

Internship @ Dohatec

Full stack

Software Development

using

- Angular
- Spring Boot
- Spring Security

SPA

↳ Single Page Application

SPA → load all the data/code and then never refresh again

→ काय वाह refresh ना होयाए , application अल्हातर fast रहे

Info:

- need to install Node for angular
- MVC Architecture
- Angular.js and Angular are different

Angular Component Life - Cycle Hook ;

Phase

1. Creation
2. Change Detection
- ~~3. Detection~~
3. Rendering
4. Destruction

that will automatically call when our pages are loaded

Creation → constructor → runs when Angular instantiates the component.

Change → ngOnInit → Runs once, after Angular has initialized all the components' inputs.

Destruction :

→ ngOnDestroy → Runs once before the component is destroyed.

Rendering

→ after Render → runs every time
all components have
been rendered to
the DOM.

→ after Next Render → runs once
the next time that ~~is~~
all components have
been rendered to
the DOM.

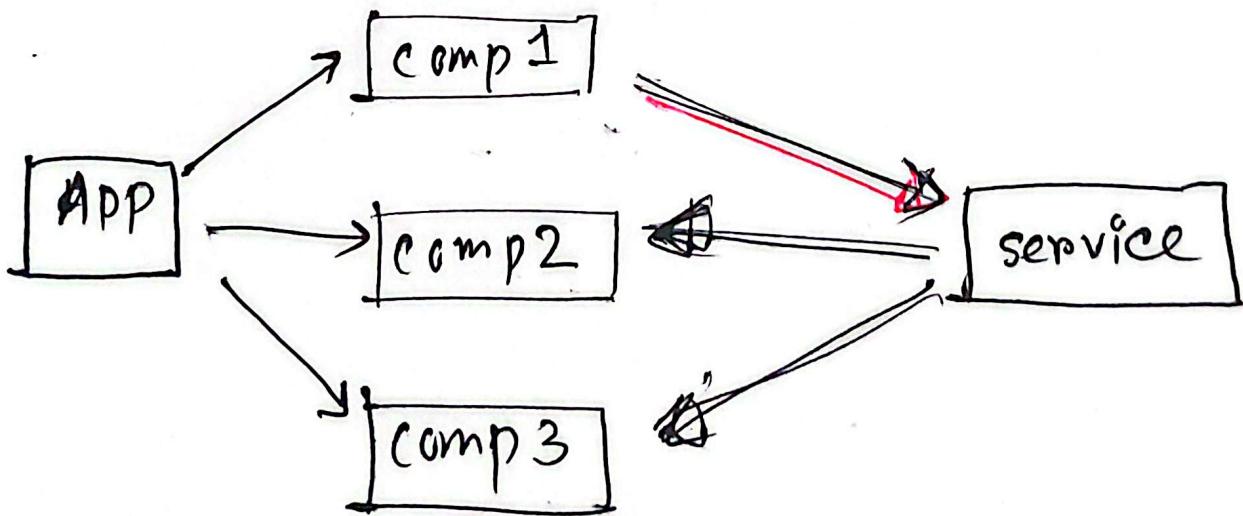
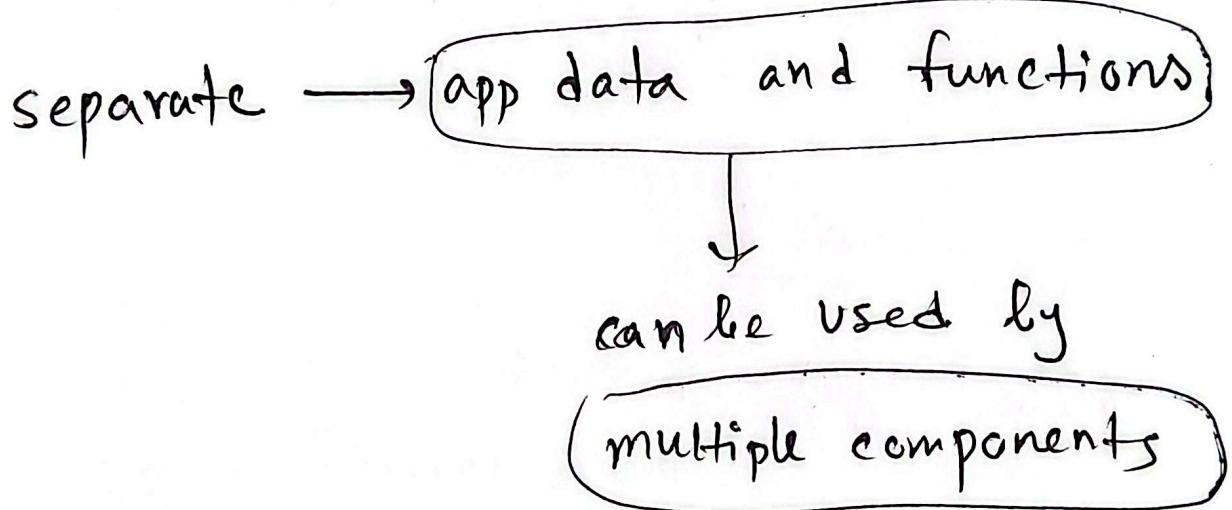
Change Detection

→ ngOnChanges → runs everytime
the components inputs
have changed .

Services in Angular

What are services?

Angular services provide a way for you to separate Angular app data and functions that can be used by multiple components in your app.



creating a service

ng g s service-name

ng g s services/service-name

Ex:

ng g s services/product

first
product.ts

```
import { Injectable } from '@angular/core';
```

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class Product{
```

```
  constructor(){
```

```
    console.log("product-service called");
```

```
}
```

```
}
```

```
getProductData(){  
  return [  
    {name: 'Samsung',  
     category: 'phone',  
     price: 110000},  
    {name: 'Asus',  
     category: 'Laptop',  
     price: 740000},  
  ]  
}
```

If we want to keep some data then we should not
keep this data in the constructor only.
Because in constructor we can't return anything.

then

2nd

to call the services in the App Component

APP.ts

export class App {

constructor(private product: Product) {}

productData: {

name: string;

category: string;

price: number;

}[] | undefined

~~const~~

getData() {

this.productData =

this.product.

getProductData();

function in the
service

serviceName

~~then~~
3rd

app.html

function written in
app.ts

<h1> Services in Angular </h1>

<button (click) = "getData()">

Load Product Data

</button>

@for (item of productData; track item){

<h2>{{item.name}}, {{item.category}},
{{item.price}} </h2>

}

call API with the Services in Angular:

API :

API : Application Programming Interface

- We can not connect database with Angular or JS
- So we make API in any backend programming Language
- API integrate with angular or react or any
- Same API can connect in any technology.

Example:

products.ts

