

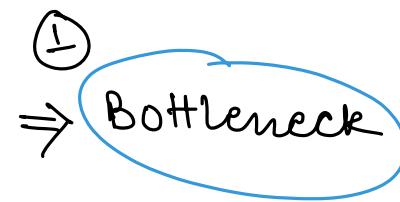
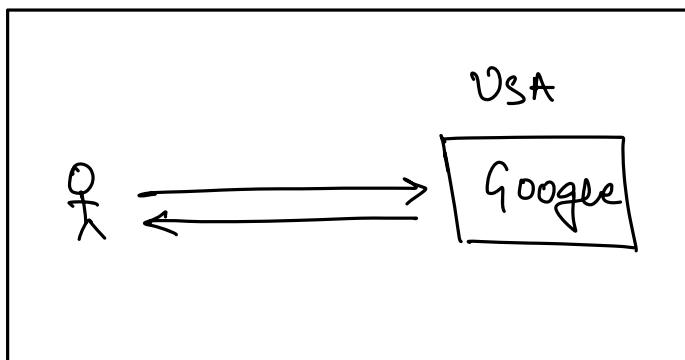
Agenda

- 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.
- } E-commerce

⇒ Outcome :

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

100M.



Slowest part of
your system.

Waiting time ↑

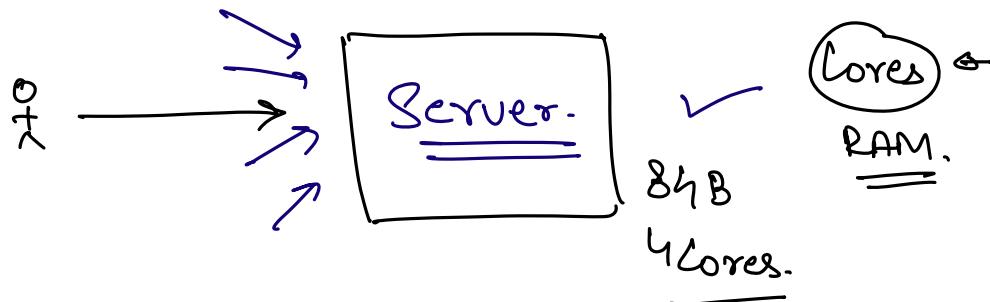
② SPOF ⇒ Single Point of failure.

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

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

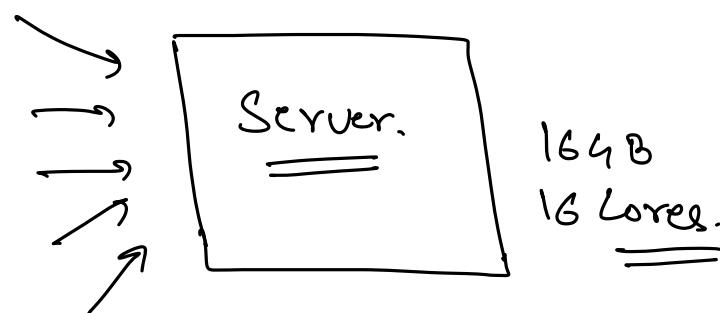
Sachin

⇒ 1 server to serve the traffic.



2010. ⇒ 1000 orders/day

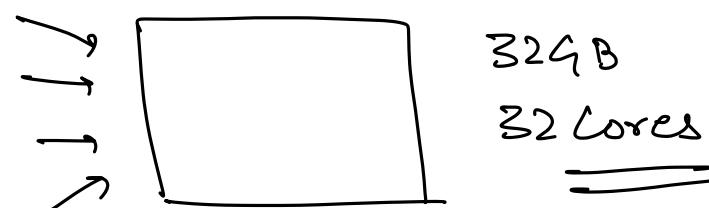
⇒ Increase the capacity of m/c.



2012.

10000 orders/day

⇒ SPOF.

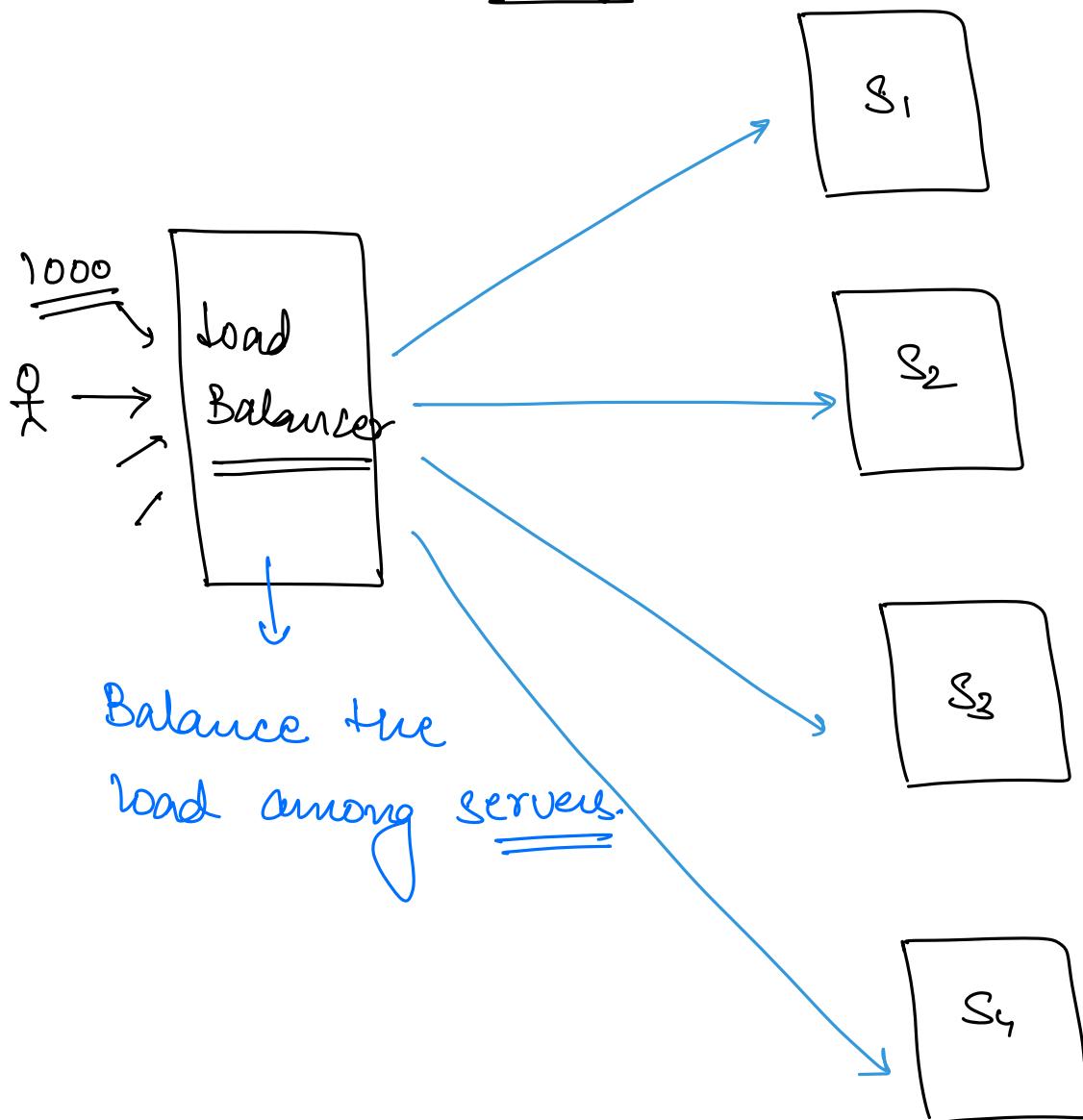


⇒ Can we increase the capacity of a single m/c infinitely. ⇒ No.

