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

→ বারবার refresh না হওয়ায়, application অনেক বেশি fast হয়

Info:

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

Angular Component Life-Cycle Hook :

↓ that will automatically call when our pages are loaded

Phase

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

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 :-

- afterRender → runs every time all components have been rendered to the DOM.
- afterNextRender → runs once the next time that all components have been rendered to the DOM.

Change Detection :-

- ngOnChanges → runs everytime the components inputs have changed.

services in Angular

What are services?

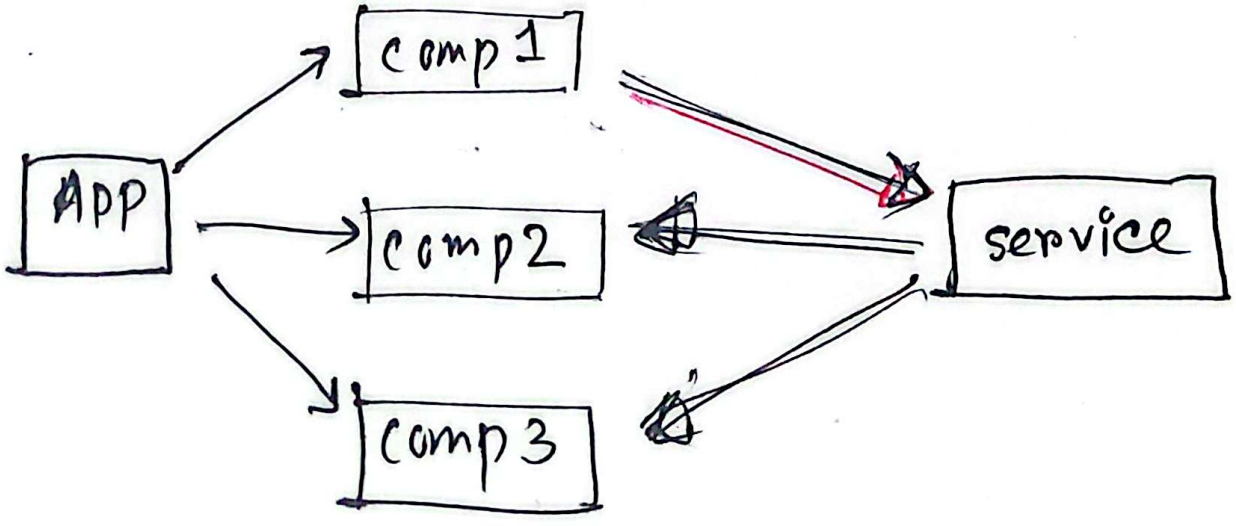
Angular services provide a way for you to separate Angular app data and functions that can be used by multiple components in your app.

separate → app data and functions

↓

can be used by

multiple components



↓

almost same functionality

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 {
```

when we don't want this to use outside this class

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

  productData: {
      name: string;
      category: string;
      price : number;
  }[] | undefined

  ~~const~~

  getData(){
    this.productData = this.product.getProductData();
  }
}
```

servie Name

function in the service

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:

product.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){ }
    ngOnInit(){
        this.product.productlist().subscribe
        (data) => {
        }}
}
```

life-cycle hook

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:

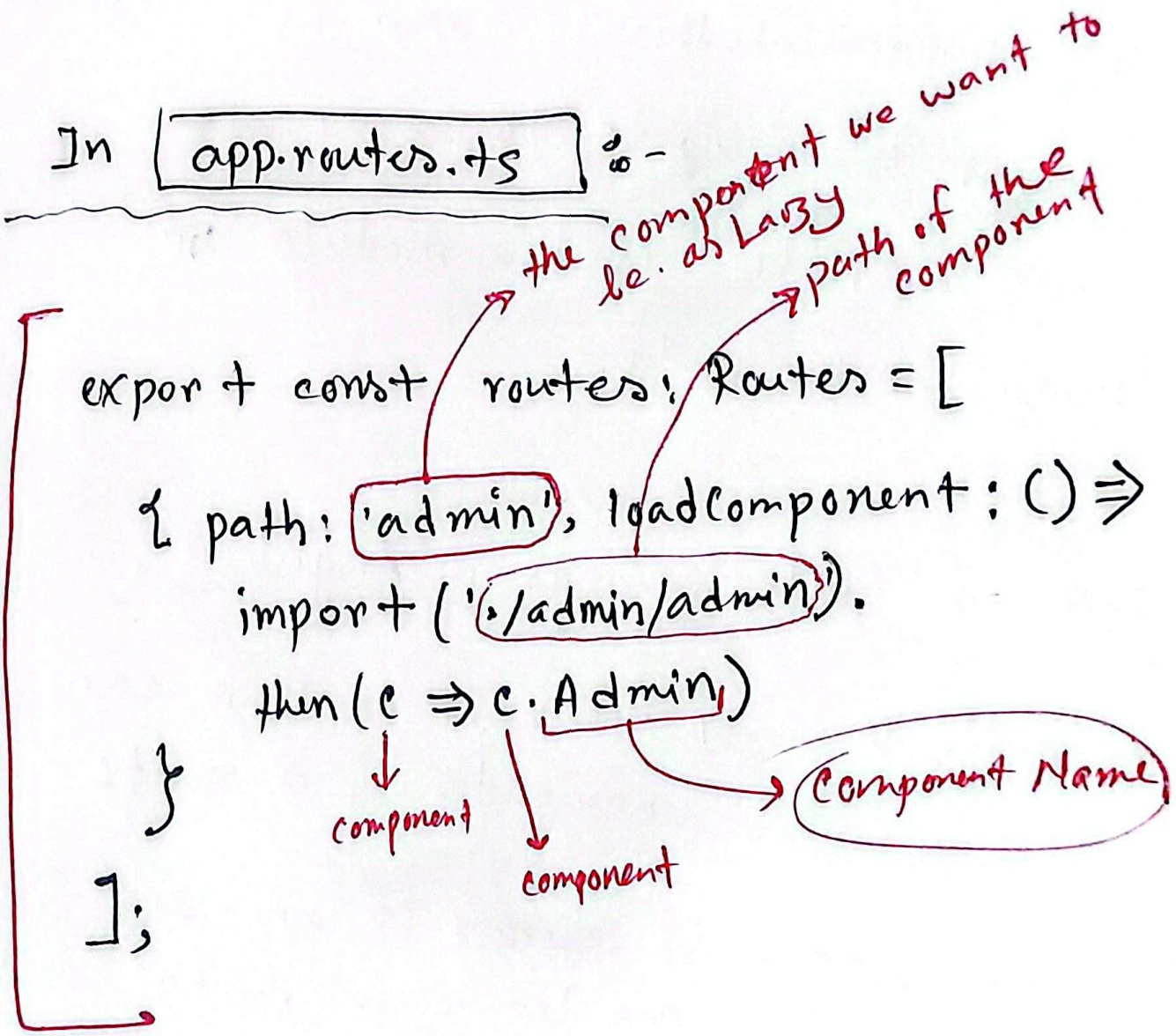
Interfaces are a feature of Typescript that allows us to define the structure or datatype.

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 :-



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

(iv) CanMatch → Allows or blocks navigation before route is matched

(v) CanLoad → Prevents lazy-loaded modules from loading

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

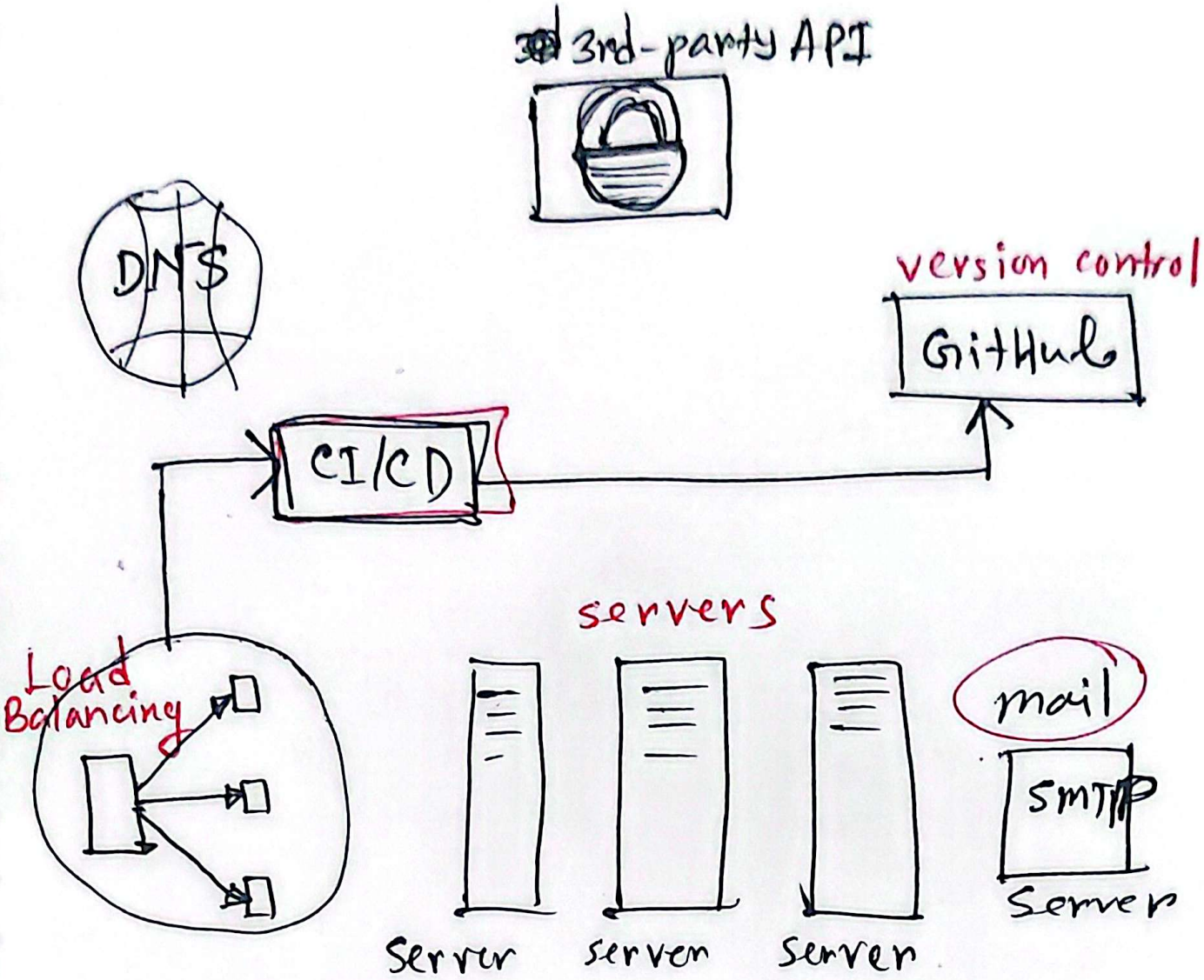
Auth Guards :

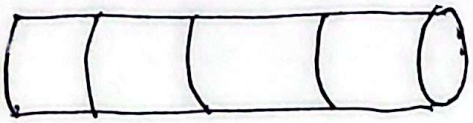
An Auth Guard in Angular is a Route Guard used specifically to protect routes from unauthorised access. canActivate guard