```
export class Products {  
  constructor(private http: HttpClient) {}  
}
```

productList() {

```
  const url = "https://dummyjson.com/products";
```

```
  return this.http.get(url);
```

app.ts

```
export class App {
```

```
  constructor(private product: Products) {}
```

life-cycle hook

```
  ngOnInit() {
```

```
    this.product.productList().subscribe()
```

```
      (data) => {
```

```
    }
```

Service name

this service
is inject
through the
constructor

function/method
in the
service

without it,
the HTTP request
will never run

subscribe():

In Angular, HTTP calls return an **observable**.

→ a container that can give you data now, later or many times in the future

To actually get the data,
you must call **.subscribe()**.

Without **.subscribe**, the HTTP request
will never run.

subscribe() → tells Angular to execute
the API call and give
you the result.

Install JSON server and Make API :-

JSON Server :-

JSON server is basically a tool that help us to make fake APIs

json server typicode

npm install json-server

1. Install JSON server
2. Make db.json file
3. Make Users API
4. Test API with postman / thunder client

GET → get the data from database

POST → store the data in database

PUT → update entire resource on server

PATCH → update partial resource on server

DELETE → delete the data from db

Define Data type for API result :-

Interface:-

Interfaces are a feature of TypeScript that allows us to define the structure or datatype.

Post API in Angular :-

Route Lazy Loading :-

With the help of lazy loading, we can only load the required code in our Angular application whenever we are using that Angular application.

In `app.routes.ts` :-

```
export const routes: Routes = [
  { path: 'admin', loadComponent: () =>
    import('./admin/admin').
    then(c => c.Admin),
  }
];
```

The component we want to be as Lazy Path of the component

Component Name

Component

Component

Route Guards:-

In Angular, Route Guards are special features that let you control access to specific routes in your application.

They decide whether a user is allowed to navigate to a route, leave a route, load a module or fetch route data.

Types of Route Guards in Angular:

i) CanActivate → checks if an user can navigate to a route

ii) CanDeactivate → checks if an user can leave a route
(useful for unsaved changes)

iii) CanActivateChild → checks if a user can access child routes

④ canMatch → Allows or blocks major navigation before route is matched

⑤ canLoad → Prevents lazy-loaded modules from loading

Resolve → This performs route data retrieval before any activation of the route.

Auth Guards

An Auth Guard in Angular is a Route Guard used specifically to protect routes from unauthorised access. **(canActivate)** guard

Spring Boot

Java Framework

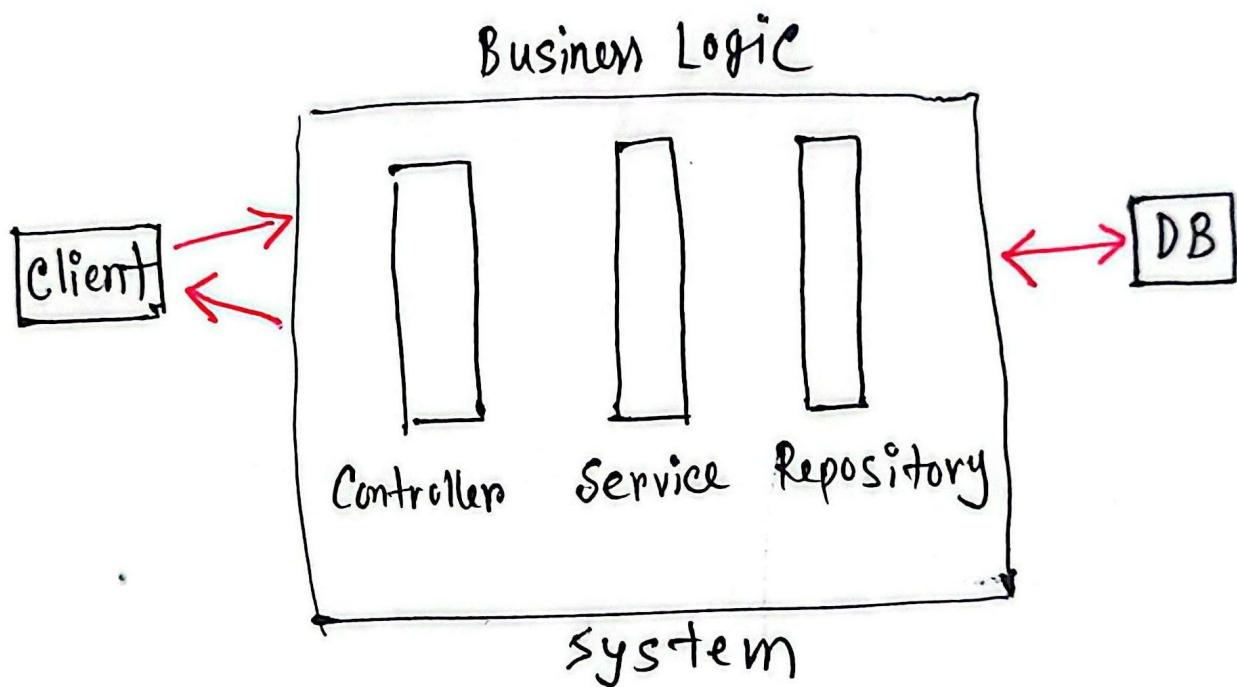
Spring Boot

IoC:

→ Inversion of Control

IoC means:

- You do not create objects manually
- The framework (Spring) creates and manages objects for you.



DI (Dependency Injection) :-

DI means:

- You do not create objects manually
- Spring creates and gives (injects) the required objects to your class automatically.

Your class simply "receives" its required dependencies instead of "creating" them.

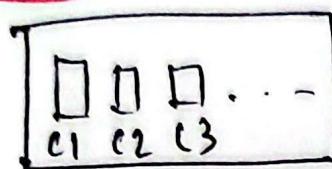
Dependency Injection is the actual implementation of IoC.

Spring Boot :-

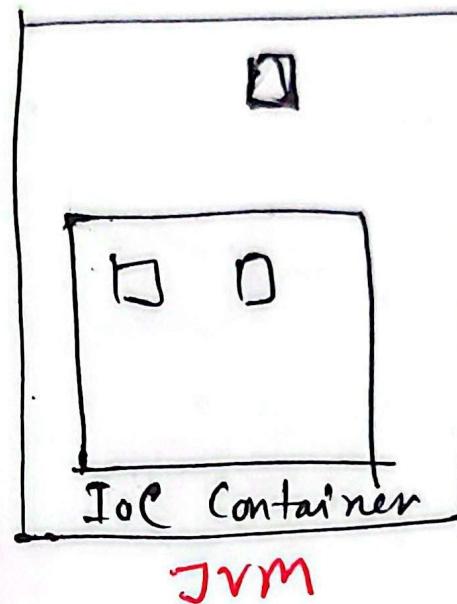
Dependency Injection Using Spring Boot

new class is outside the container:

class

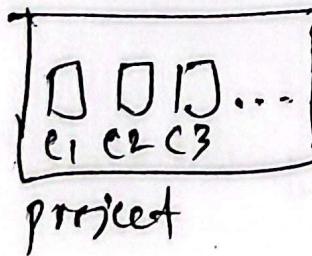


project

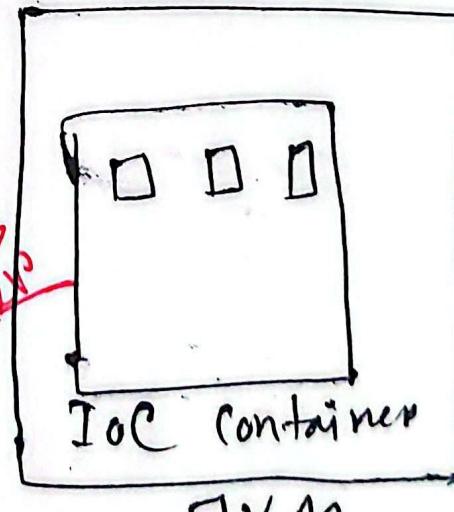


JVM

Let's assume the newly created class is already in the IOC container



project



JVM

tree of this
IOC container

Application
context

Examples

Dev.java