⇒ Vertical Scaling: Increasing the capacity of some m/c
↳ has some limitation.

⇒ Solⁿ: Add more machines. No SPOF

2015: 200,000 order days.

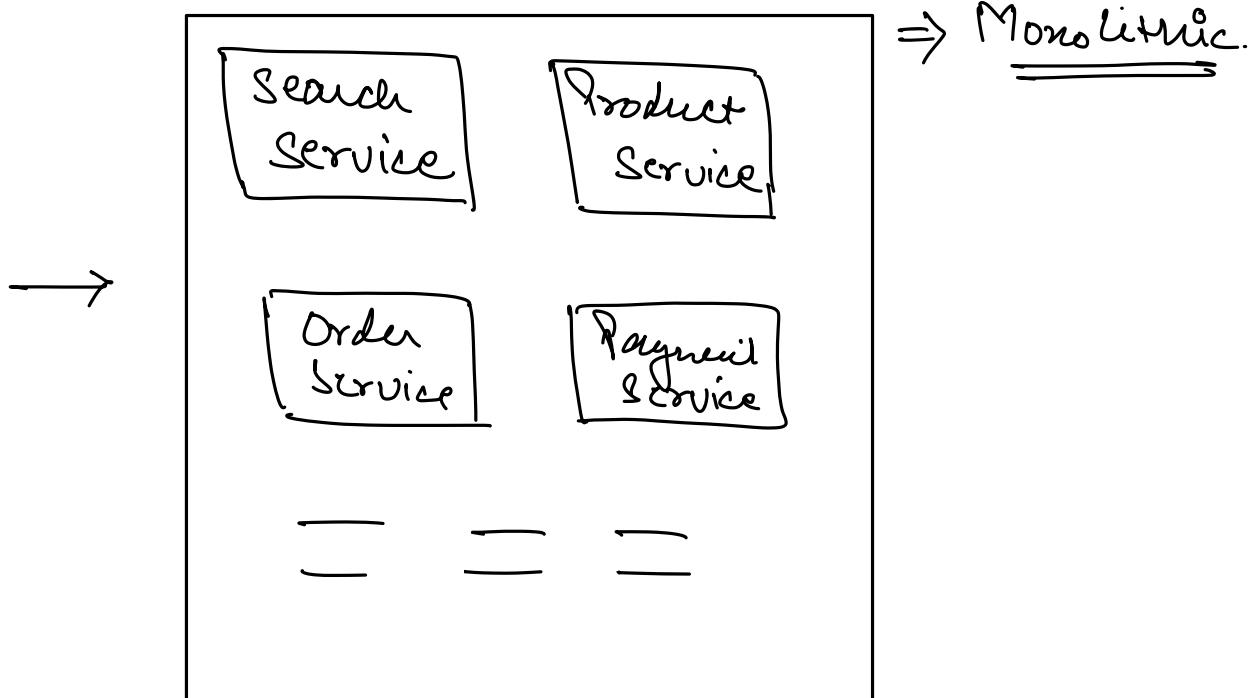


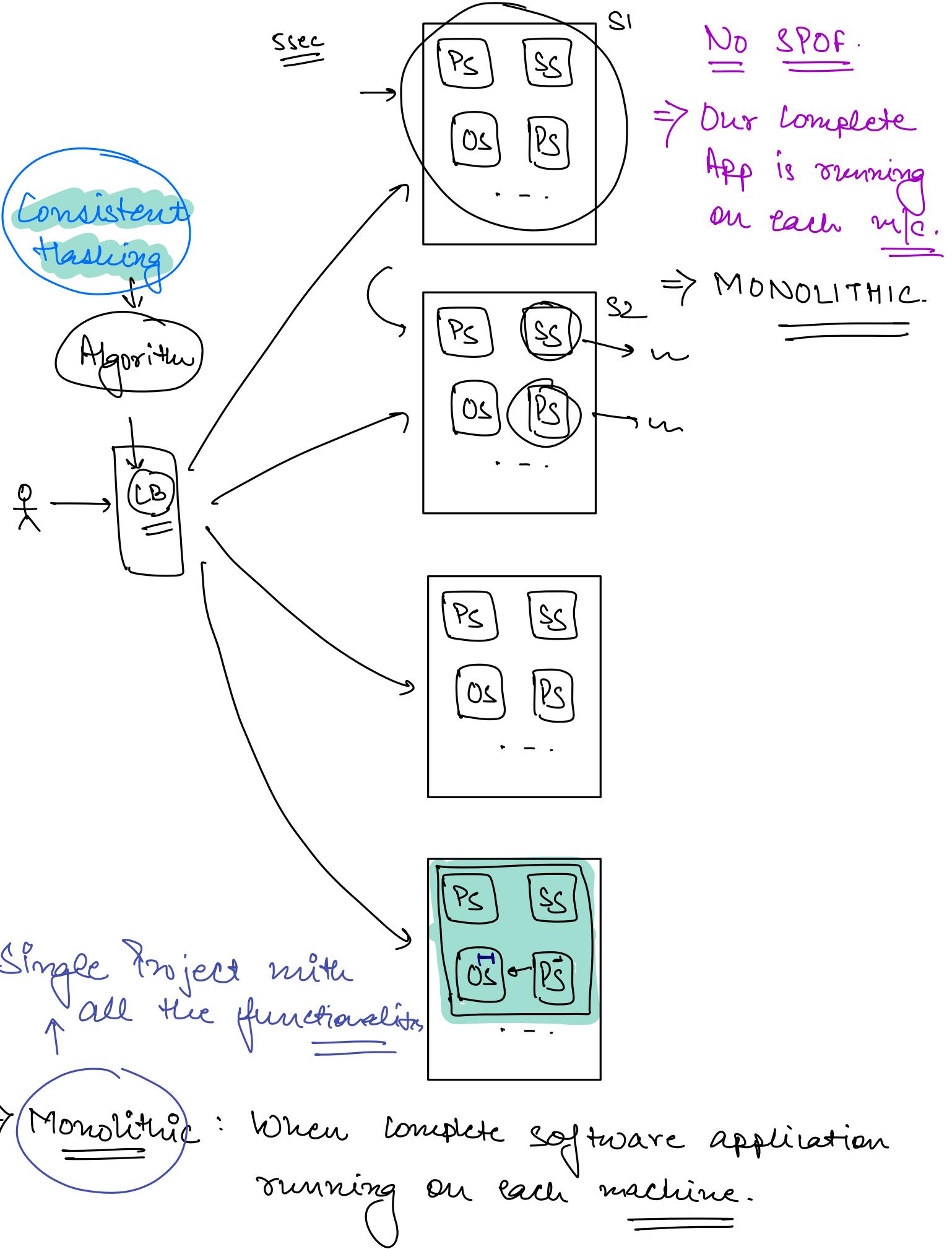
Horizontal Scaling

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

⇒ flipkart / Amazon

- ProductService
- UserService
- SearchService
- OrderService
- PaymentService
- Cart Service