Spring Boot

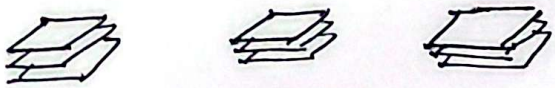
Java Framework

Backend

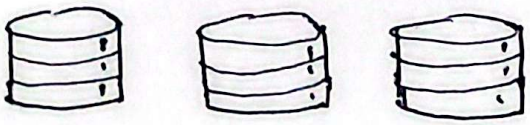




→ Queing for more traffic



→ Radis clustering



→ DB clustering

Xovir
Acyclovir USP

Spring Boot

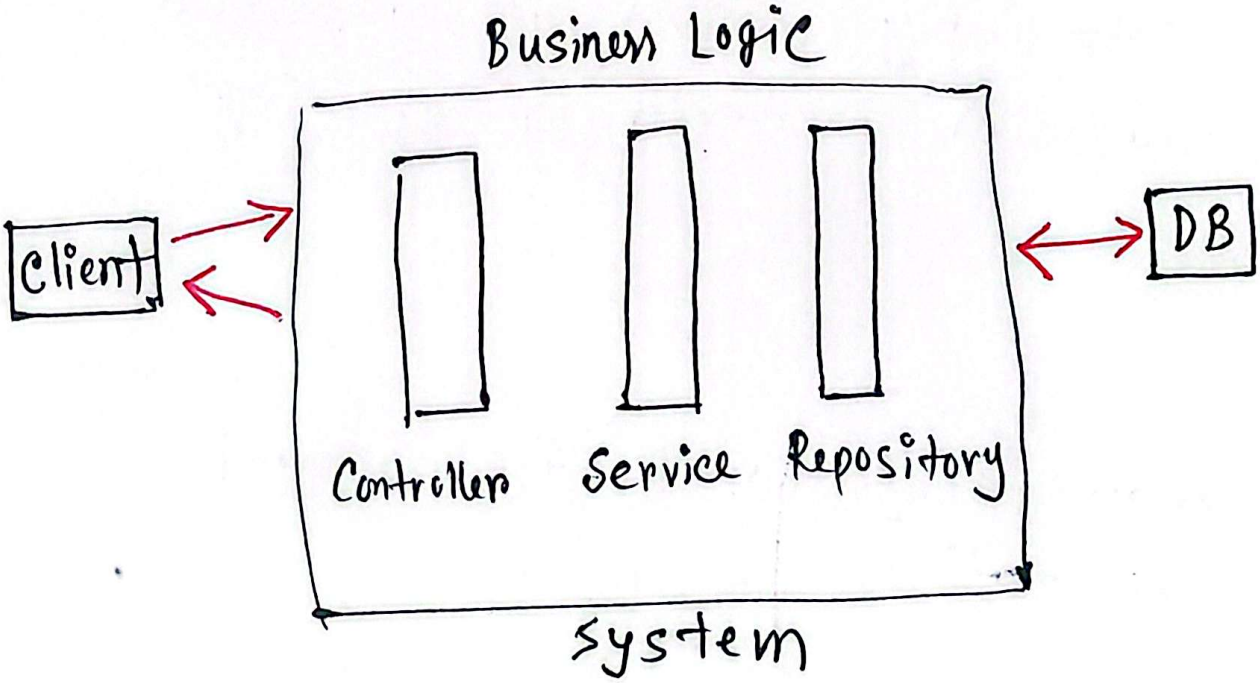
IoC:

↳ 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.

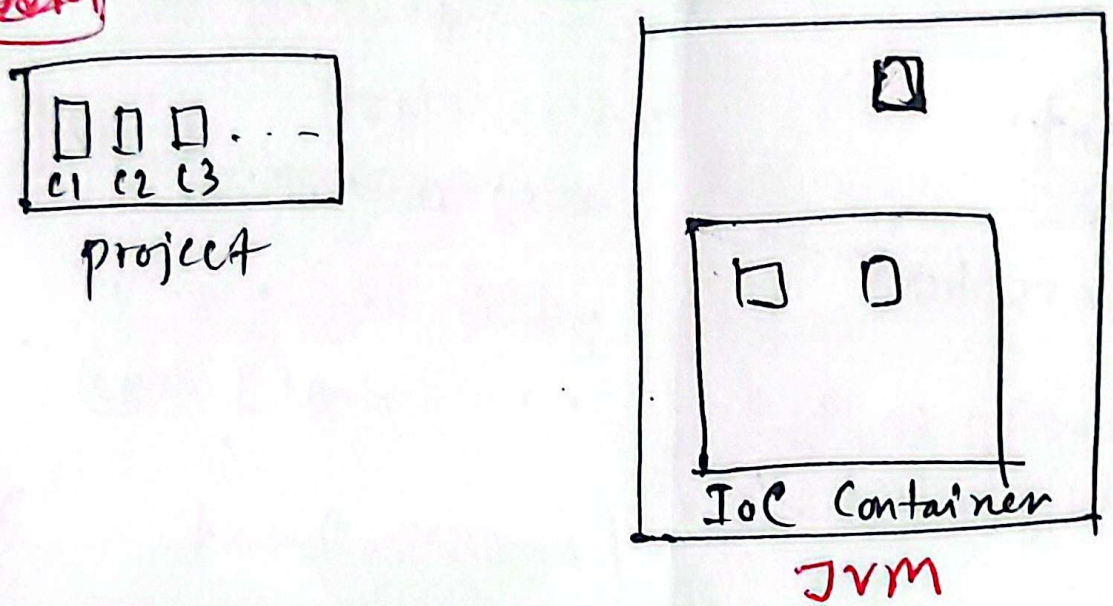
Your class simply "receives" its required dependencies instead of "creating" them.

Dependency Injection is the actual implementation of IoC.

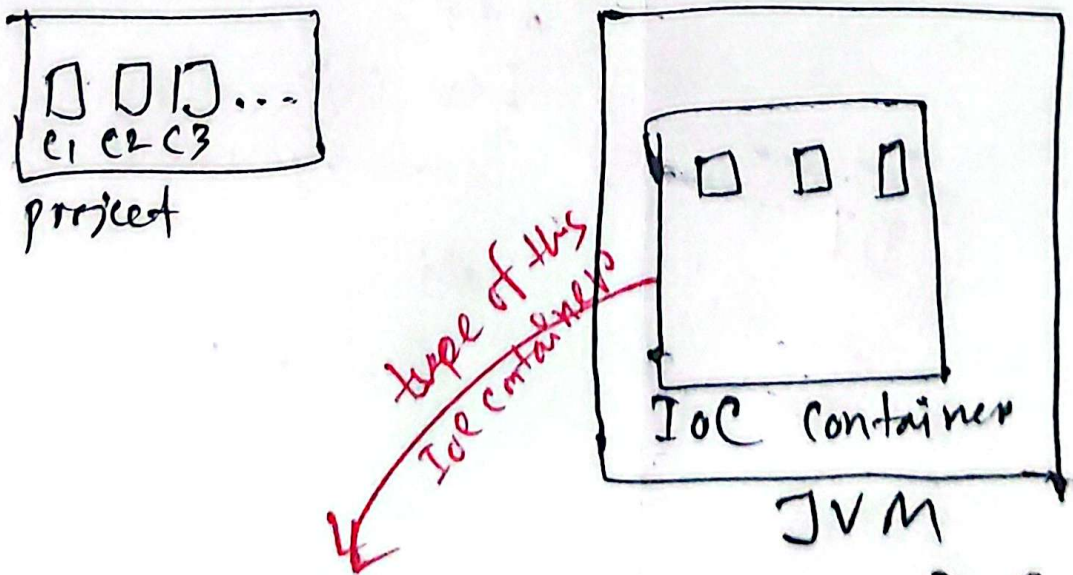
Spring Boot :-

Dependency Injection Using Spring Boot

→ new class is outside the container :



Lets assume the newly created class is already in the IoC container



main → java → com → App2

Examples

Dev.java