```
package com.tarrik.App2;
import ---.Component;
```

~~@Component~~

```
public class Dev{
```

```
    sout("Working on a  
spring project");
```

↳

main → java → com → App2

(App2Application.java)

```
package ...;
import ...;
```

@SpringBootApplication

public class

SpringApp2Application{

public static void

main(string[] args){

ApplicationContext

context = SpringApplication

SpringApplication.run();

~~KK~~

Dev obj =

context.getBean(Dev.class);

obj

obj.build();

}

}

DI

↳ Injects the
Dependency.

Autowire :-

`@Autowired` } is a Spring annotation that

tells Spring :

→ please create the object for me
and inject it here.

- Field Injection (`@Autowired`)
- Constructor Injection (Default)
- Setter Injection (`@Autowired`)

Laptop.java

```
package ...;
```

```
import ...;
```

```
@Component
```

```
public class Laptop {
```

```
    public void compile() {
```

```
        sout ("Compiling with 404 bugs");
```

```
}
```

```
}
```

Laptop.java B Dev.java connect 22

by type

Laptop.java
public class Laptop {

connect

Dev.java

private Laptop laptop;

}

Constructor Injection

Dev.java

```
import ... ;
```

@Component

```
public class Dev {
```

```
    private Laptop laptop;
```

```
    public Dev(Laptop laptop){  
        this.laptop = laptop;
```

```
}
```

```
    public void build(){  
        laptop.compile();  
        System.out.println("working on  
        awesome project");  
    }
```

Setter injection

```
private Laptop laptop;
```

@Autowired

```
public void setLaptop(Laptop laptop){  
    this.laptop = laptop;  
}
```

Field Injection

Dev.java

```
import ... Autowired;  
import ... Component;
```

@Component

```
public class Dev {
```

@Autowired

```
private Laptop laptop;
```

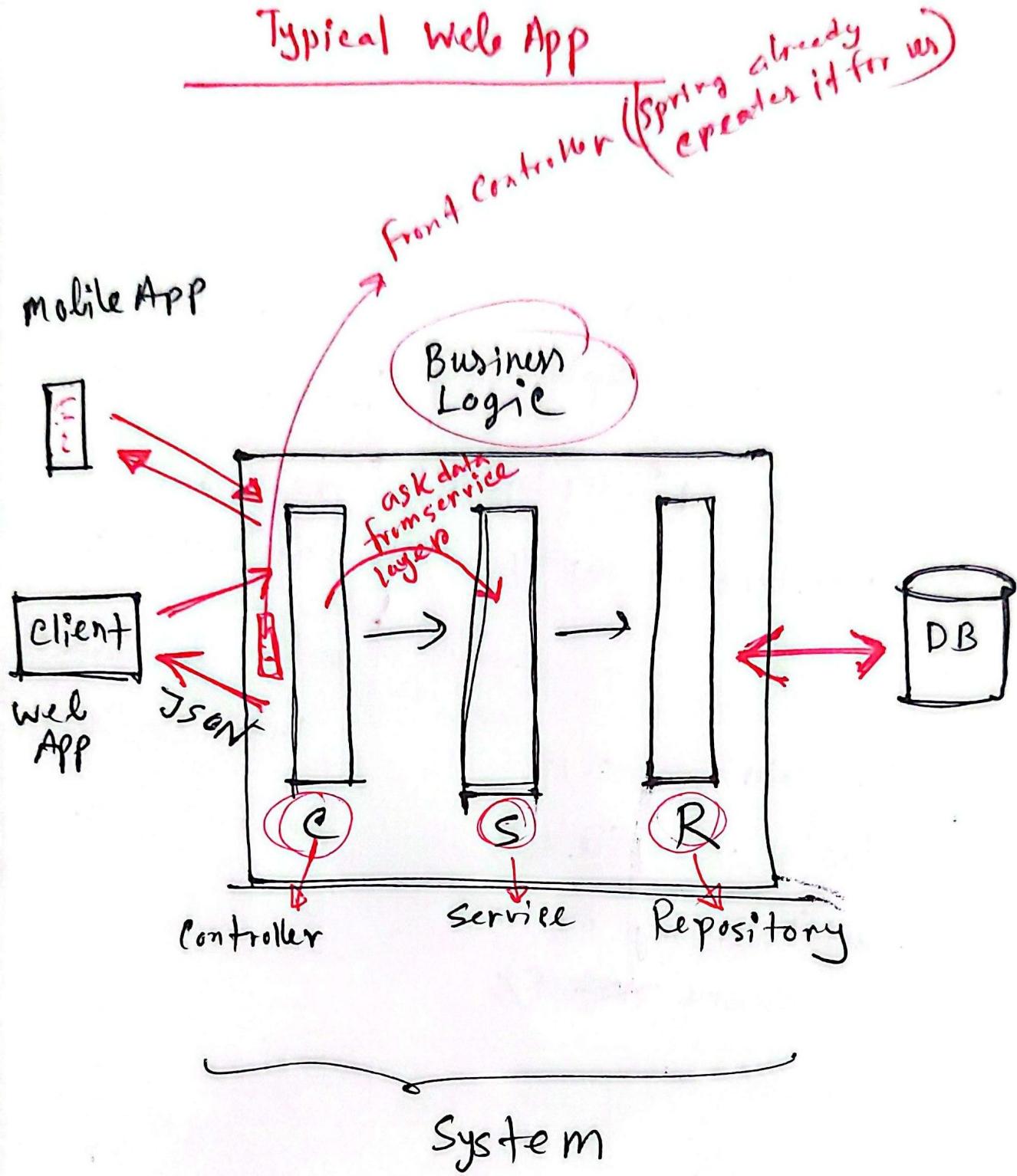
```
public void build(){
```