Cons of Monolithic

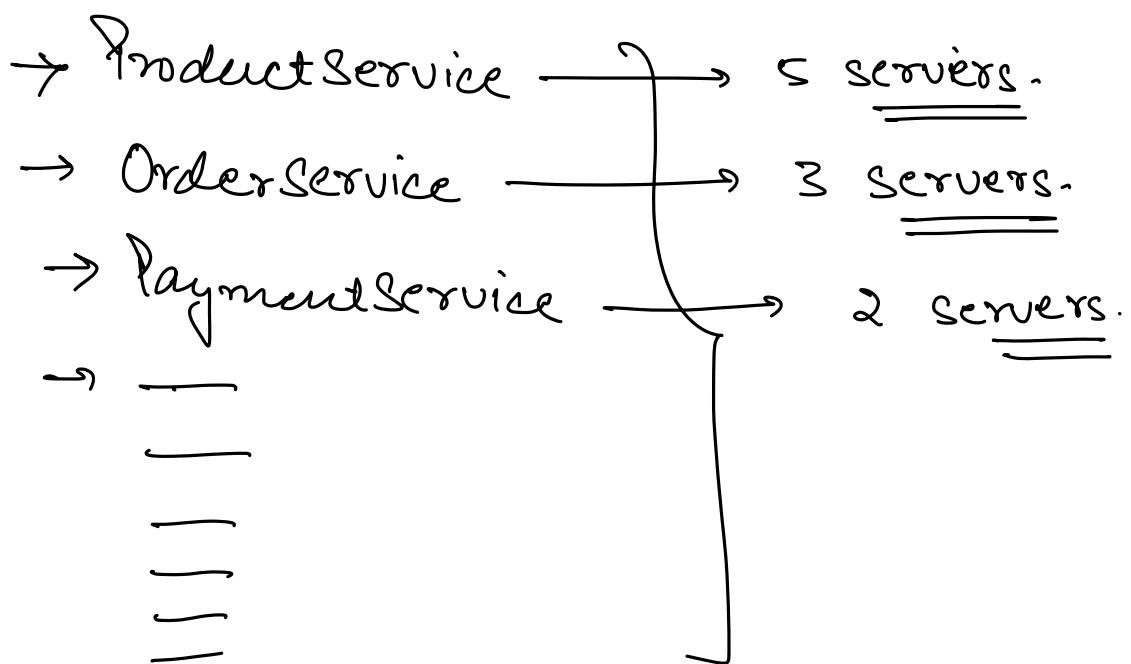
- 1) NO selective scaling → PaymentService $\Rightarrow 2 \text{ m/c}$
- 2) No tech stack flexibility → OrderService $\Rightarrow 3 \text{ m/c}$
- 3) High Deployment time → SearchService $\Rightarrow 10 \text{ m/c}$
- 4) A small bug can make the entire App down

Pros.

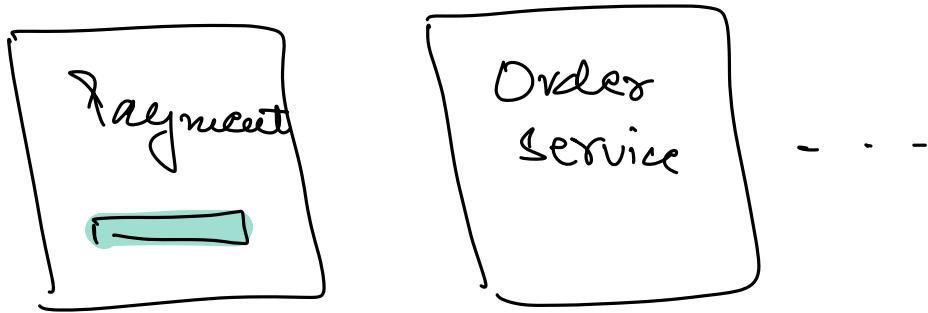
- 1) Single Deployment.
- 2) No n/w call, low latency.

⇒ MICROSERVICE.