```
package com.tasrik.App2;

import ..... .Component;

@Component
public class Dev{

    sout("Working on a
        spring project");

}
```

App2Application.java

```
package . . . . ;

import . . . . ;

@SpringBootApplication
public class
    SpringApp2Application{

    public static void
    main(String[] args){

        ApplicationContext
        context = SpringApplication.run();

        Dev obj =
        context.getBean(Dev.class);

        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 & Dev.java connect हुए

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();
        sout("working on
            awsome project");
    }
}
```

Field Injection

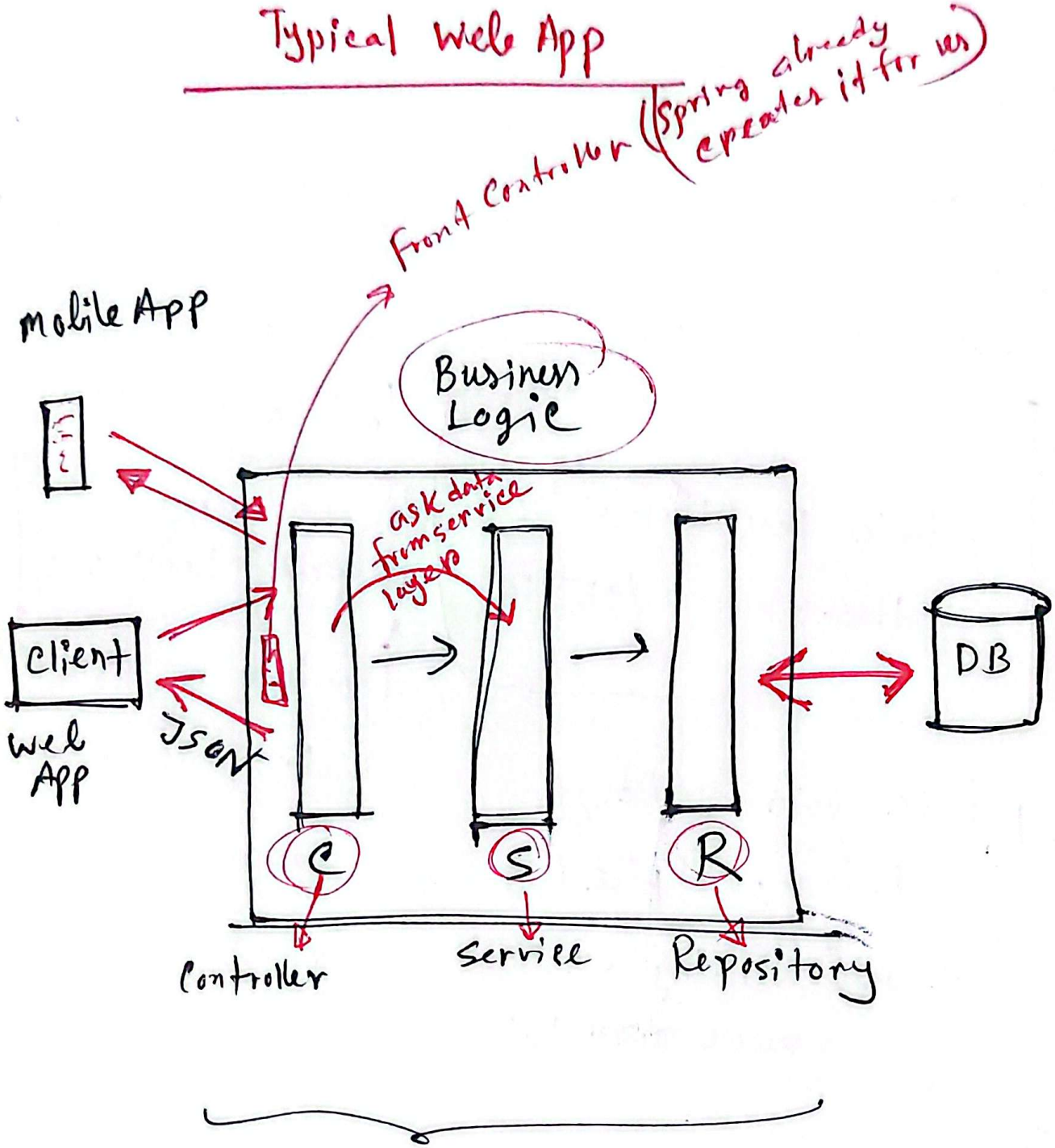
Dev.java

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

@Component
public class Dev {

    @Autowired
    private Laptop laptop;

    public void build(){
        laptop.compile();
        sout("    ");
    }
```

Setter injection

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

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

27/11/2025

Controller:

A controller is a Java class that:

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

→ Proces 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.

lombok 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

TableName

prodID, prodName, Price


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

30/11/2025

ORM tools

- Hibernate → full flaged
- 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 {
    psvm(String[] args){
        Animal a = new Dog();
        a.sound();
        a.eat();
    }
}
```

Using JPA Repository Step 1

Reprsitory Package → Interface → ProductRepo.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

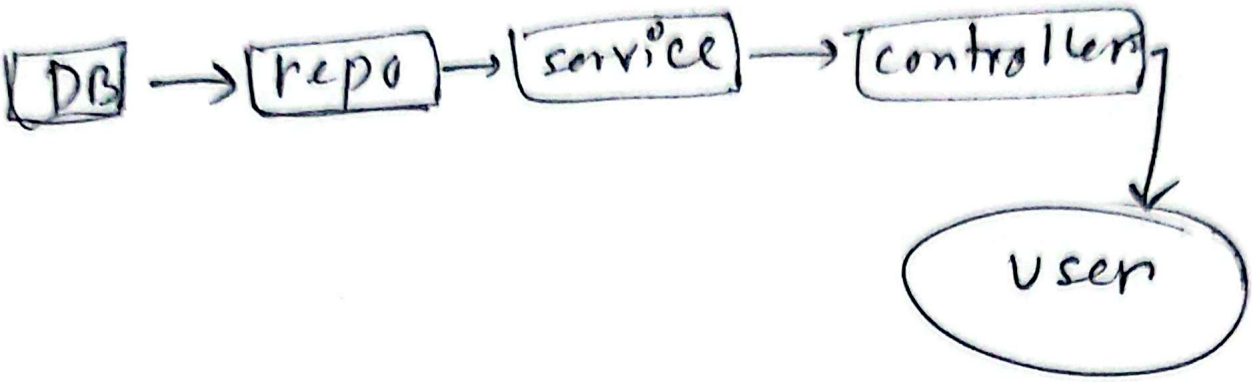
releaseDate → Date

available → boolean

quantity → Int

? Img →

when data fetching what happens



CORS Error is Spring Boot g

→ 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 Open ID 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 fiters and filter chain.

1) Add Spring Security Dependency

↓

Dependency add করলে আমার প্রজেক্ট এ spring security এর কিছু default behavior add হয়ে যাবে → EX:

i) route → protected Route / Authinticated Route

ii) to access a route → login first

iii) to login → generated spring security password use করতে হবে

iv) user → by default user নামে একটা user create করে দিয়ে আমার login page এ

v) login করার পর refresh করলে page/state change হয় না → spring security নিজেই একটা session create করে

update the default user name and password :

src → main → resources → application.properties