```
    laptop.compile();
```

```
    System.out.println(" ");
```

```
}
```

Typical Web App



Controller → just for req and res
not any business logic

Service → all the business logic

Model → data of the project

Repository → connect with database

What is the particular thing who can handle the request → controller

27/12/2025

Controller:

A controller is a Java class that:

→ Receives requests from the client (web browser/Mobile App)

→ Process the request

→ Sends a response back (JSON, text, data... etc)

Controller → API entry point

its like a receptionist in an office:

who:

→ Accepts requests

→ Redirects to the correct Department

→ Sends the result back.

Tombok dependency

MVC

(Model View Controller)

Controller → req, res

View → pages (UI)

Model → Data

Service → All the business Logic

~~REST~~

→ HTTP methods

→ GET → fetch data

~~CRUD~~ → POST → add new data

→ PUT → update the data

→ DELETE → delete the data

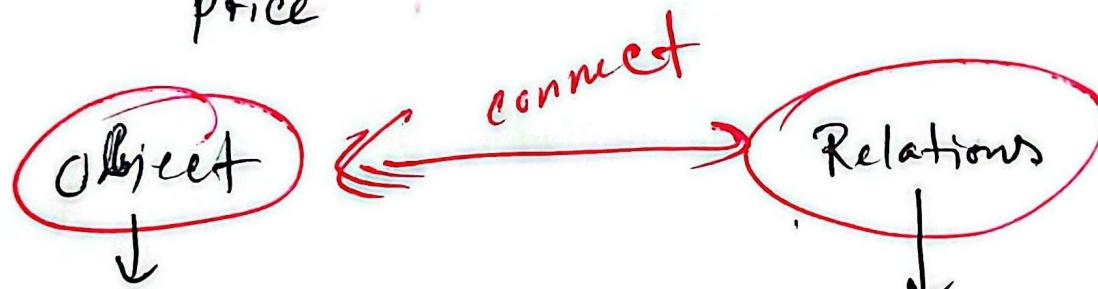
Spring Data JPA :

ORM

↳ object Relational Mapping

object → Data + behavior

prodID,
prodName,
price



Data

prodID, prodName, Price

[
 {
 },
 {
 },
 {
 },
 {
 },
]



prodID	prodName	Price

30/12/2025

ORM tools

Hibernate → full fledged

EclipseLink

MyBatis

Object

Relations

className

Table Name

variables
(each)

column name
(each)

each object

each row

JPA

Java Persistence API

With the help of Spring JPA we can

store the data which we are doing

in the service layer in the DB.

package → Repository

Repository → Interface

An Interface in Java
is a collection of
abstract methods and
constants.

methods without
a body

→ Any class that
implements an interface
must provide the
method definitions.

Example :

Interface

```
interface Animal {  
    void sound();  
    void eat();  
}
```

class implementing the
interface

```
class Dog implements Animal {  
    public void sound() {  
        sout("Dog barks");  
    }  
    public void eat() {  
        sout("Dog eats bones");  
    }  
}
```

using the class

```
public class Main {
    public static void main(String[] args) {
        Animal a = new Dog();
        a.sound();
        a.eat();
    }
}
```

Using JPA Repository Step 1

Repository Package → Interface → Product Repo .java

imports; the class we will work with primary key type

