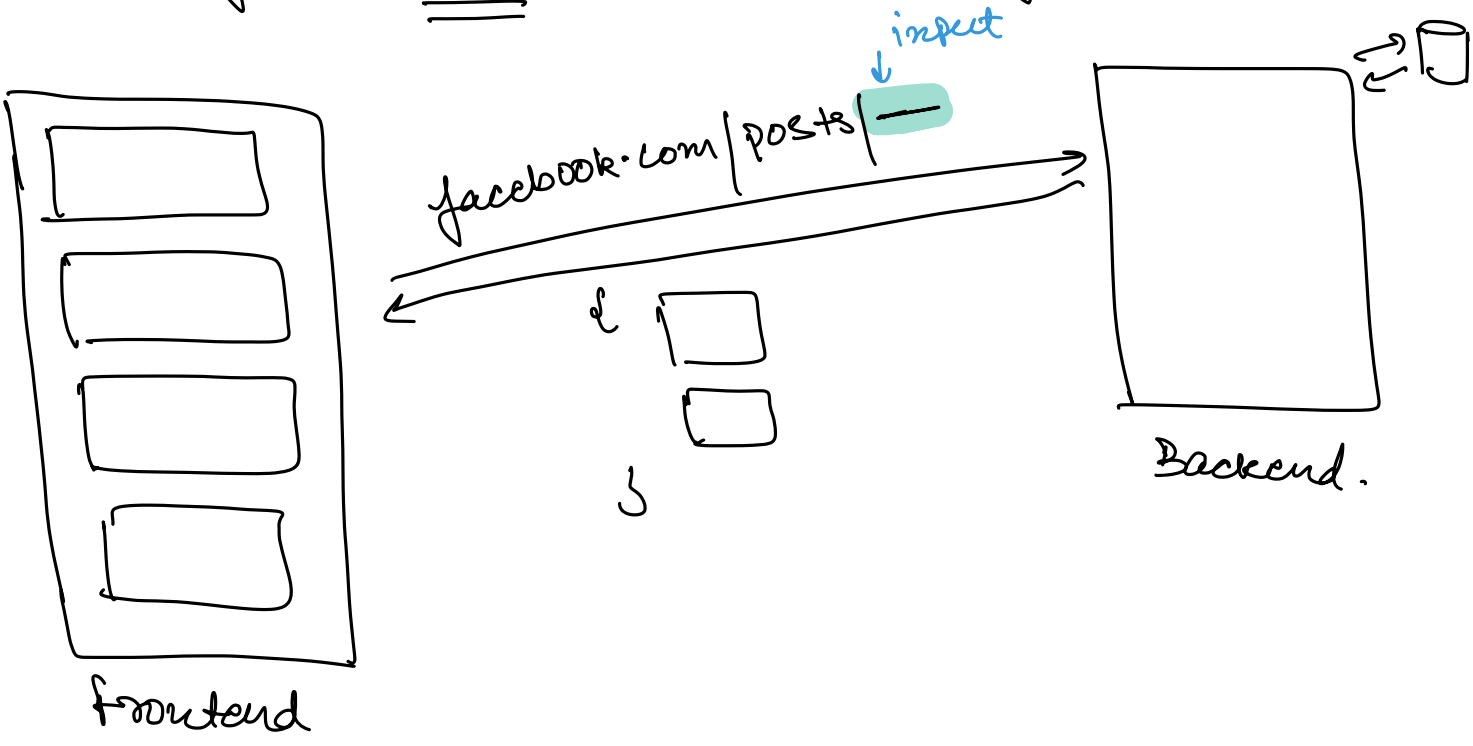
↓

Contract



1 $ | API call.

⇒ API : functionalities provided by companies to fetch Data.



⇒ flipkart.com/search/ —

flipkart.com/products/ —

flipkart.com/orders/ —

Endpoints.

