Agenda

1) Project Overview
2) Microservice vs Monolith
3) Intro to spring framework
4) Dependency Injection & IOC
5) Springboot
6) Build first API

PRD

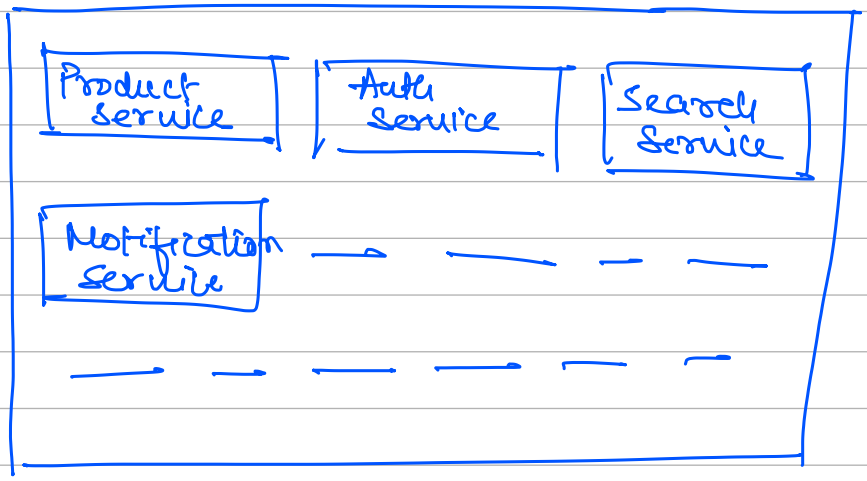
- Authentication Service
- Cart Service
- Product Service
- Order Service
- Logistics Service
- Notification Service
- Search Service
- User Service