```
@Repository
public interface ProductRepo extends
    JpaRepository<Product, Integer> {
```

}

Step 2 → ProductService.java

imports . . - . ;

@Service

public class ProductService {

@Autowired

ProductRepo repo;

Project

(R)

int id

sn

Product:

id → primary
key

name → string

description → "

brand → "

price → int

category → string

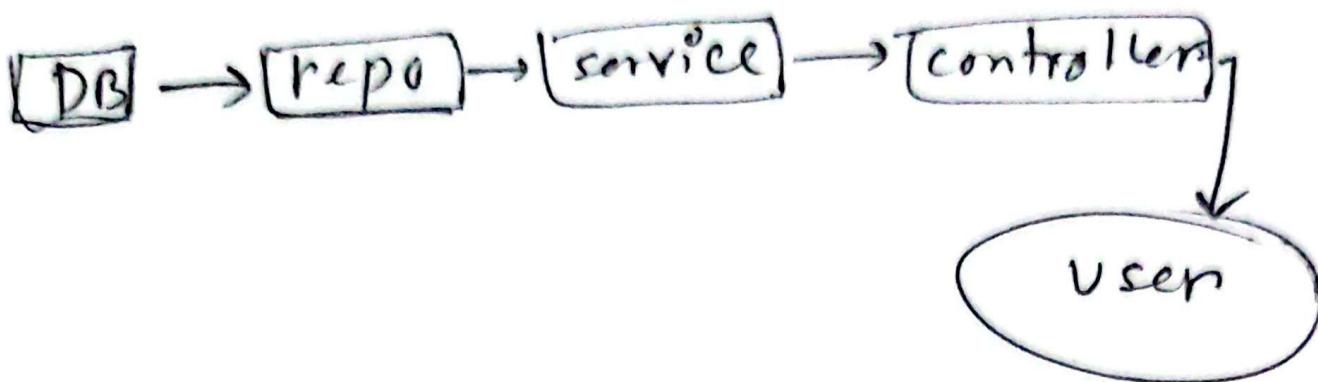
release Date → Date

available → boolean

quantity → Int

? Img →

when data fetching what happens



CORS Error is Spring Boot of

→ Cross-Origin Resource Sharing

It is a browser security system that prevents a website from calling APIs on another domain unless allowed.

→ For the security reasons.

it is not allowed to send a request with some another address.

Spring security

1/12/2025

Spring security can support the following right out of the box :-

- ①. Username/ Password authentication
(from login)
- ②. JWT (JSON Web Token)
- ③. OAuth2 and OpenID Connect
(OIDC) integration 3rd-party auth
- ④. Social login
(Google, GitHub, Facebook etc)
- ⑤. LDAP authentication
- ⑥. Remember - me authentication
- ⑦. Role-based access control (RBAC)
- ⑧. Fine-grained permission
expressions using SpEL

9. Concurrent session control
(limit login sessions)
10. session timeout and invalidation
11. Built in CSRF token generation
and validation
12. Cross-Origin Resource sharing (CORS)
header
13. X-frame-options for
clickjacking protection
14. custom authentication providers
15. custom filters and filter chain.

1) Add **Spring Security** dependency



Dependency add करते ही सेटिंग्स

project में spring security एवं

spring default behavior add

रखें या नहीं → EX:

i) route → protected Route
Authenticated Route

ii) to access
a route → login first

iii) to login → generated
spring security
password
use करते ही

iv) User → by default
User नाम लेने
users.create करने
को मिलता है
login page वाला

v) login অন্তর্ভুক্ত
পুনৰ রেফ্রেশ
কর্যালয় ও
page/state
change করা
জন্য

→ spring security
সিইডি-এক্সেস
session
create
করা

update the default user name and password :

src → main → resources → application.properties

security config

spring.security.user.name = annu

spring.security.user.password = pass

→ But session will not work

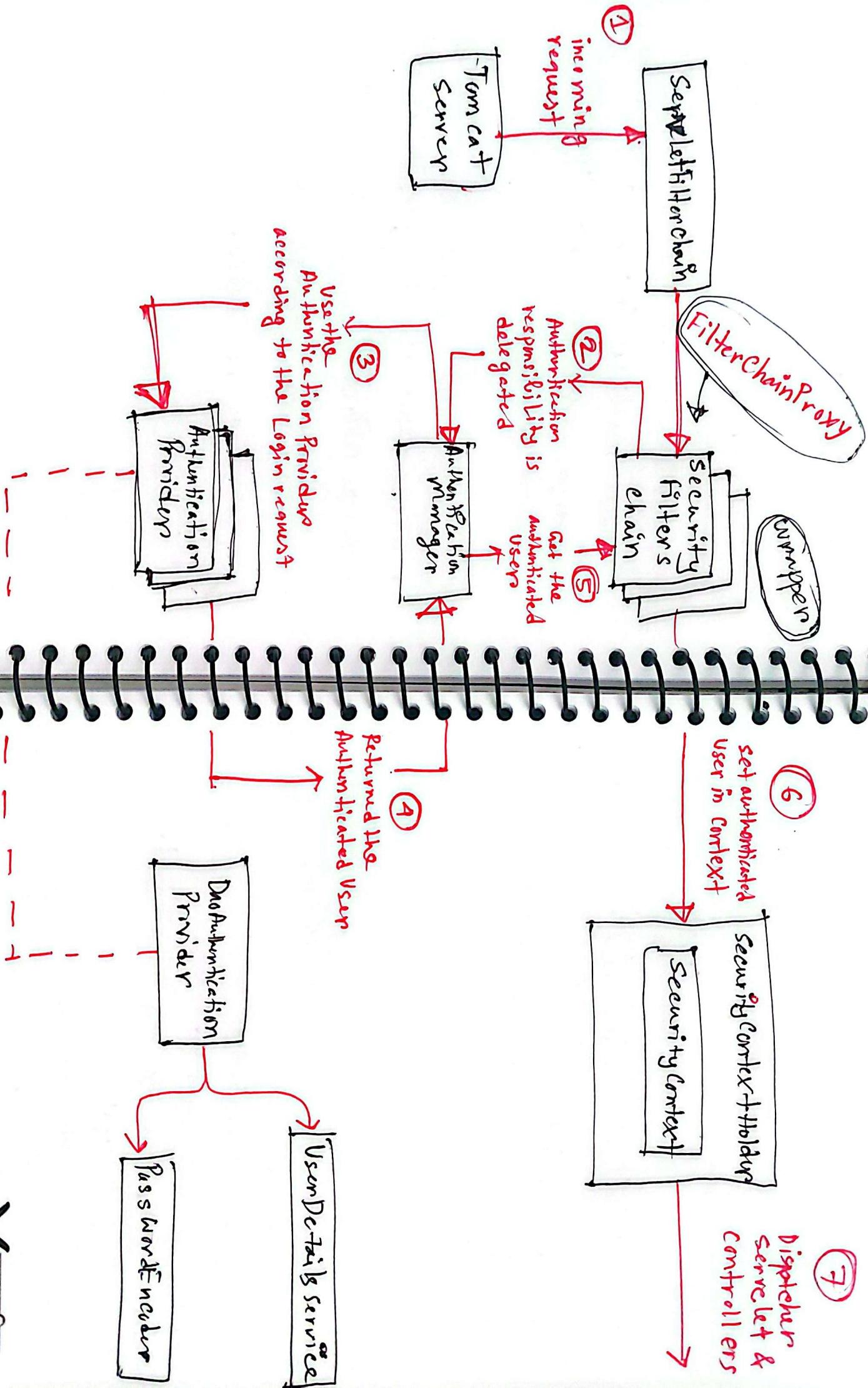
2) Security filter chain → **Middleware chain**

Filter chain → design pattern

⇒ Filter chain ज़ुँग through to
आम्हारा multiple independent filters
तातारूचे नाही एवं एका chain मध्ये
तातारूचे नाही

⇒ So that, **एरे request** ने आप्युक्त, येण-
• आम्हारा ट्राई करा **modify** करून नाही;
• आम्हारा एरिटर चेक करून नाही;
• आम्हारा डिझायन बदलून नाही

On the basis of different different
filters.



Path:

src/main/java/security/webSecurityConfig:-

WebSecurityConfig.java

```
import ... ;
```

```
@Configuration
```

```
public class WebSecurityConfig {
```

@Bean

It tells Spring:

Create this object and put it
into the Application Context, so it
can be injected elsewhere

Beans
always have to be
public

@Bean