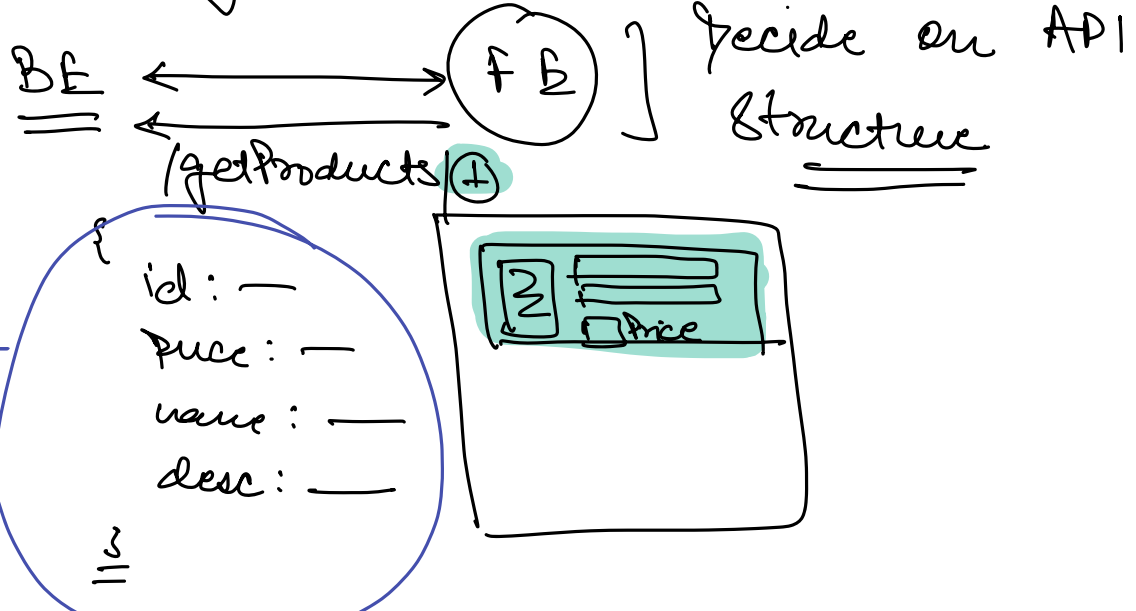
⇒ API : Set of Methods | URL's | endpoints | functionality provided by the application in order to interact with the data.

(get | update | create | delete) ⇒ CRUD

↓

Read

⇒ Functionality



JSON Object:

{"id":1,
"title":" Fits 15 Laptops",
"price":109.95,
"description":"Your perfect pack for everyday use and walks in the forest.
}

API's

- Endpoints
- Input
- Outputs.

SOLID.

⇒ REST

RE : Representational
S : State
T : Transfer.

⇒ Good coding conventions to build API's.

products → Resource.

orders

REST Practices.

1) All the API's should be structured around the resources that they are interacting with.

   ↓

   Objects. | Entities.