```
# security config
spring.security.user.name = anuj
spring.security.user.password=pass
```

→ But session will not work

2) Security filter chain ⟶ Middleware chain

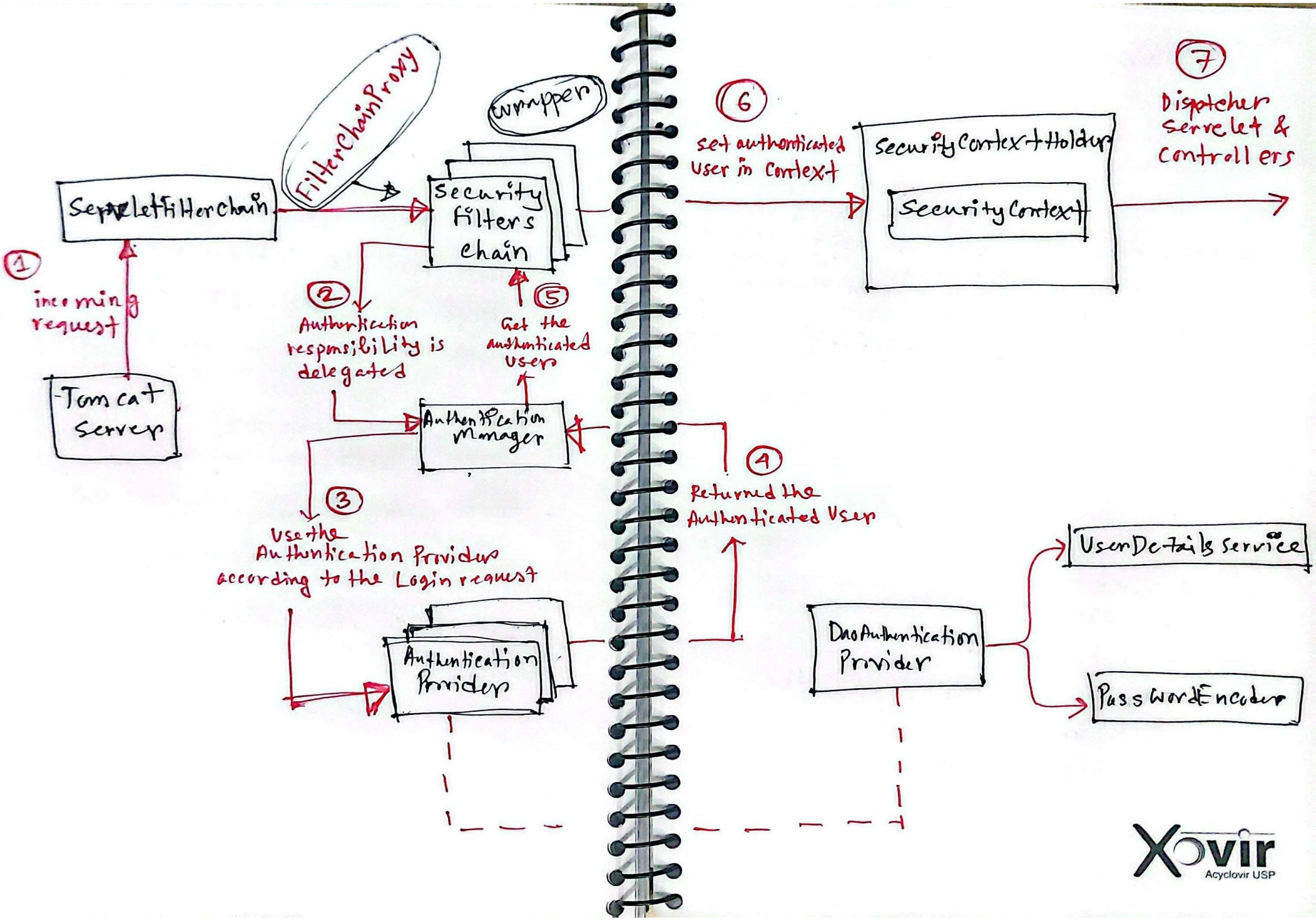
Filterchain ⟶ design pattern

⇒ Filter chain এর through তে আমরা multiple independent filters বানাতে পারি এবং একটা chain এ রাখতে পারি

⇒ so that, যেই request টা আসুক, যেন -

- আমরা সেটাকে modify করতে পারি;
- আমরা সেটাকে check করতে পারি.
- আমরা সেটাকে change করতে পারি

on the basis of different different filters.


FilterChainProxy
wrapper
ServletFilterChain
Security filters chain
1
incoming request
Tomcat server
2
Authentication responsibility is delegated
5
Get the authenticated user
Authentication Manager
3
Use the Authentication Provider according to the Login request
Authentication Provider
4
Returned the Authenticated User
6
set authenticated user in context
SecurityContextHolder
SecurityContext
7
Dispatcher servelet & controllers
DaoAuthenticationProvider
UserDetailsService
PasswordEncoder
Xovir
Acyclovir USP

path:

src/main/java/security/WebSecurityConfig :-

WebSecurityConfig.java

@Bean
It tells Spring:
Create this object and put it into the Application Context, so it can be injected elsewhere

```
import . . . ;

@Configuration
public class WebSecurityConfig {

    @Bean
    public SecurityFilterChain
        securityFilterChain(
    HttpSecurity httpSecurity) throws
        Exception {
        httpSecurity
        :