```
public SecurityFilterChain
```

```
securityFilterChain(
```

```
HttpSecurity httpSecurity) throws
```

```
Exception {
```

```
httpSecurity
```

:

}

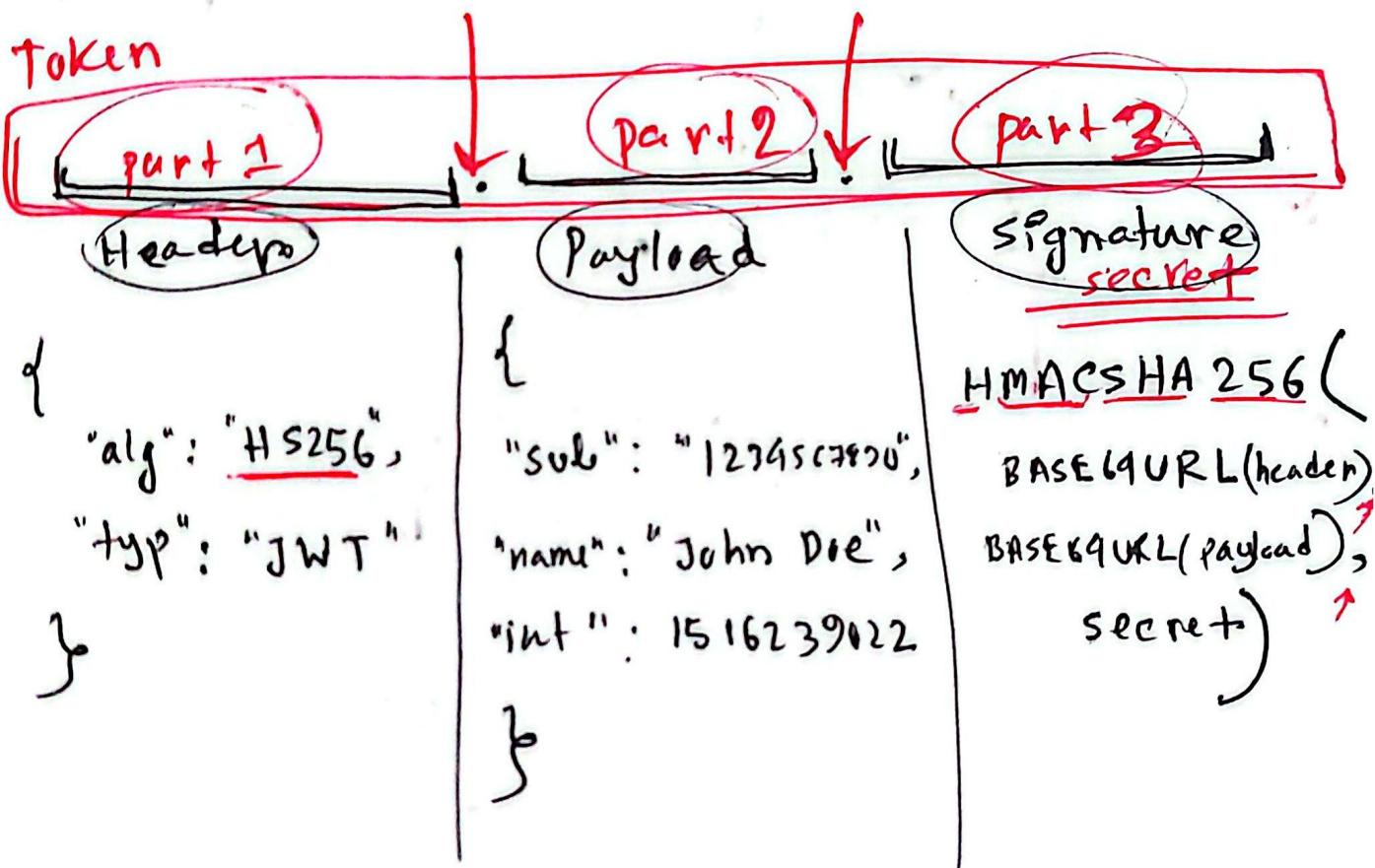
Spring Security 6

JWT Authentication

↳ can use in any backend
(with)

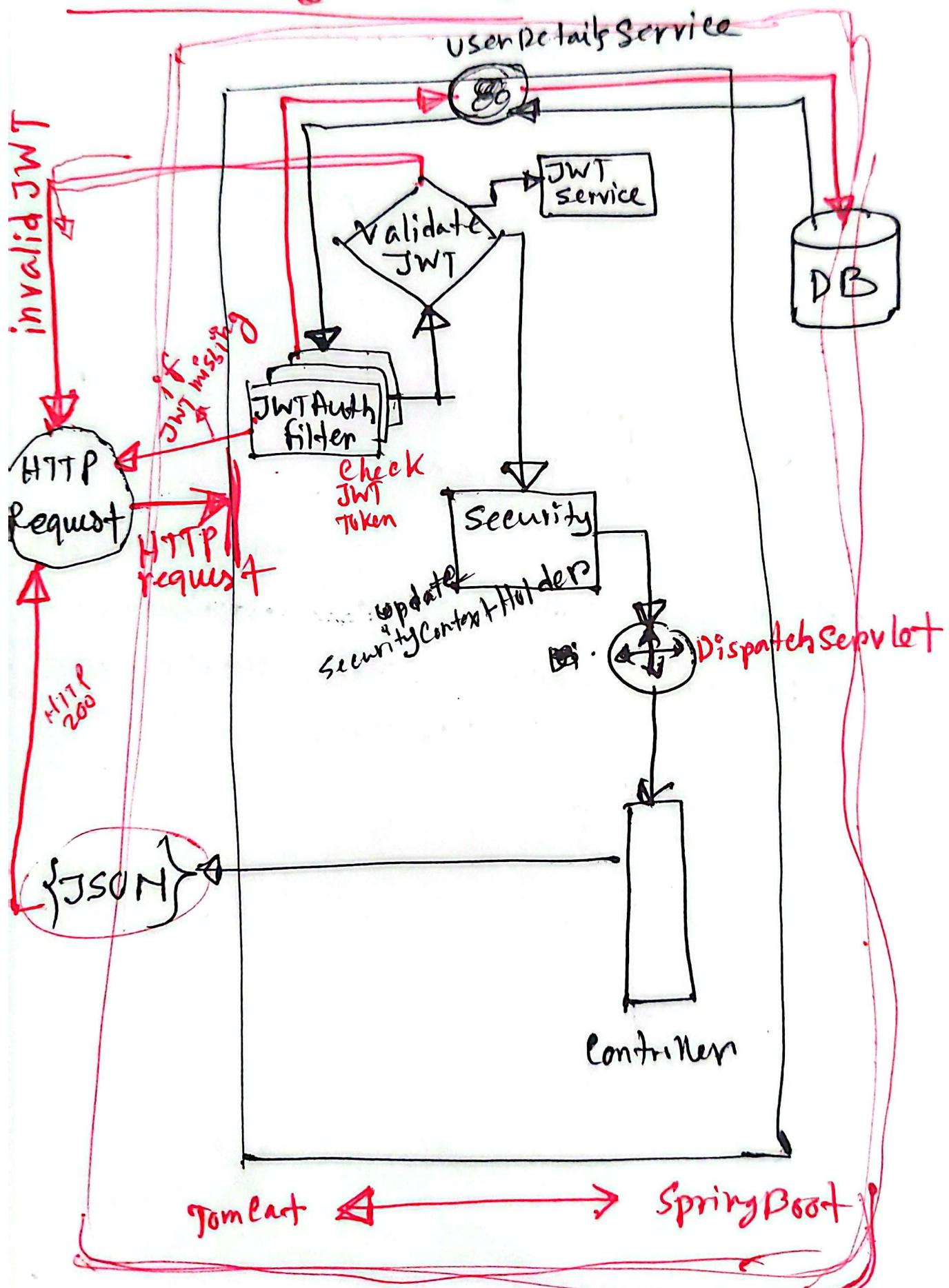
JWT → Widely used mechanism
to secure our authentication

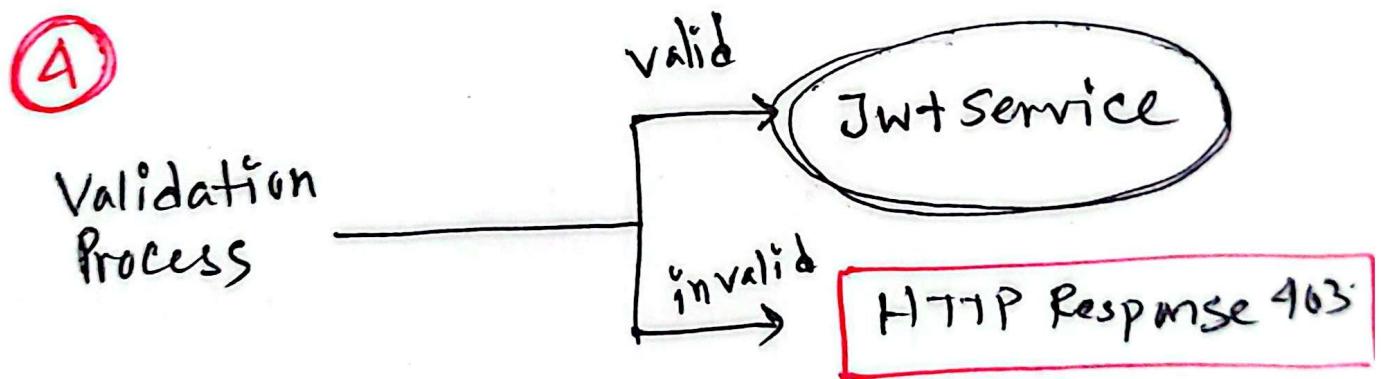
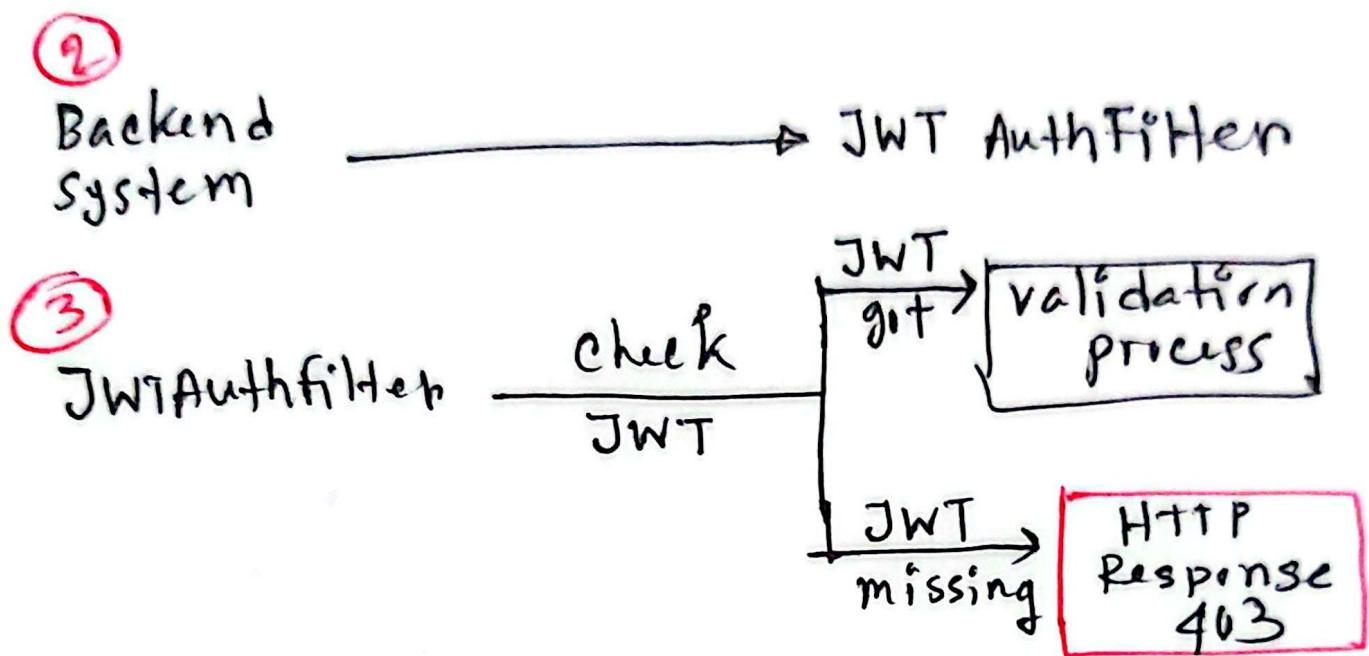
JWT → JSON Web Token



Spring Boot 3 + Spring Security 6

with JWT Authentication and Authorization

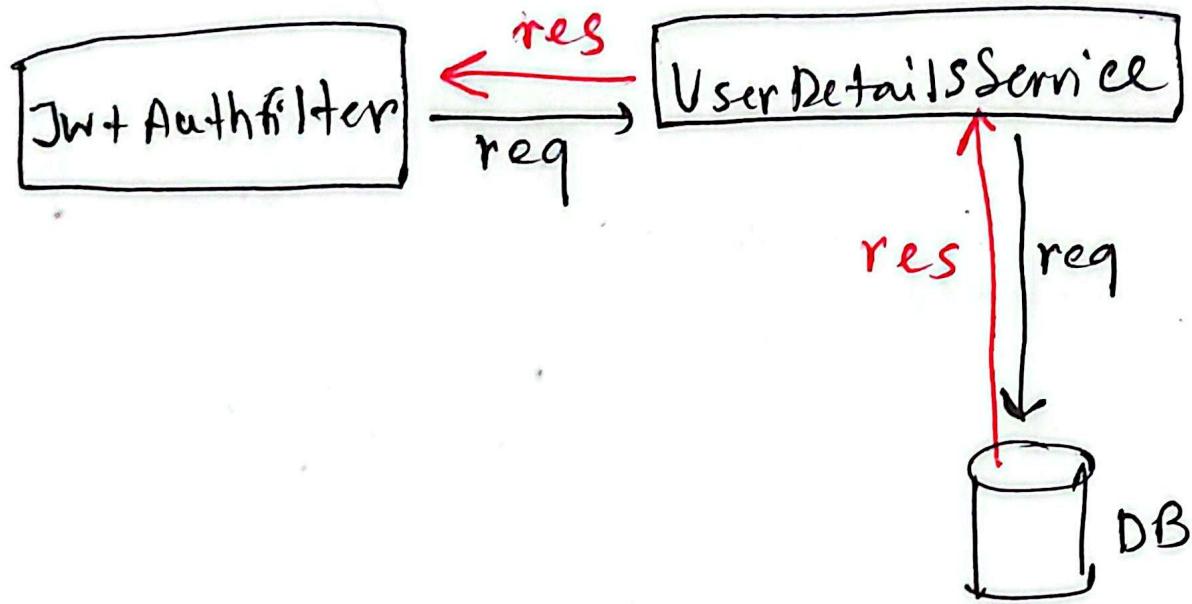




Validation Process:

JWT AuthFilter → make the call

(15+)



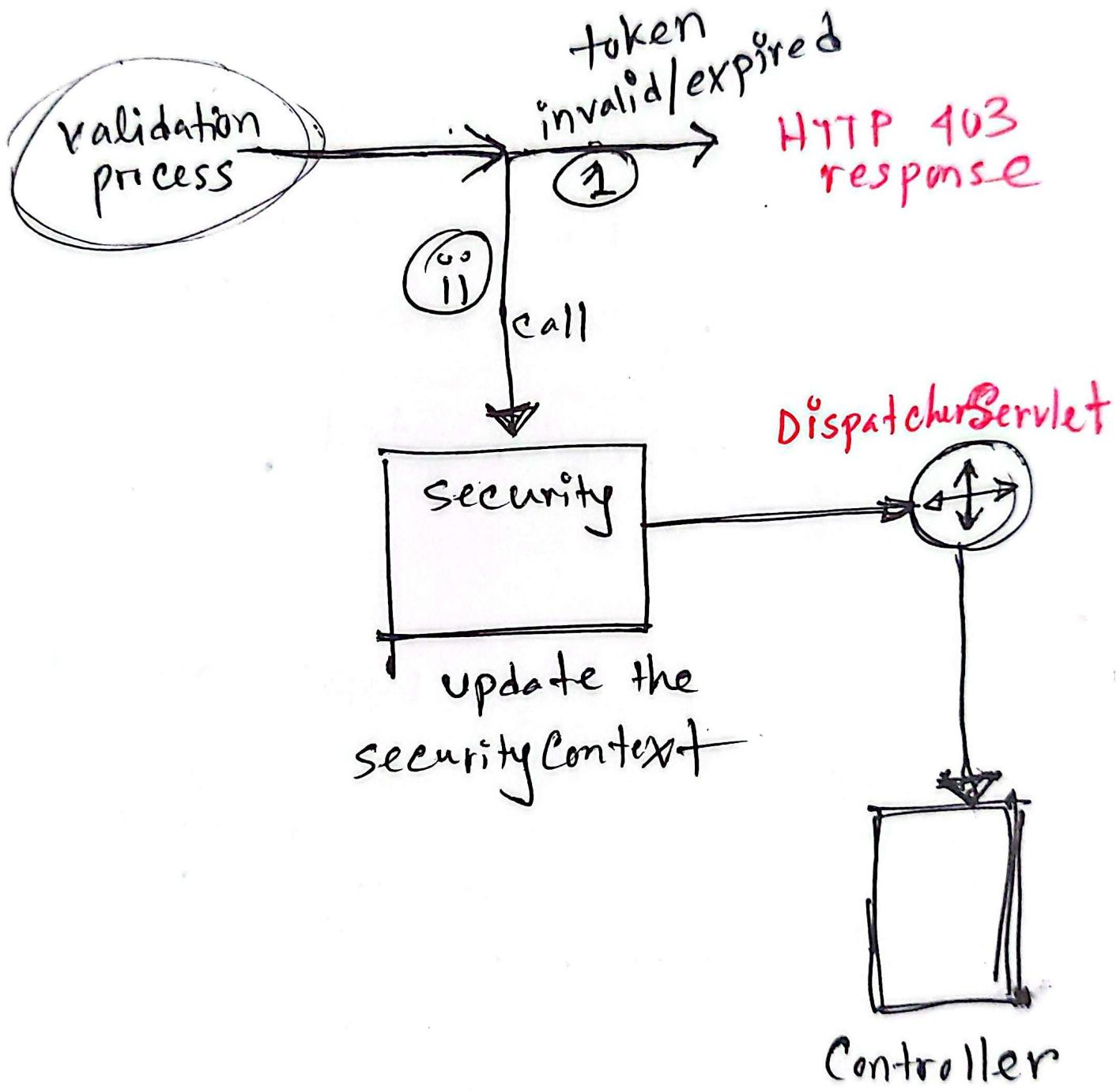
⑤ After validation Process

(i) The token is not valid/expired