   /users/

   /products/

   /orders/

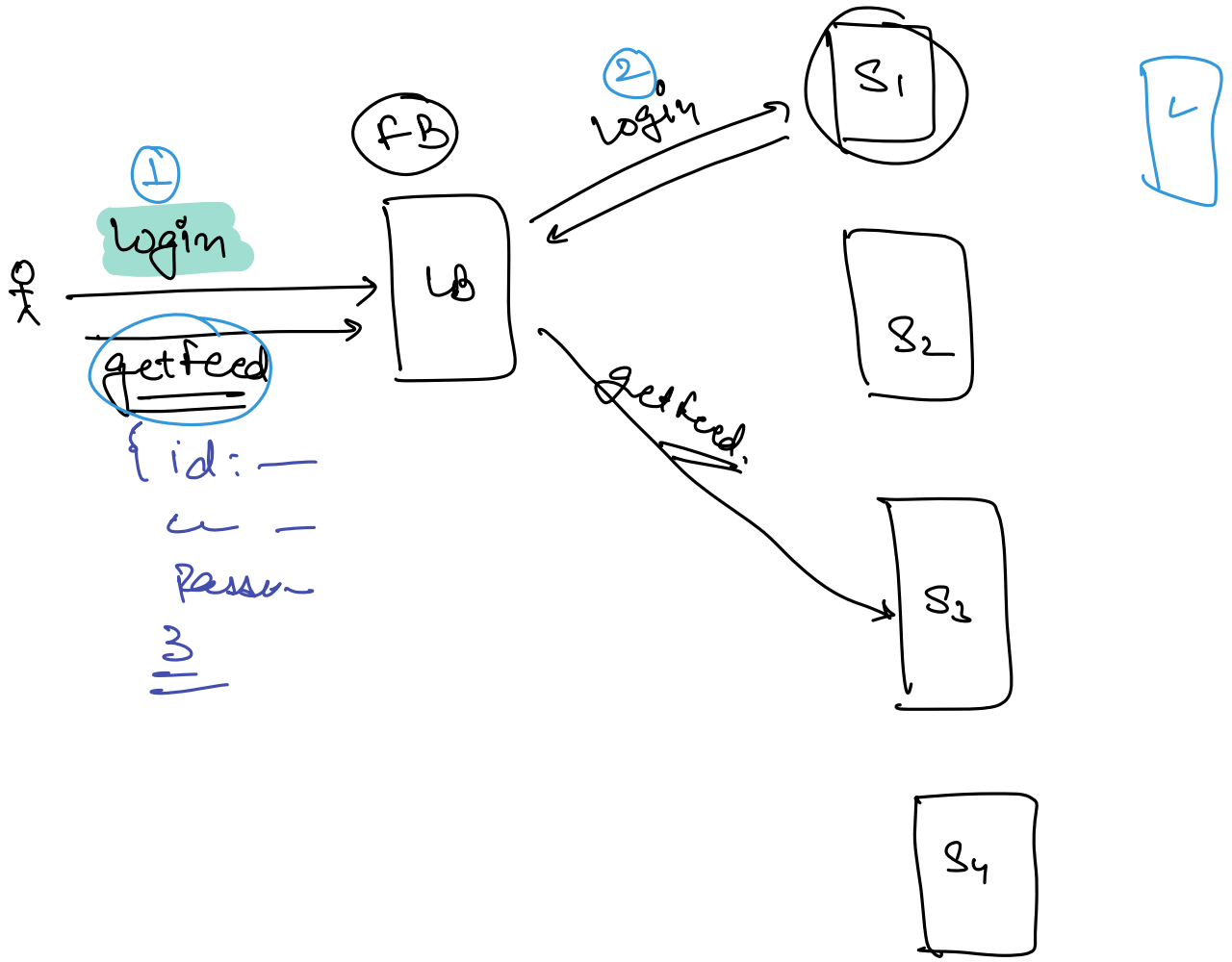
2) Type of action should be ideally specified via the HTTP Method instead of giving in the URL.

   ↓

   GET
   POST
   PUT
   PATCH
   DELETE.