Monolith

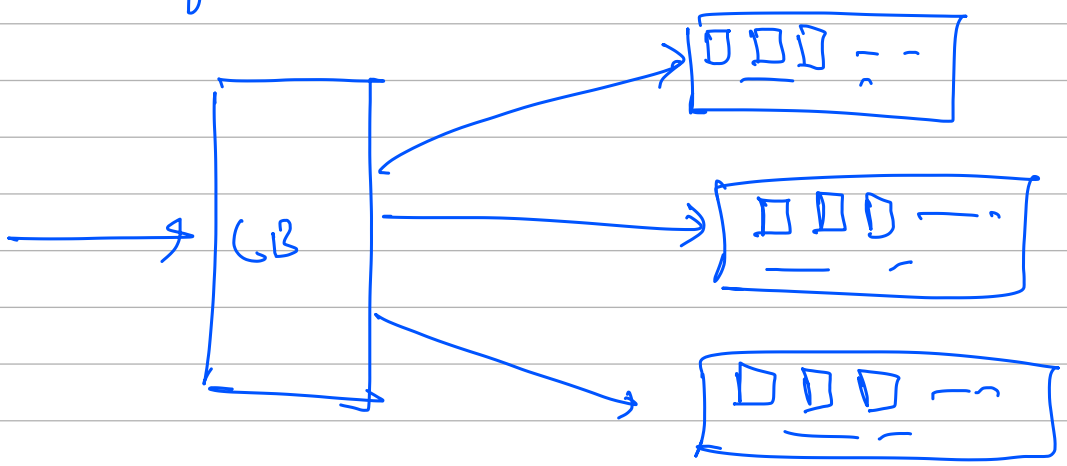
Microservice

Bansal Duo
↳ Flipkart

Amazon
4/200 Services



We will have to run multiple servers of complete codebase



Pros

1. Single deployment
2. Easy debugging
3. Low latency
4. Standardisation across codebase

Cons

1. High deployment time
2. No tech flexibility
3. No selective scaling
4. Small bugs have higher blast radius

Conf 1 Million → 45 mins

100 → 20 search Services
100 Notification Service

200k Amazon

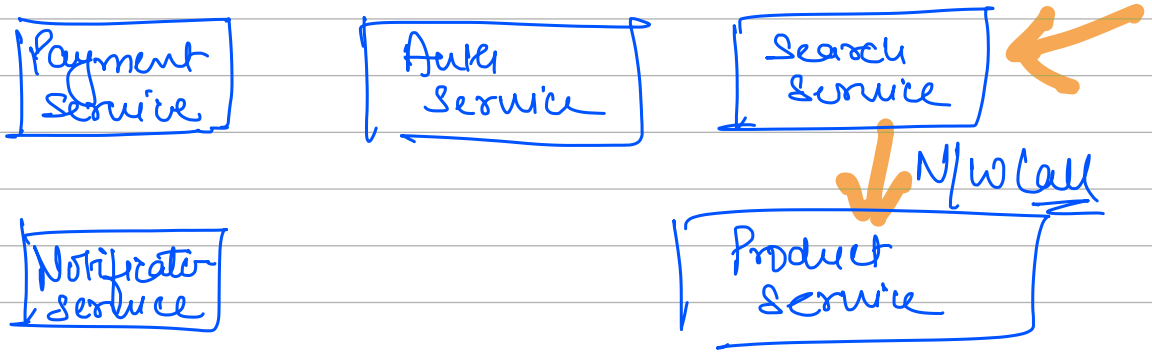
↳ Single Codebase

Microservice

↳ Service which catters to single responsibility

1 : 200000

1200 : 200000



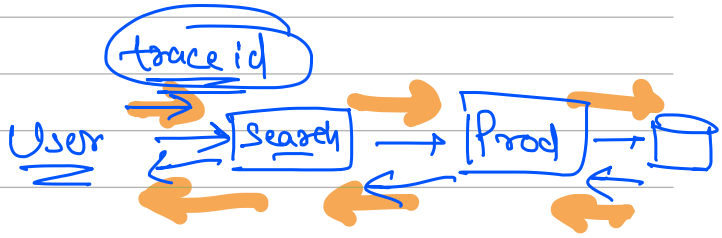
Pros

1) Selective scaling
2) Faster deployment
3) Tech stack flexibility

Cons

① Debugging tough

② N/W latency



Tier0 → Business Impact

Tier1 → Low Business Impact

Tier2 →

!

Pricing

↳ Tiers

Microservices

1) Product Service
2) Category Service
3) User Service (Auth)
4) Search Service
5) Payment Service
6) Notification Service

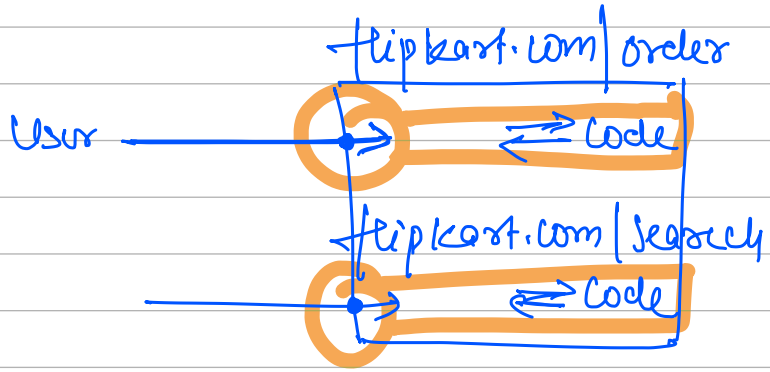
Framework

"Don't Reinvent the Wheel"

DRY

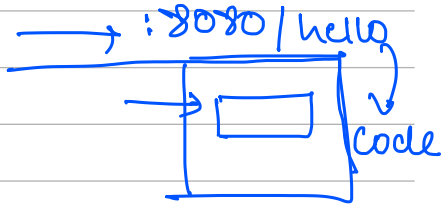
⇒ Provides ready to use implementation of common things that many software engineers would be using in their project

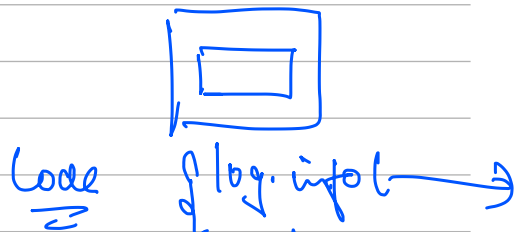
⇒ As a software engineers we would like to focus on the business logic



Framework provides functionality in a ready to use way

- → Creating APIs
- → DB comm
- → Configuration
- → logging





Spring framework

Set of projects that helps in creating Enterprise (Production) level java Applications easily

framework

```
Loggerfile {
    log.info() {
        openfile
        write(message)
    }
}
```