(ii) Update the security context
and set this connected user

→ as a result that user
will be considered as

Authenticated User



⑥ Check Users Authentication

If the user is authenticated, and we will update the authentication manager.

→ Everytime we check if this user is authenticated for this request, the answer will

be Yes Authenticated

Once

Security context

will be updated

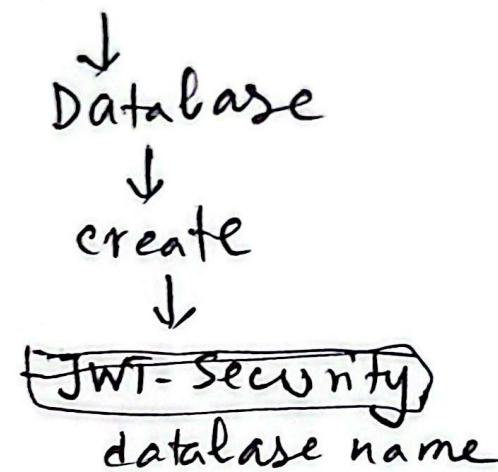
DispatcherServlet



it will automatically dispatch the req
and it will send req to the DispatcherServlet

Apply JWT Security :-

1. Create Spring Boot Project
Open it in IntelliJ Idea
2. PgAdmin → PostgreSQL 17



3. IntelliJ Idea →
src → main → resource → application.properties

4. application.properties:
~~spring:
 jpa:
 datasource:
 url: jdbc:postgresql://
 localhost: 5432/
 databaseName~~

4. configuration and Connection project with dB

application.properties:

spring.application.name = JWT-WebSecurity

spring

spring.datasource.url = jdbc:postgresql://
localhost:portno/
database name

spring.datasource.driver-class-name =
org.postgresql.Driver

spring.datasource.username = postgresql

spring.datasource.password = realpassword

spring.jpa.hibernate.ddl-auto = create-drop

spring.jpa.show-sql = true

spring.jpa.properties.hibernate.format-sql = true

spring.jpa.database-platform = org.hibernate.dialect.PostgreSQLPlusDialect

⑤ Now, start writing Backend codes

① Models

② Repository



packages
of a

JWT filters:

① new package → config

② config → JwtAuthenticationFilter.java
class