Each service is an individual Project

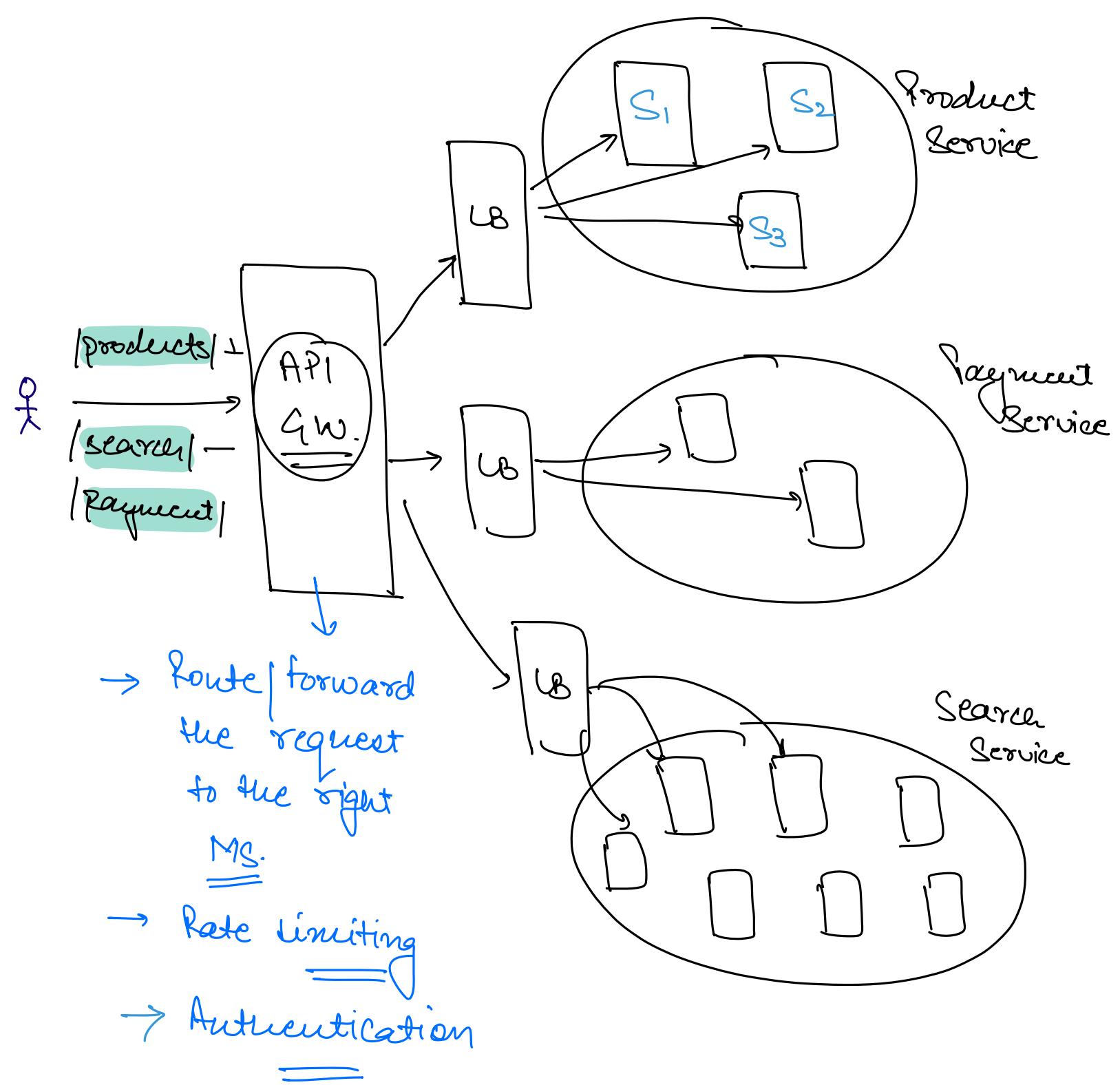


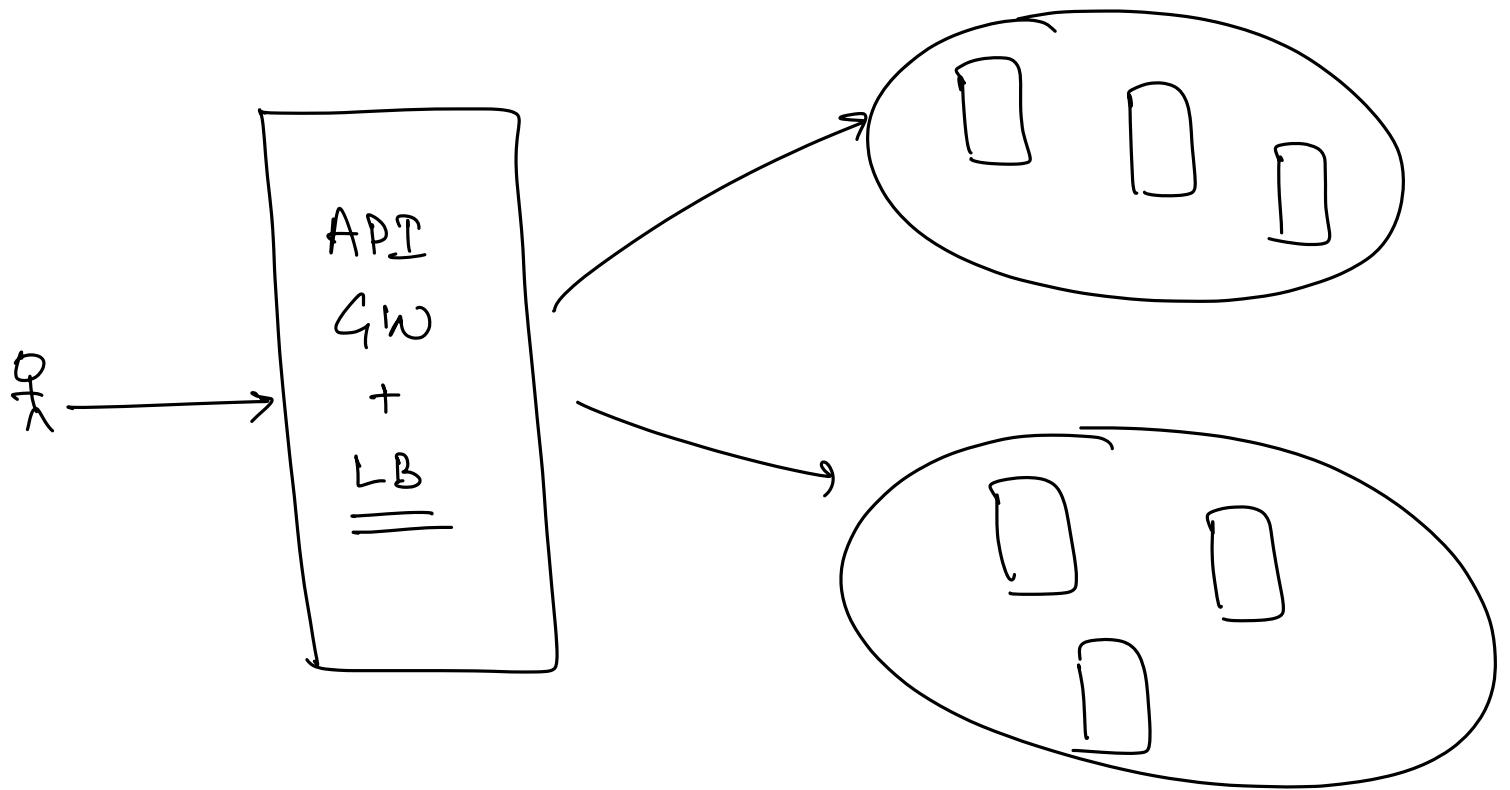
Microservice: When the Slow system is broken down into individual services & each service runs individually.