Spring

Core Project

Object Creation
Transaction
etc

+

Add ONs

Auto
Webserver
kafka
Redis

Spring Boot

SOLID

D = Dependency Inversion

Dependency Injection

Class A which is dependent on Class B

```
Class A {
    B b;
}
```

① Creating object of B inside the Class A

```
Class A {
    B b = new B();
}
```

② Set the value of object via Constructor

```
Class A {
    B b;
    A(B b) {
        this.b = b;
    }
}
```

```
B b = new B();
A a = new A(b);
```

Creating of dependencies outside of class & injecting them via const/setter is better way than creating object inside the class

1) Reuse of the object

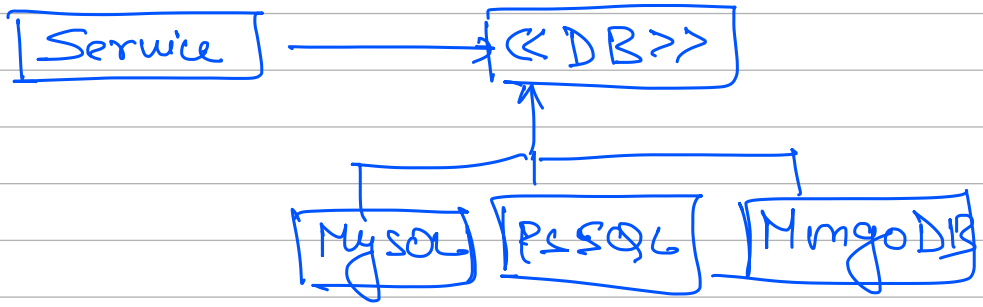
```
UserService {
    Db db;
        = new DB()
}
```

```
ProductService {
    Db db; = new DB()
}
```

CommonDB

```
Main {
    DB db = new DB();
    US us = new US(db);
    PS ps = new PS(db);
```

2) Loose Coupling

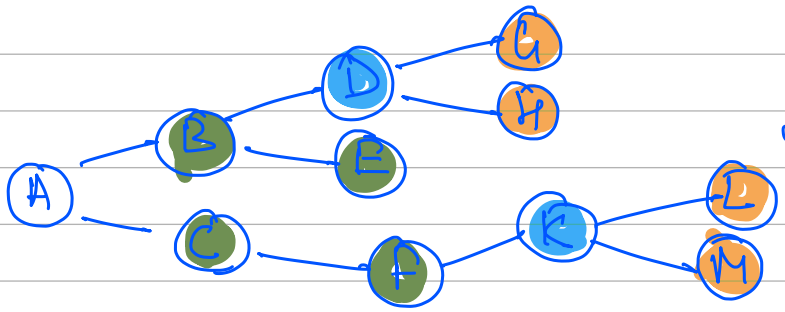


```
User Service {
    DB db = new MySQL();
}
```

PostgreSQL

With Dependency Injection

```
DB db = new PostgreSQL();
US us = new US(db);
PS ps = new PS(db);
```