}
```

Bean always have to be public

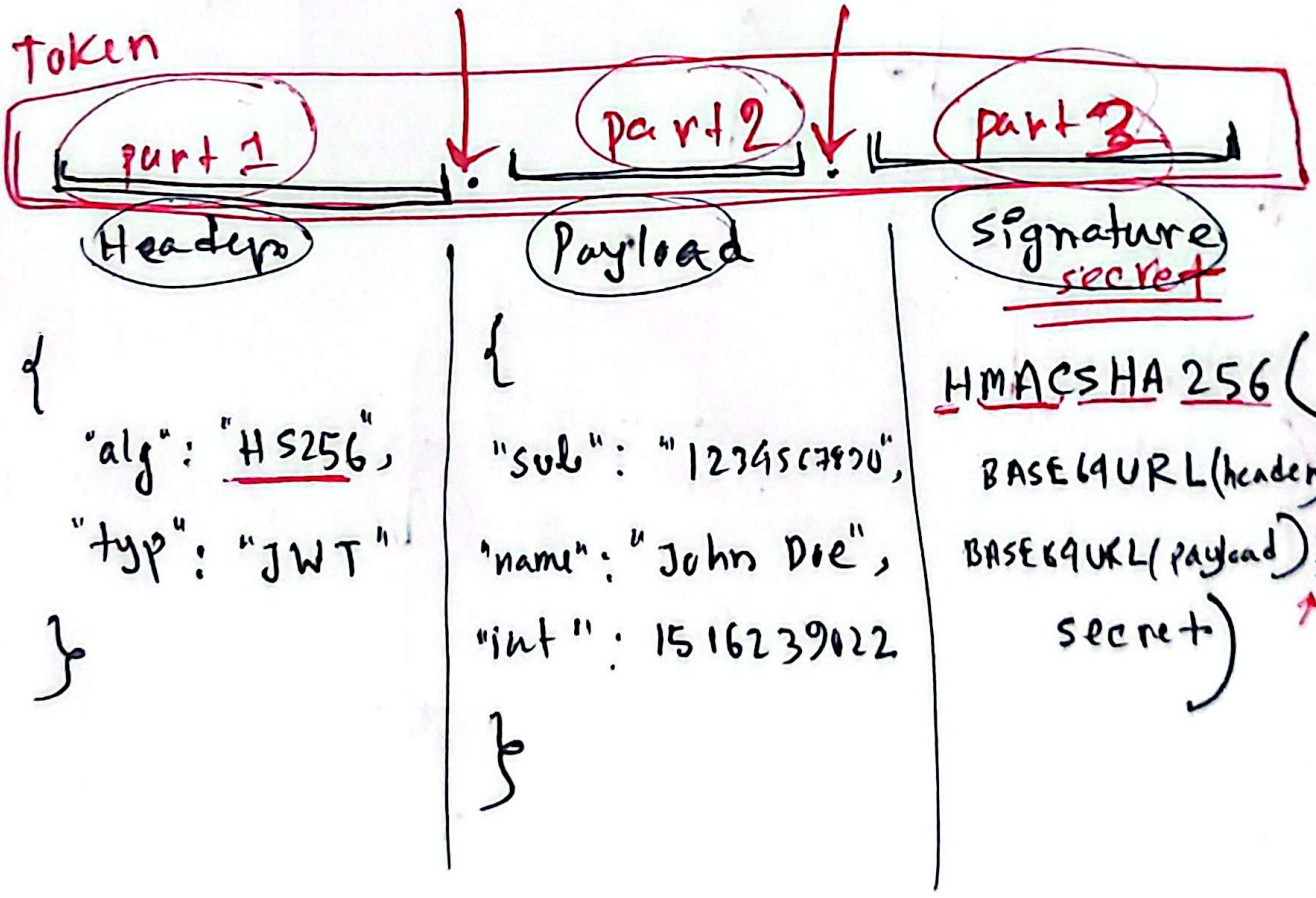
Spring Security 6

JWT Authentication

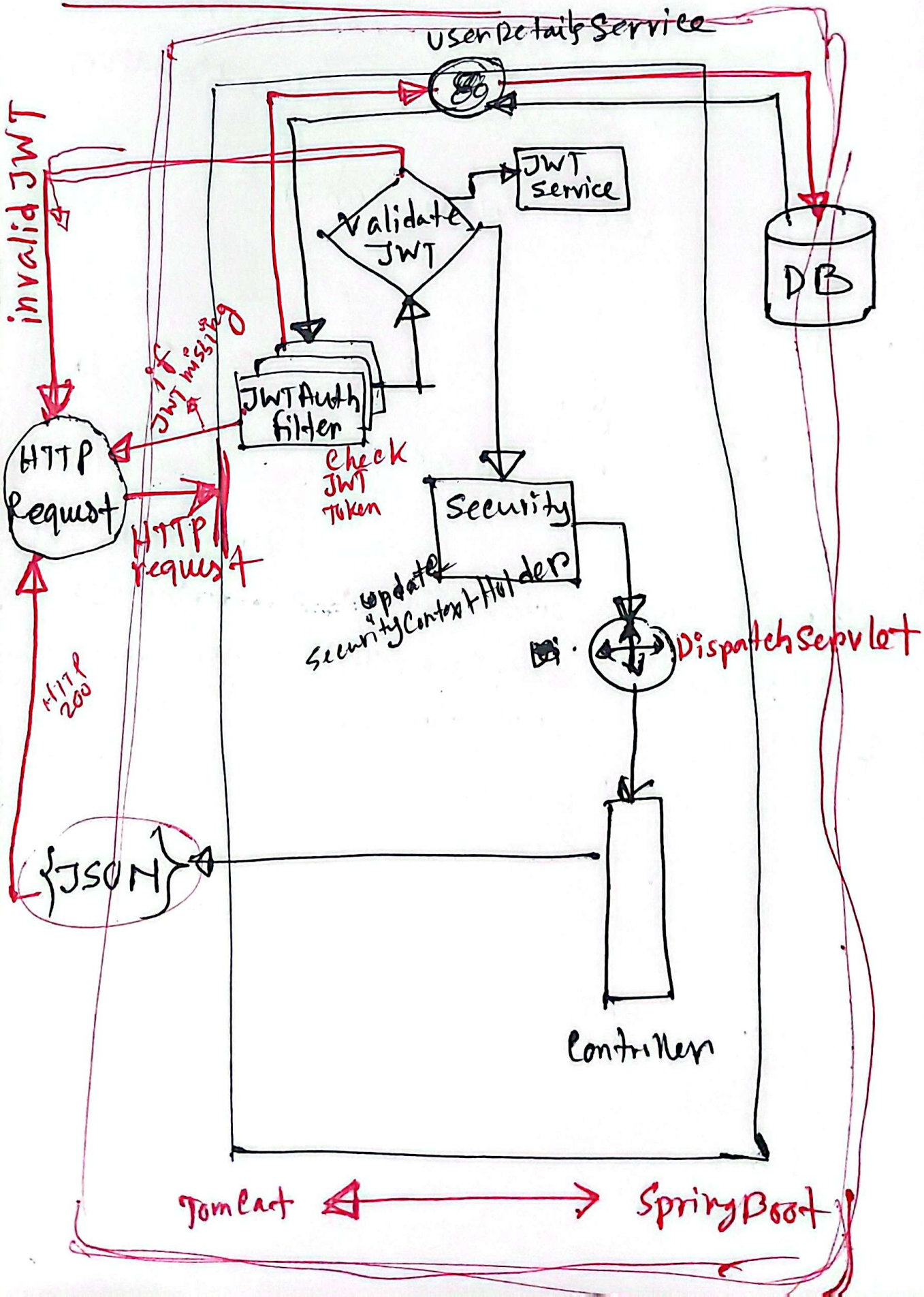
↳ can use in ~~any~~ with backend

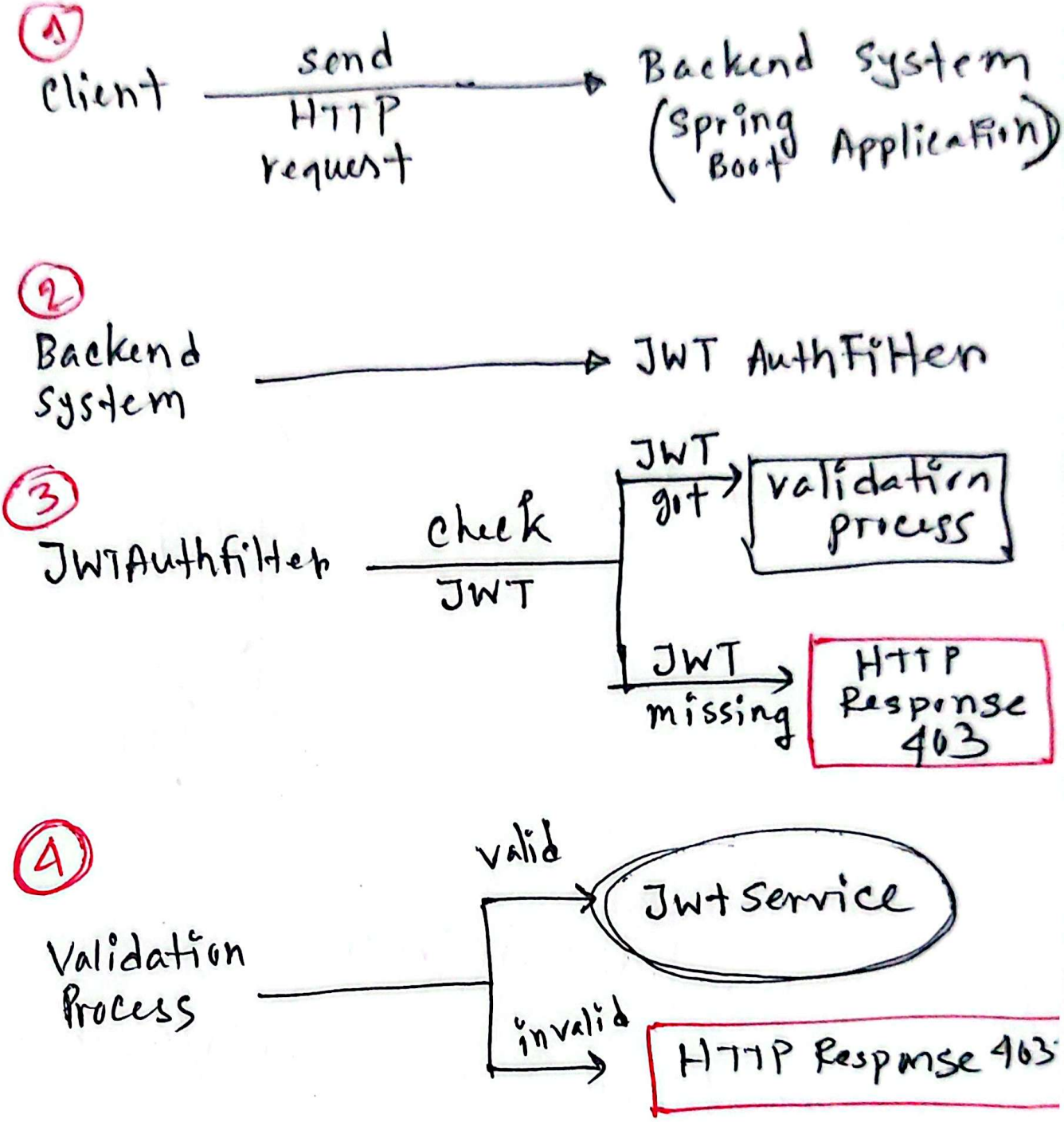
JWT → Widely used mechanism to secure our authentication

JWT → JSON Web Token



Spring Boot 3 + Spring Security 6 with JWT Authentication and Authorization &

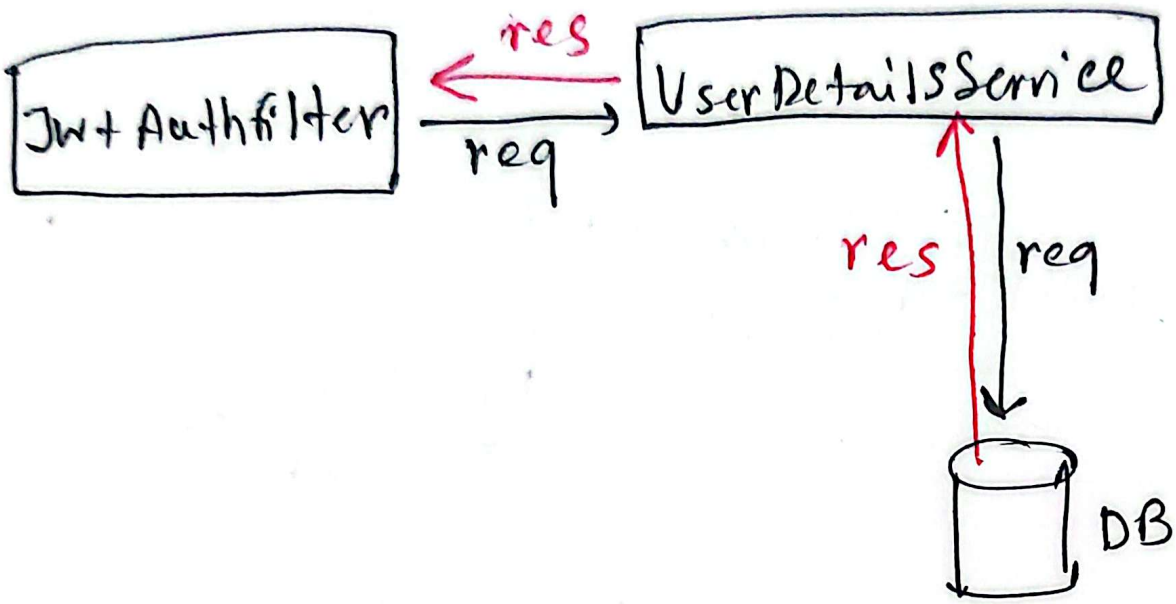




Xovir
Acyclovir USP

Validation Process:

JWTAuthFilter ⟶ (1st) make the call



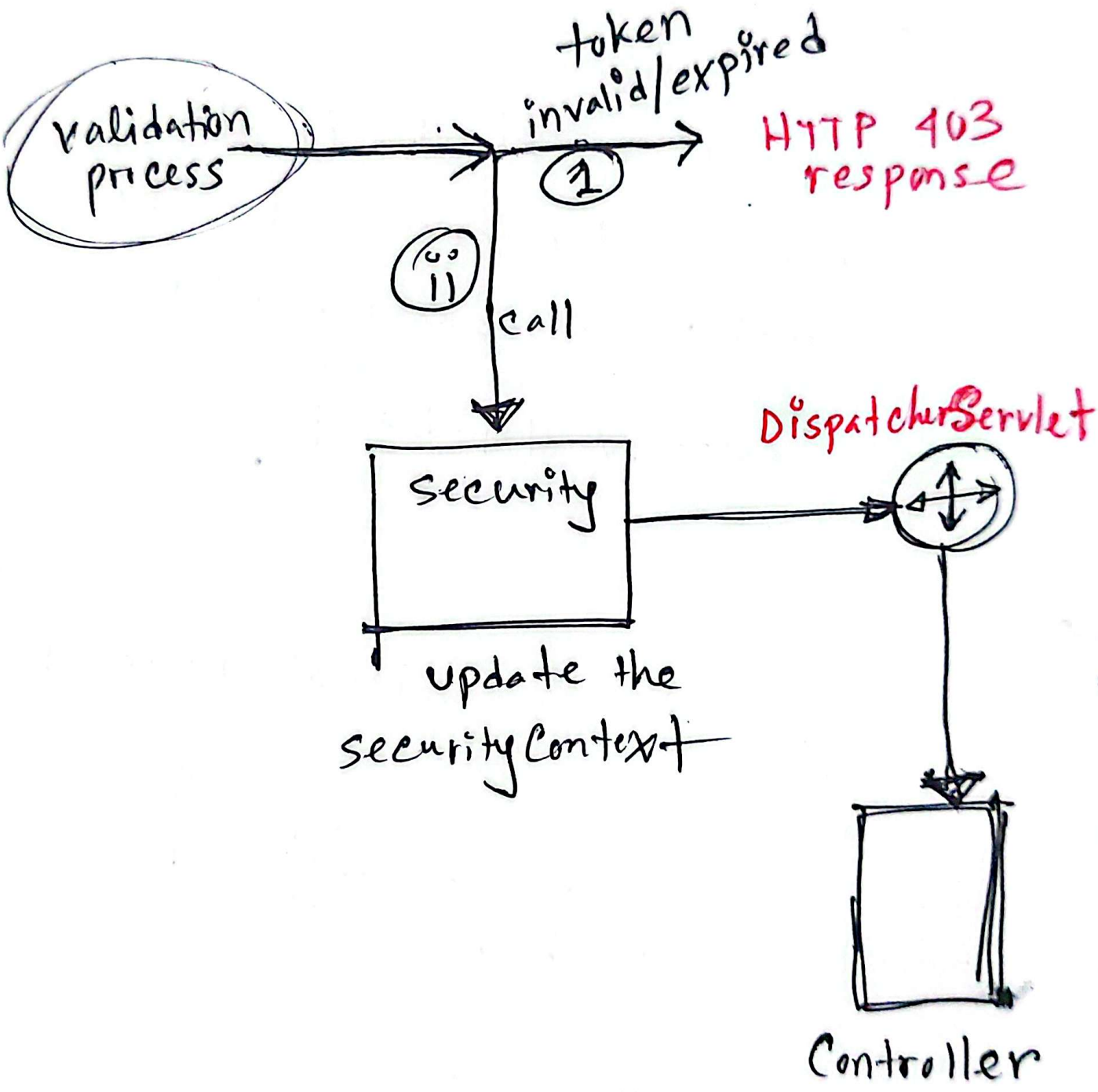
(5) After validation Process

(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

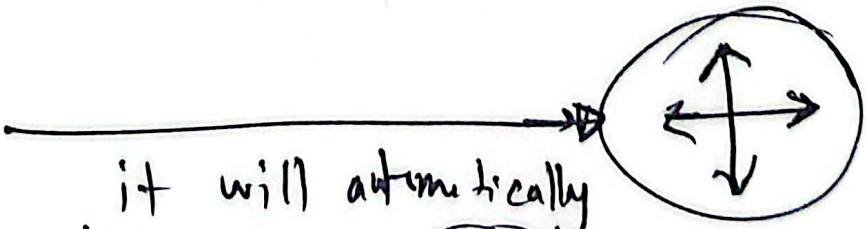
If the user is Authenticated, and we will update the Authentication manager.

→ Everytime we check if this user is authenticated for this (request), the answer will be (Yes) Authenciated

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. PgAdmain ⟶ postgreSQL 17
   ↓
   Database
   ↓
   create
   ↓
   JWT-security
   database name

3. Intellij Idea ⟶