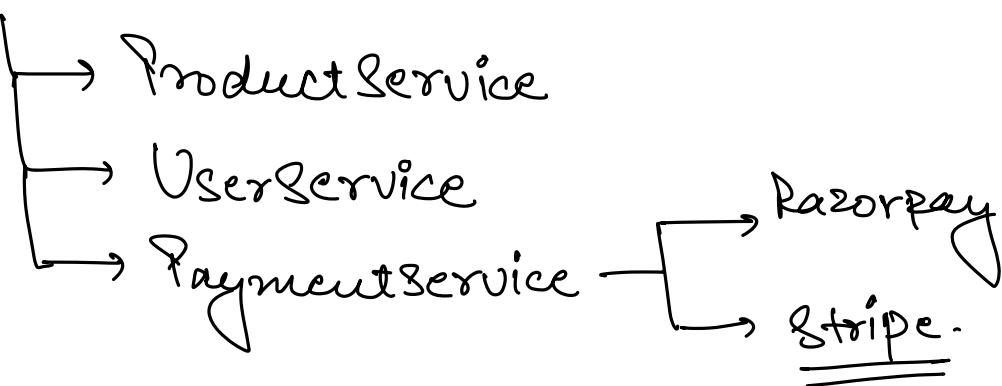
⇒ SRP: Single Responsibility Principle

Microservices.: Each service should have a single responsibility,



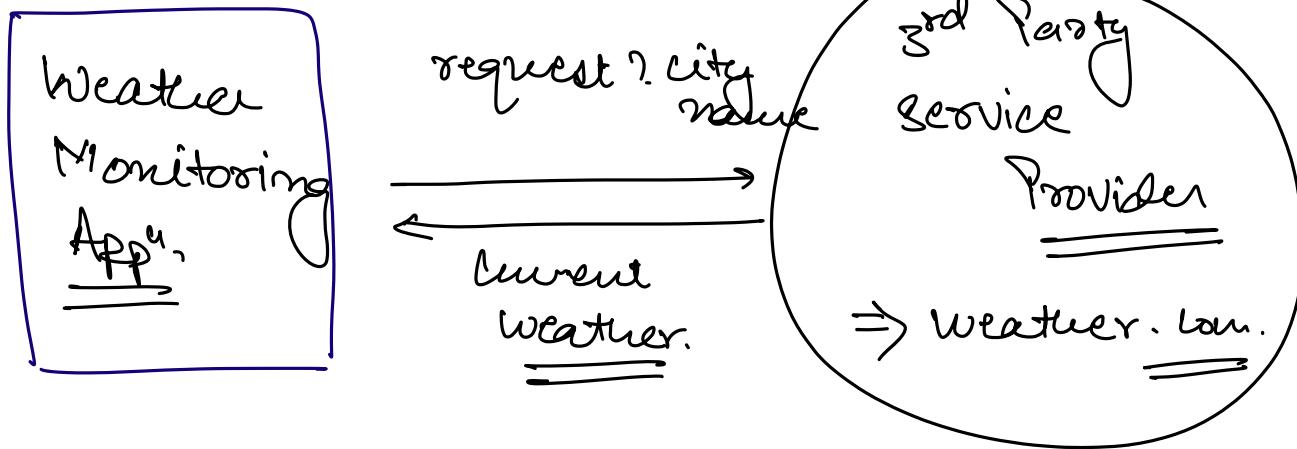
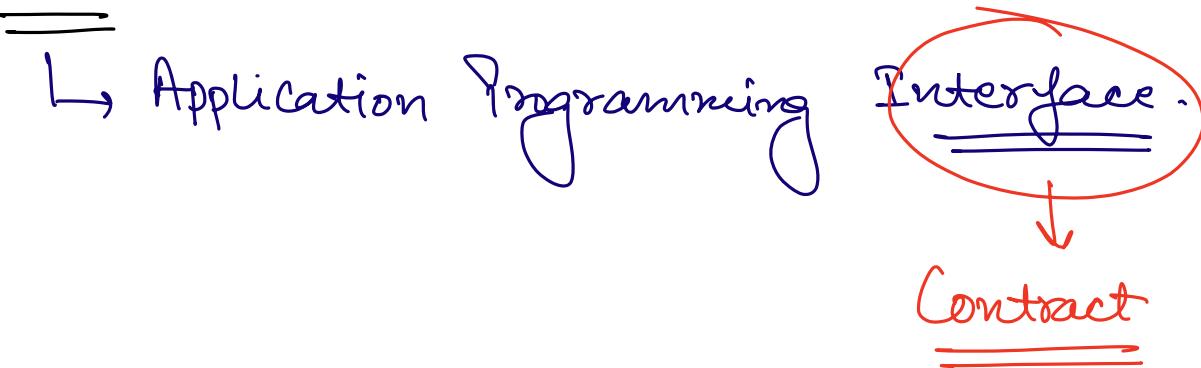


⇒ MICROSERVICES.



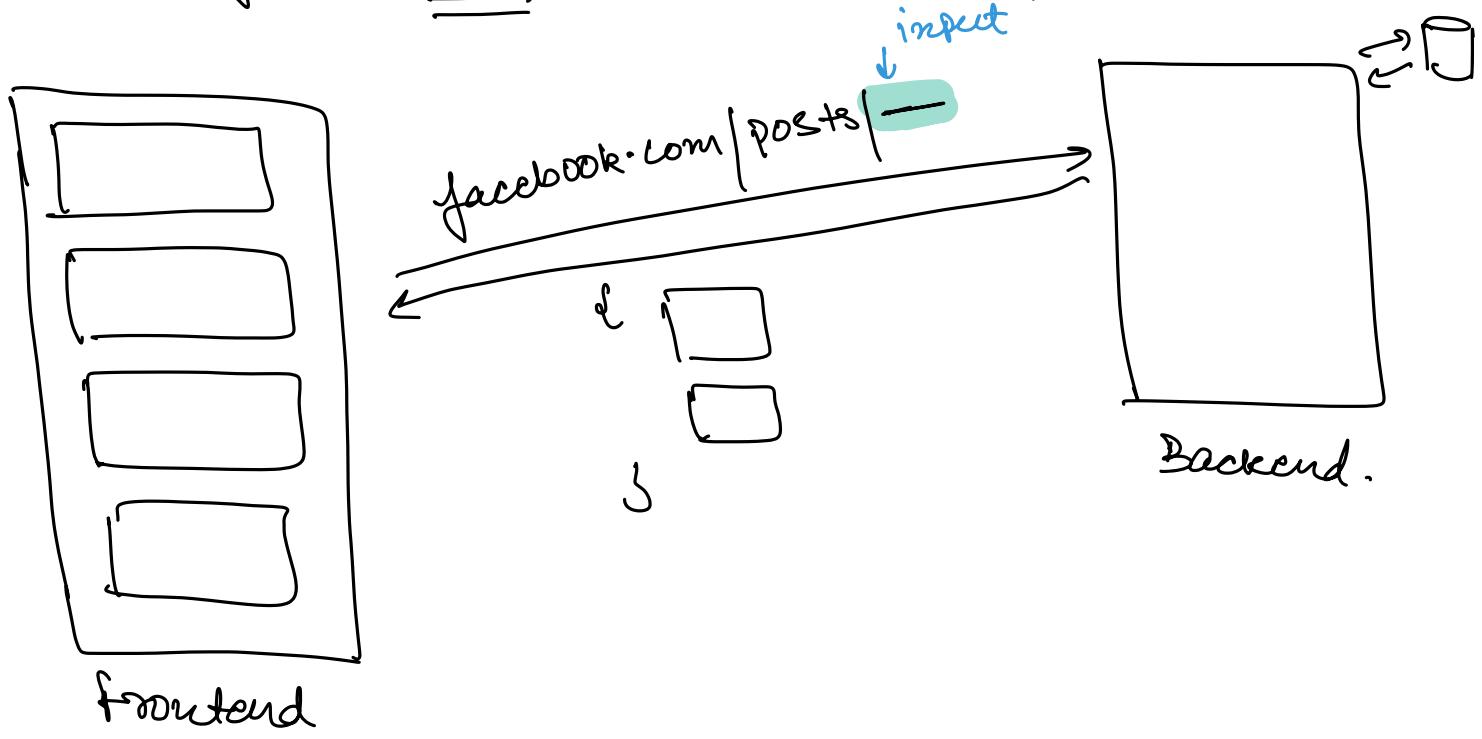
JAVA + SpringBoot.