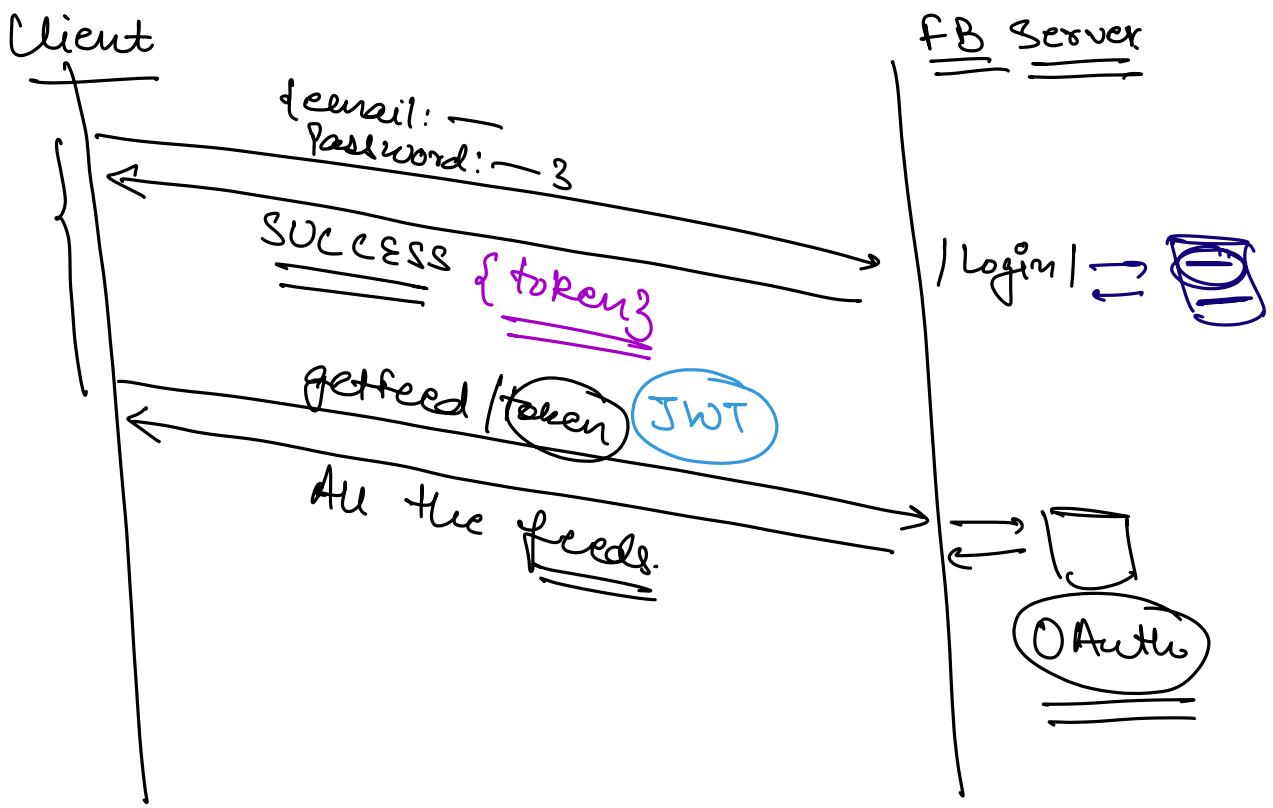
Interview-

3) REST API's should be stateless.



⇒ Each request should be completely independent & self sufficient in itself.

⇒



⇒ Stateless APIs.

⇒ Stateful.

↳ In the new request, we'll not send the required data, server should get the data from my previous request.

Read-

⇒ FTP

↳ File Transfer Protocol.

(4) There's NO restriction on data type of Response.

↳ JSON ⭐ ⇒ easy to read.

↳ XML

```
{
    id: ____
    name: ____
    age: ____
    email: ____
}
```

⇒ /src/

models/

Controller/

services/

repositories/

} Division of Responsibility

⇒ Well Structured Code.

MVC.

- Maintainable ⇒ Current code should keep running.
- Extensible ⇒ Adding new features.