Topological sorting

Spring : Dependency Injection framework

Confluence → 10000 Objects

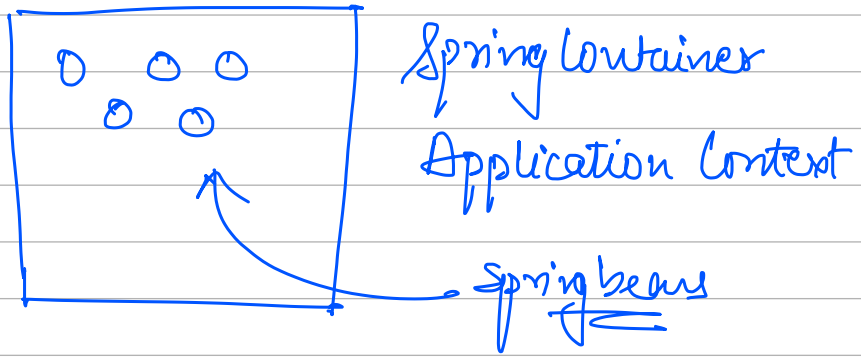
Inversion of Control

framework does the dependency injection on our behalf

Bean → Object

→ Objects that are managed by spring

↳ An object that spring will create manage & use for injecting wherever required

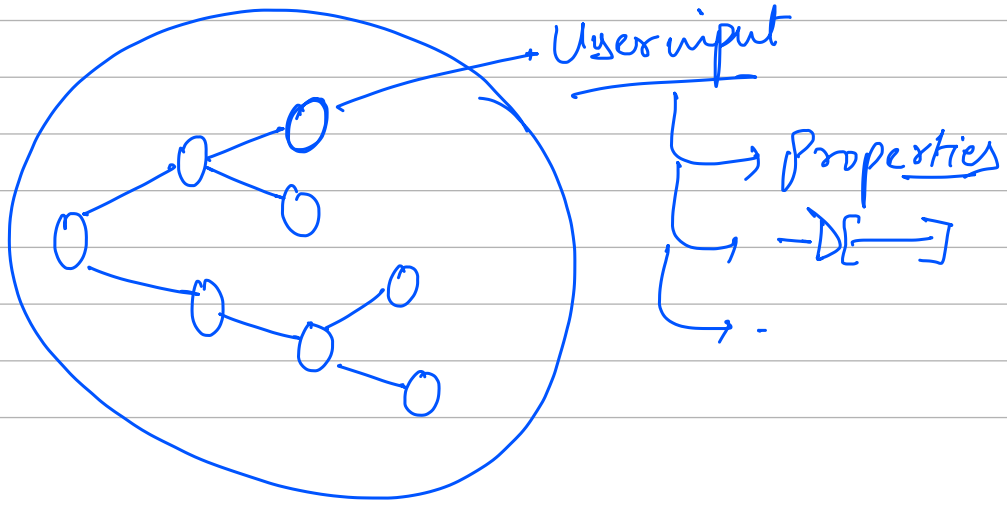


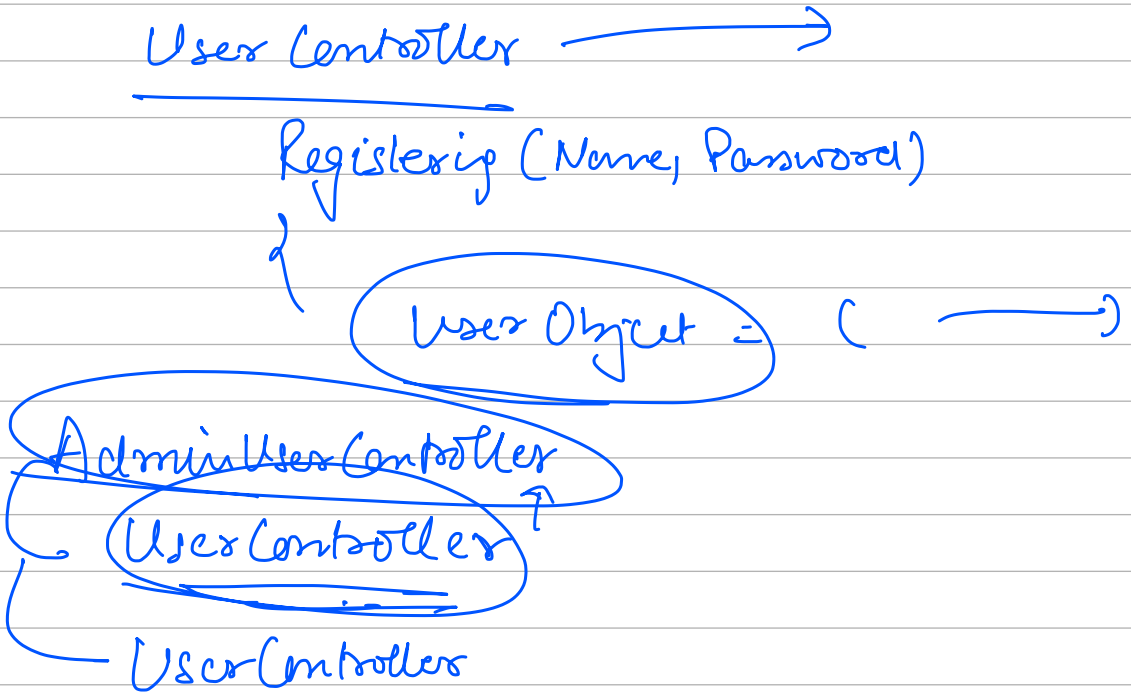
1) Start the Application]
2) Create the Beans
3) Stored in Application Context

Spring fw

↳ dependency Injection

Spring Boot = spring fw + Addons




User Controller
Registering (Name, Password)
{
User Object = (
AdminUserController
UserController
UserController