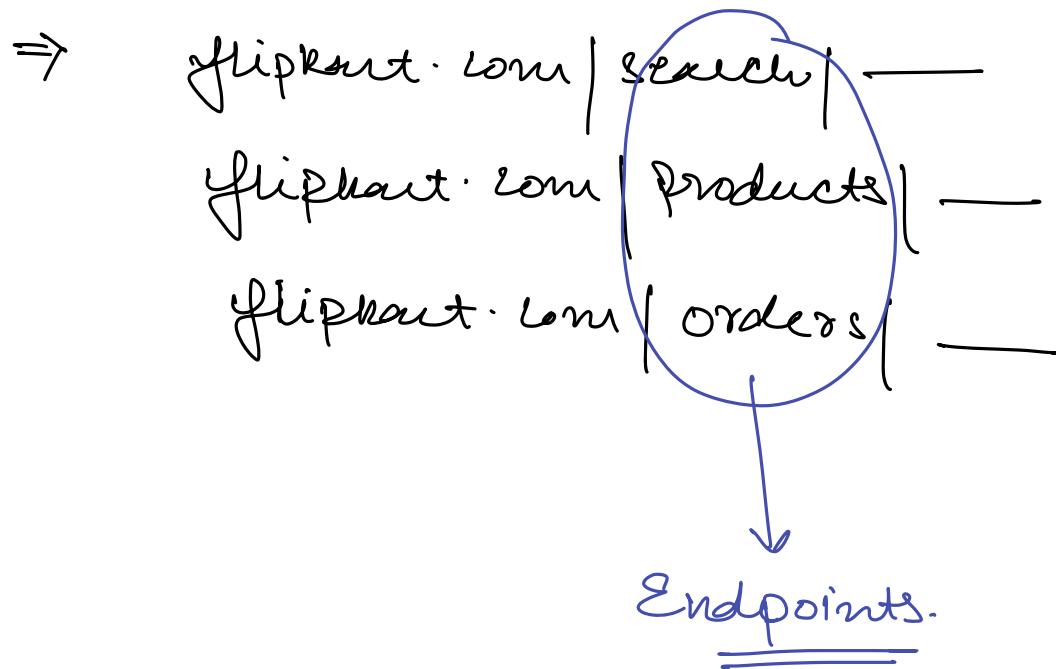
API's.



↑ \$ | Api call.

⇒ API: functionalities provided by companies to fetch data.



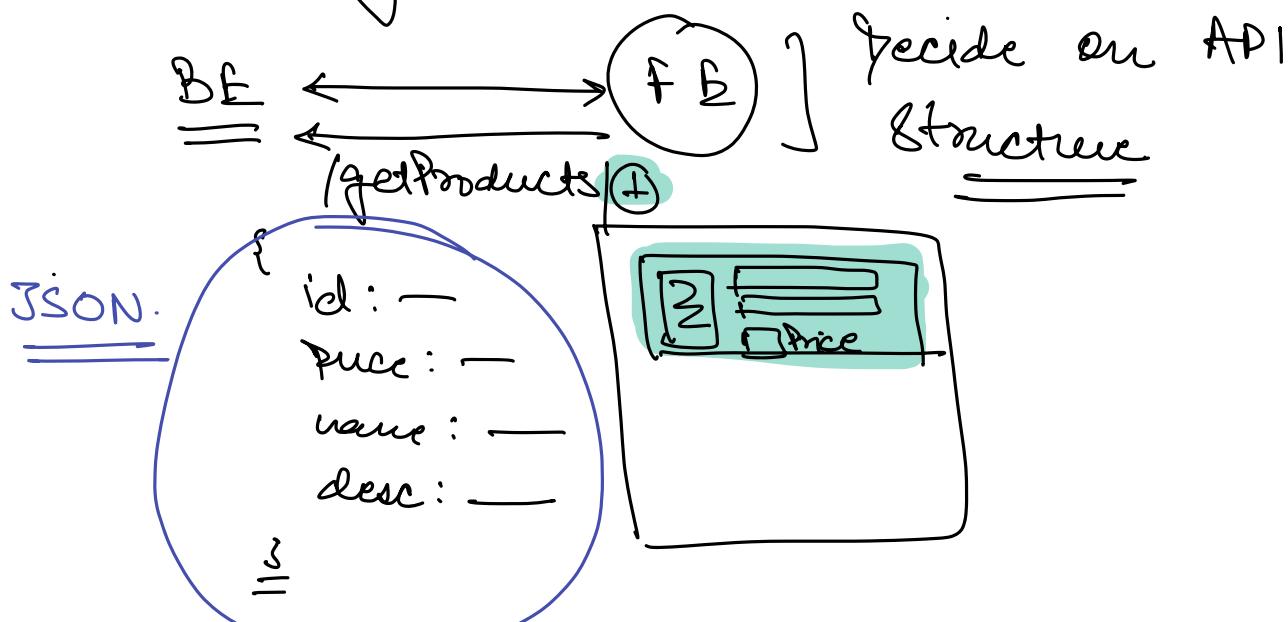


⇒ API : Set of Methods | URL's | endpoints | functionality
 provided by the application in order to
 interact with the data.

(get | update | create | delete) ⇒ CRUD

read

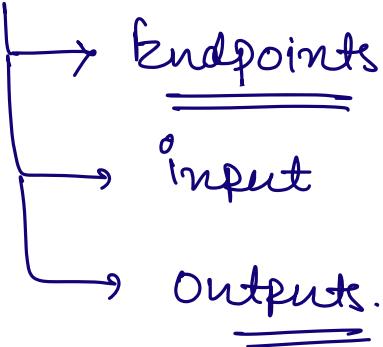
⇒ functionality



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



SOLID.

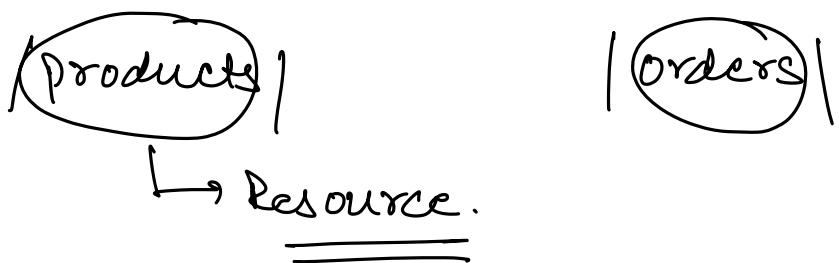
REST

R : Representational

S : State

T : Transfer.

⇒ Good coding conventions to build API's.