   src → main → resource → application.properties

4. application.properties :

   spring:
   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/
databasename

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

spring.datasource.usepname = postgresql
spring.datasource.password = real-password

spring.jpa.hibernate.~~ddl~~ddl-auto = create-drop
spring.jpa.show-sql = true
spring.jpa.properties.hibernate.format_sql = true
spring.jpa.database-platform = org.hibernate.
dialect.
PostgreSQLDialect

(5) Now, start writing Backend codes

(i) Models

(ii) Repository

} packages of

JWT filters:

(i) new package → config

(ii) config → JwtAuthenticationfilter.java

class

Refresh token :-

JPA — auto property :-

↳ in application.properties

spring.jpa.hibernate.ddl-auto =

create-drop : create a database when the app starts and drops them when it stops

create : drops and recreates a table on start up.

→ useful for testing but we will lose all of our data each time

none: simply disables automatic schema generation

update: It creates and updates table and columns based on the entity classes without deleting existing data.

super convenient for development

validate: checks if the database matches our entities without making any changes.

If it does not match, the app won't start.

DTO

→ package

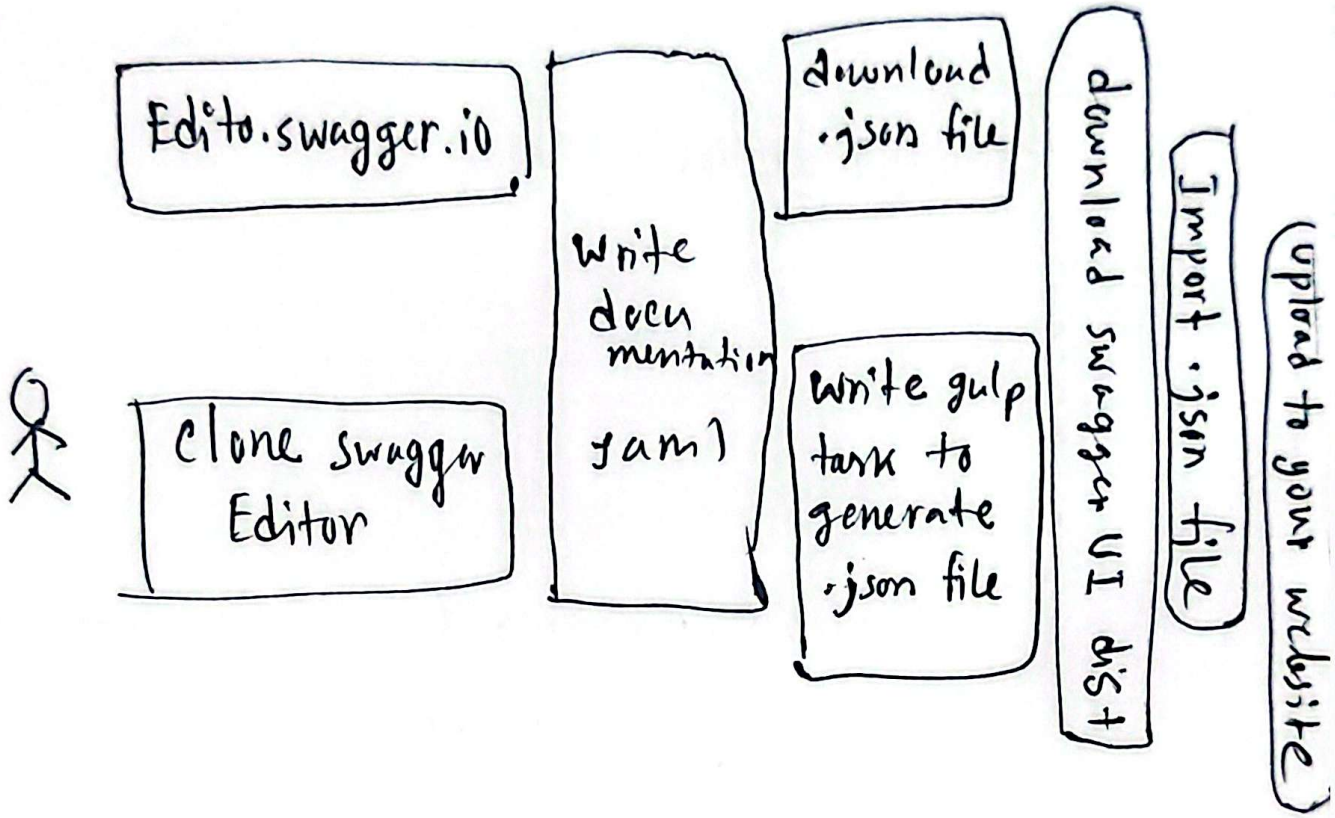
Data Transfer object

pom.xml

↓

project object model

Swagger API documentation



step - 1

→ add dependency in pom.xml

springdoc-openapi-starter-webmvc-ui

step - 2:

in config/SecurityConfig.java

```
.requestMatchers(
    "/api/auth/**",
    "/swagger-ui/**",
    "/swagger-ui.html",
    "/v3/api-docs/**"
).permitAll()
.anyRequest().authenticated()
```

Step-3:

Create a new file (class) in config folder.

```
public static final
String [] AUTH_PUBLIC_URL = {

        "/api/auth/**",
        "/swagger-ui/**",
        "/swagger-ui.html",
        "/v3/api-docs/**"

};
```

Step-4:

replace the urls with variable

```
= =
- -
.requestMatchers(
    AppConstant.AUTH_PUBLIC_URL
).permitAll()
.anyRequest().authenticated()
```