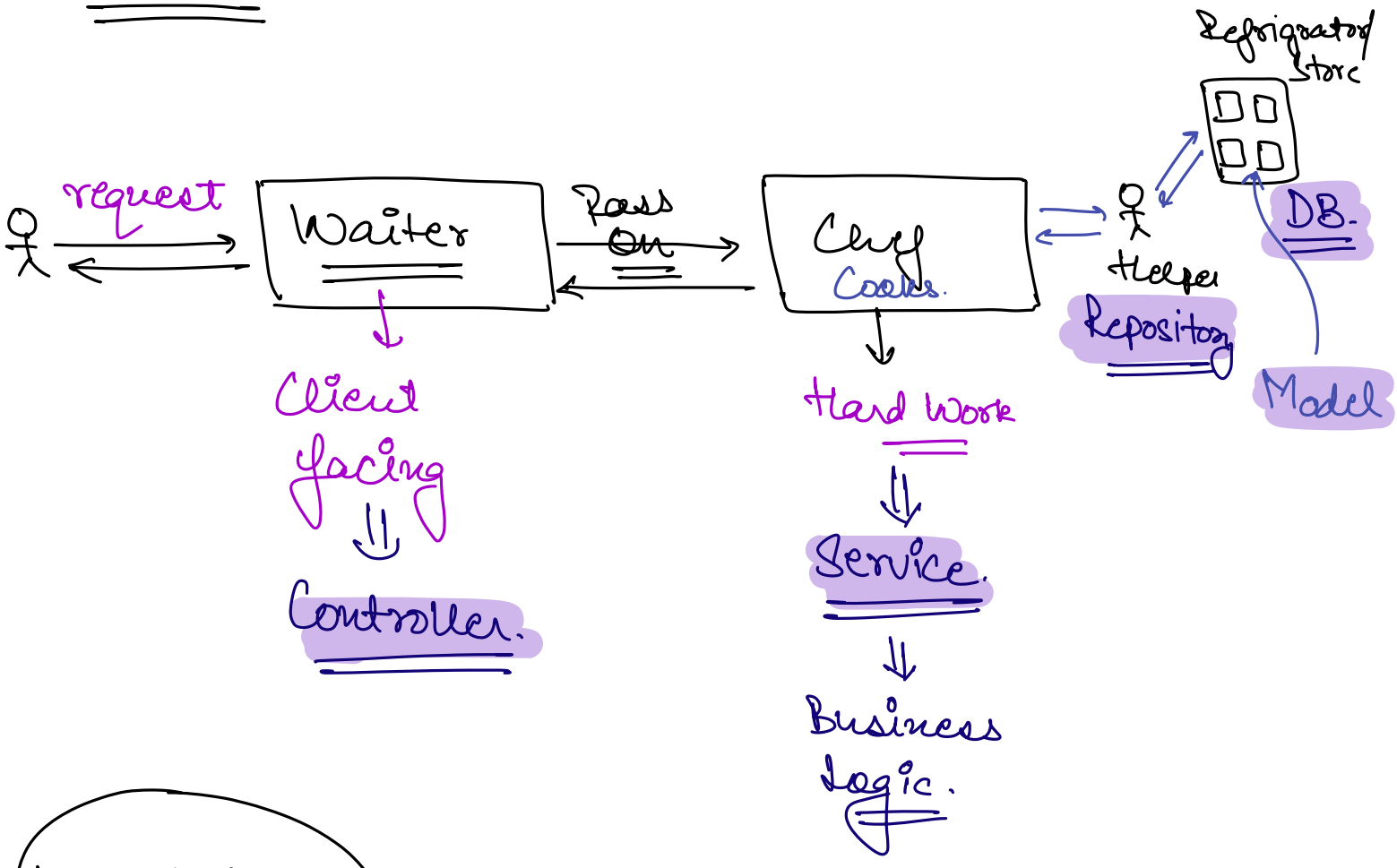
MVC.

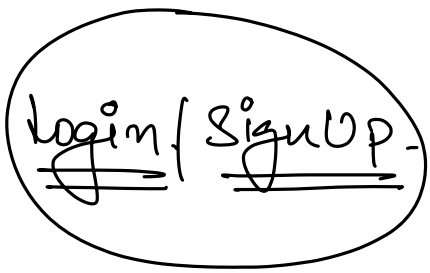
Model

View × UI

Controller.

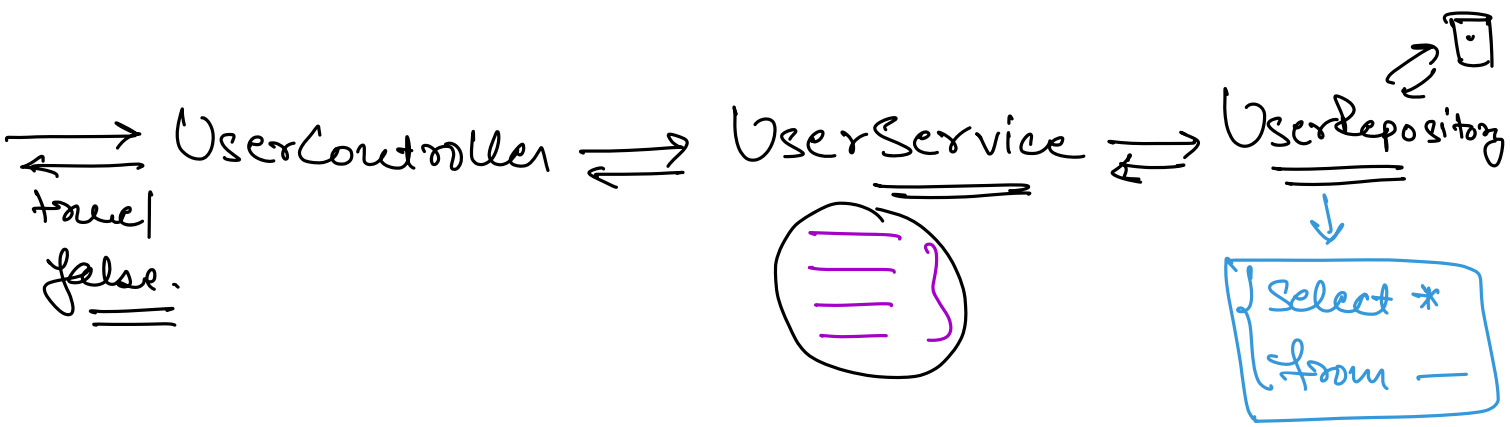
Restaurant.