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

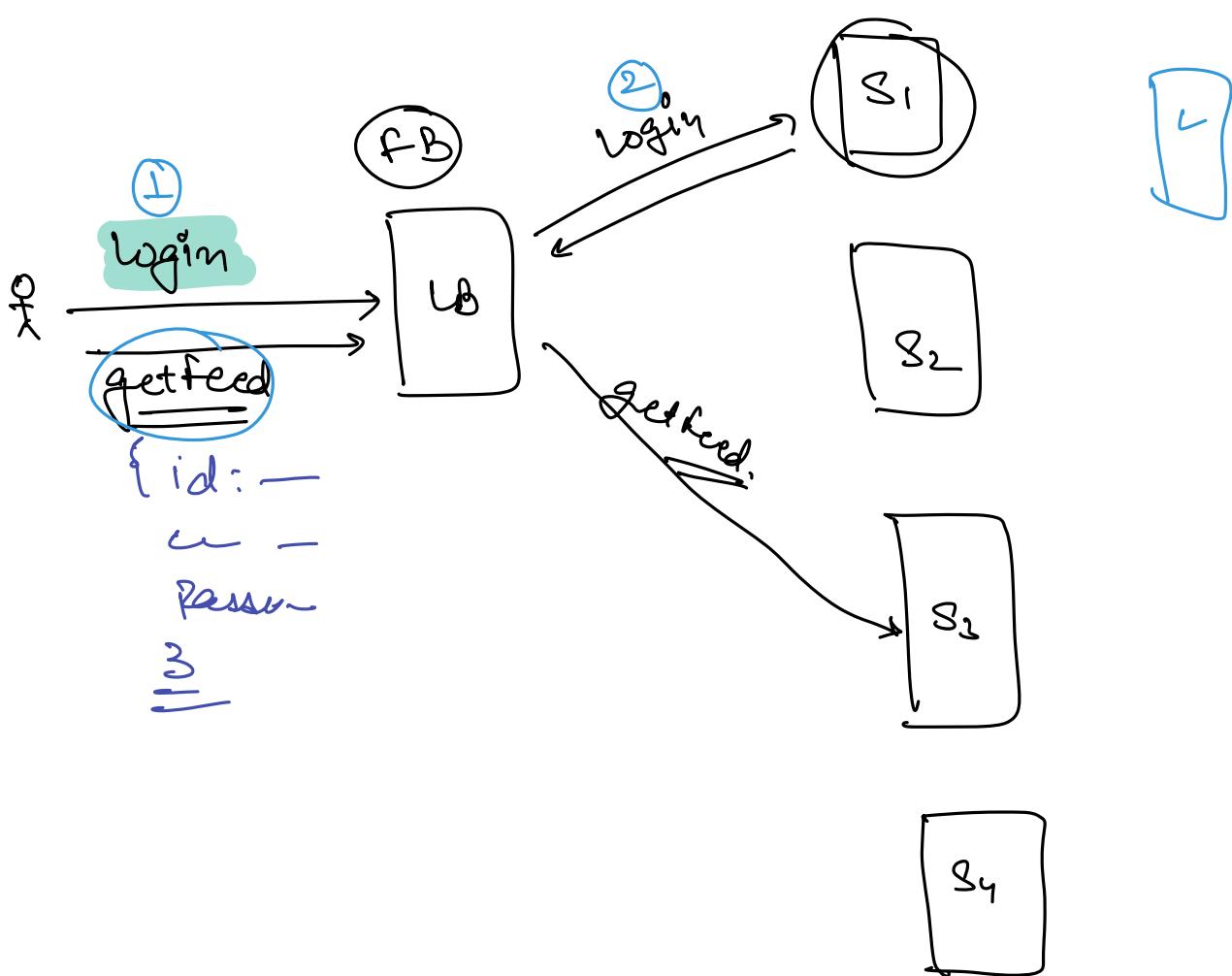
PUT

PATCH

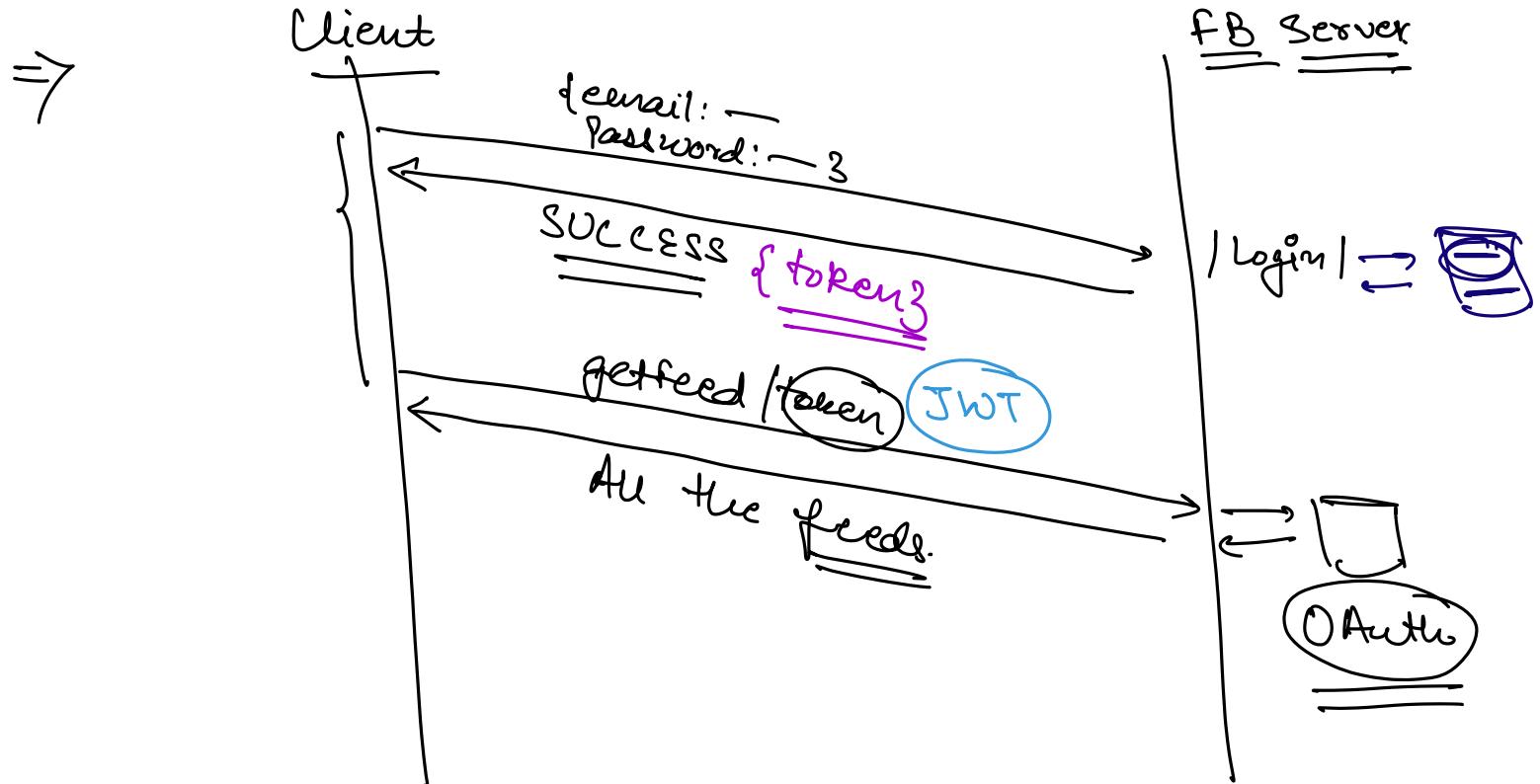
DELETE.