User.

Login → input email password.



⇒ Next Class.

1) Implement more API's in ProductService

2) UserService < login / SignUp

3) LoadBalancer + API GW.

4) PaymentService.

⇒ Recap.

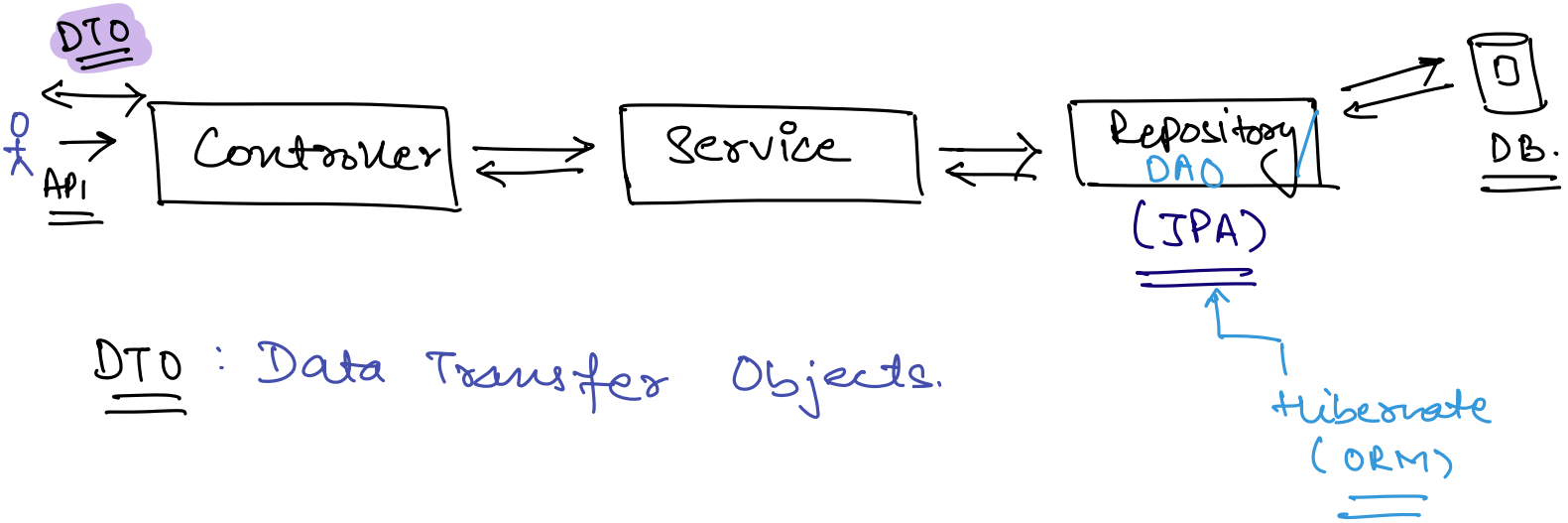
Microservices (vs) Monolithic.

Amazon Prime Videos moved from Micro to Mono.

⇒ ProductService

→ Product Information.

→ All the Product related API's.



DTO : Data Transfer Objects.

⇒ UserService

↳ login | signUp