Interview -

③ REST API's should be stateless.



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



⇒ Stateless API's

⇒ Stateful

In the new request, we'll not send the required data, server ~~will~~ get the delta from my previous request.

Read-

⇒ FTP

File Transfer Protocol.

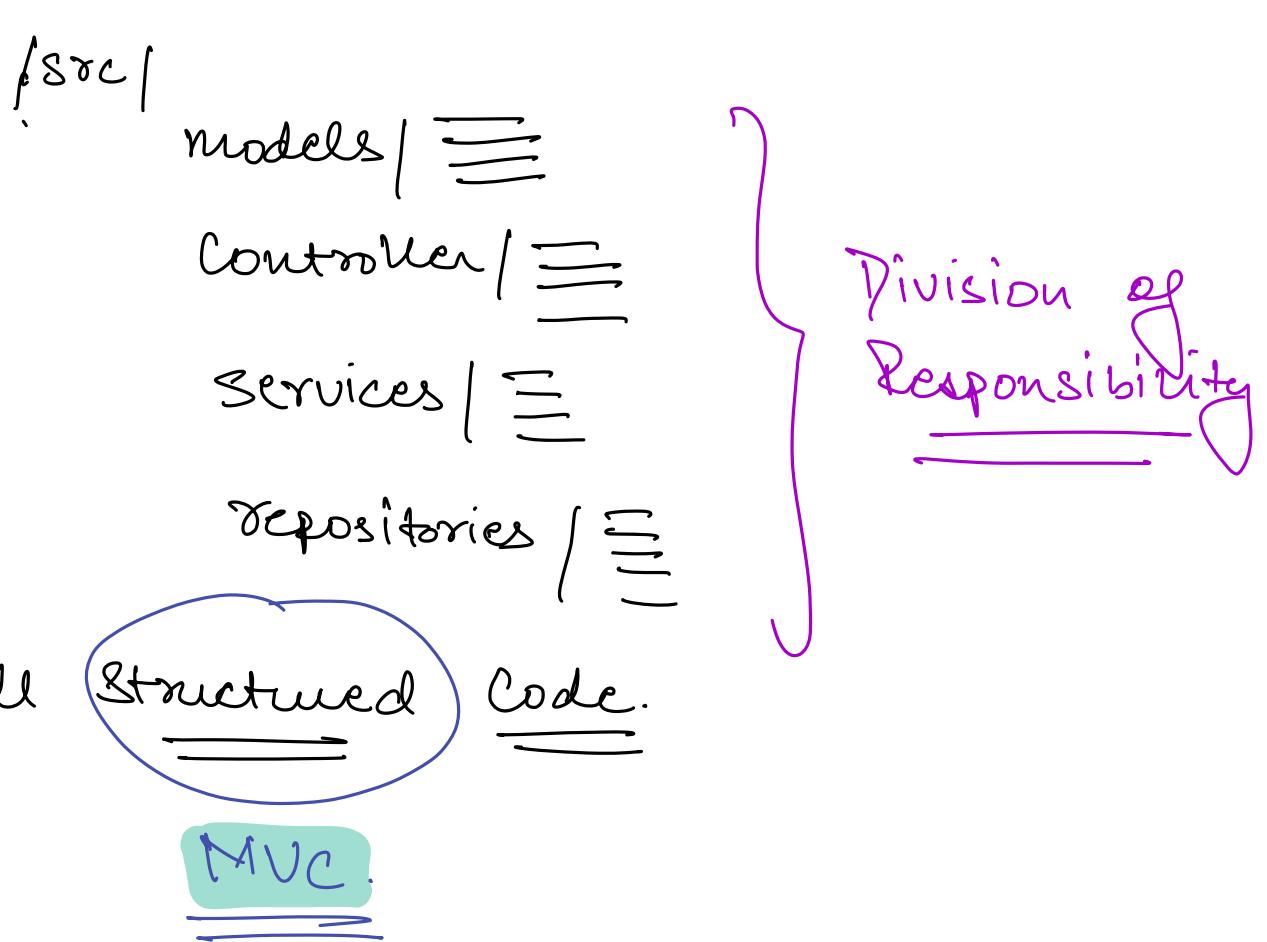
④ There's NO restriction on data type of

Response.

JSON ★ ⇒ easy to read.
XML

{
 id: —
 name: —
 age: —
 email: —

3

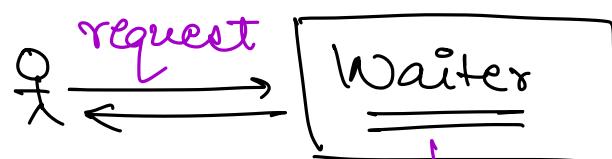


Maintainable \Rightarrow Current code should keep running.

Extensible \Rightarrow Adding new features.

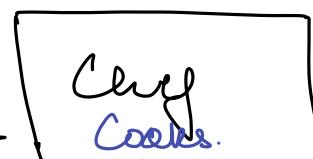
The diagram illustrates the MVC (Model-View-Controller) architecture. It features three main components arranged horizontally: "Model" (written vertically), "View" (enclosed in a large oval), and "Controller" (written vertically). Below the "View" component, there is a handwritten note "X" followed by a purple oval containing the letters "V".

Restaurant



Client facing
Controller.

Pass On

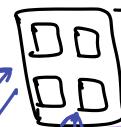


Hard Work

Service.

Business Logic.

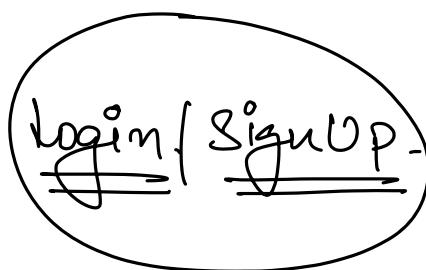
Refrigerator Store



DB-

Repository

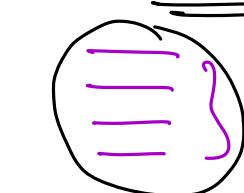
Model



User.

login → input
email
password.

UserController → UserService
true
false.



UserService → UserRepository



Select *
from —

⇒ Next Class.

- 1) Implement more API's in Product Service
- 2) UserService < ^{Login}
^{SignUp}
- 3) Loadbalancer + API GW.
- 4) Payment Service.