Spring Boot

DATE
PAGE

SpringBoot ⇒

It is a part of Spring-framework
it is module of Spring

Difference b/w Spring and SpringBoot

Features	Spring framework	SpringBoot
① Configuration	XML / JavaBasedConfig	Auto-Configuration
② Setup	Long	very fast
③ Server	External	Embedded.
④ Dependencies.	Manual	Starter-based.
⑤ Boiler-plate code	High	low-level.
⑥ Microservices.	Hard.	very easy.

SpringBoot Basically works on a Principal :—

Convention over Configuration.

★ In short :—

- Spring → Power but complex
- SpringBoot → easy (Spring but easy).

SpringBoot = SpringFramework + AutoConfiguration + Embedded Server

Features : # Starter - Dependencies. :— ①

⇒ Starters are pre-defined dependency Conflicts

(1) Spring-boot-starter-web → (web + REST)
(2) Spring-boot-starter-data-JPA → (JPA + Hibernate)
(3) Spring-boot-starter-security → (Security)

✓ Reduces dependency Conflicts

STS → Spring Tool Suit (for SpringBoot Application)

Mast Boot Application.

(2) ★ Embedded Server.

Spring boot includes

- Tomcat (default)
- Jetty
- Netty

No need to deploy WAR file

(3) ★ Standalone App...

- web app.
- Console app.
- Microservice.

Can run as Standalone Java app

- only main method required

(4) ★ Externalized Configuration :-

⇒ Spring Boot Supports

① application.properties (key value pair ke form me entry karo)

② application.yml (tree structure ke form me entry karenge)

③ • Environment variables (Data whose security is very important, we do entry in environment variable)

(5) Auto-Configuration :-

- Automatically configures beans based on dependencies.
- Example :-

If you add spring-boot-starter-web, it auto-configures :-

(1) DispatcherServlet
(2) Tomcat
(3) Jackson (JSON)

⇒ No need to write extra Configuration.

SpringBoot :-

SpringBoot is a java-framework built on top of spring that simplifies application development.

It eliminates boiler plate code with auto-configuration.

SpringBoot comes with an embedded server, making applications production ready.

⇒ why SpringBoot ?

- fast Development
- Auto-Configuration
- microServices Friendly
- Seamless Integration
- wide Adoption.

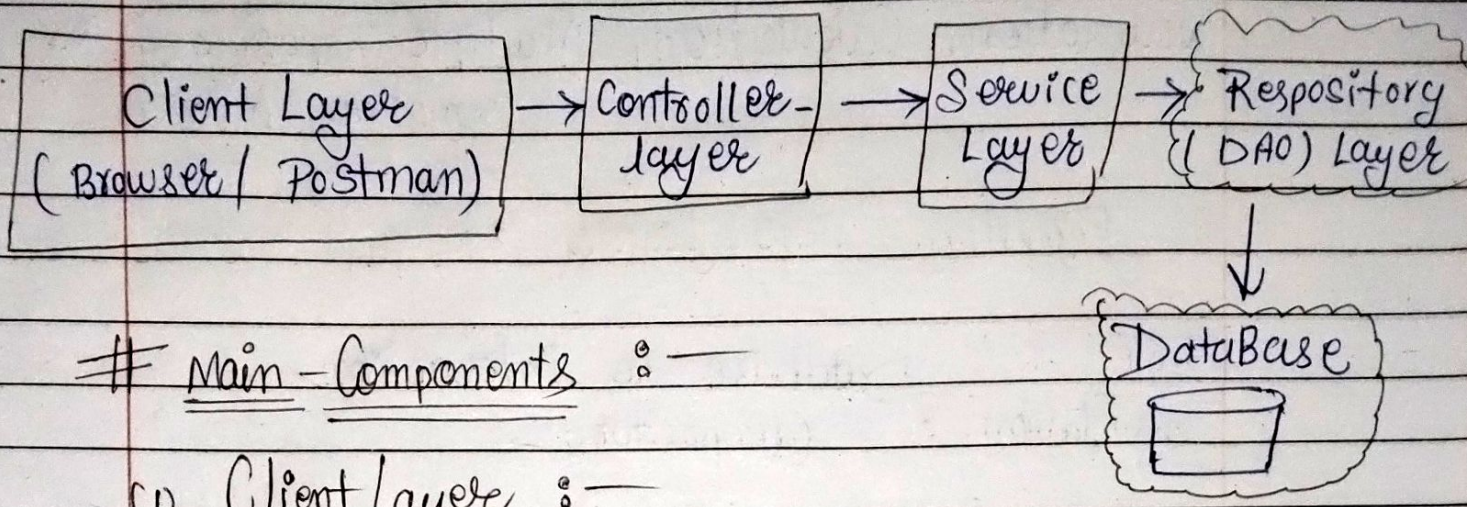
⇒ why SpringBoot came ?

Traditional Spring has problems like :-

- Too much **XML** Configuration
- Manual dependency management.
- Need to configure TomCat Separately
- Long set-up time

⇒ SpringBoot solves all these.

Sprint - Boot Architecture :-



Main - Components :-

(1) Client Layer :-

- Brawser, MobileApp, Postman, etc.
- HTTP request send karta hai (GET, POST, PUT, DELETE).

(2) Controller layer :-

Role :- Request ko receive karna & response dena

★ Annotation :-

@RestController @RequestMapping
@Controller @GETMapping, @PostMapping

★ Work :-

- Request accept karta hai
- Service layer ko call kareta hai
- Response return karta hai (JSON / View).

(3) Service Layer :-

Role :- Business logic handle karna.

Annotation :- @Service

WORK :-
Validation, Calculation, Multiple Repository ko combine karna.

④ Repository (DAO) Layer :-

Role :- Database se interaction
Annotation :- @Repository

Interface :- JPARepository

Work :-
- CRUD operation
- Query execution.

⑤ Database Layer :-

- MySQL, PostgreSQL, Oracle, H2, etc.
- Data permanently store hota hai

⇒ Benefits of Spring Boot Architecture.

- Loose Coupling, • Easy Maintainance,
- Faster Development, • No XML Configuration,
- Production Ready

Spring Boot - Annotations

⇒ Annotations in Spring Boot are metadata that simplify configuration and development. Instead of XML, annotations are used to define beans, inject dependencies and create REST endpoints.

⇒ They reduce boilerplate code and make building applications faster and easier.

Core Spring Boot Annotations :-

① @SpringBootApplication annotation.

- This annotation is used to mark the main class of a Spring Boot Application.
- It encapsulates @SpringBootConfiguration, @EnableAutoConfiguration and @ComponentScan annotations with their default attributes.

Example :-

```
@SpringBootApplication.
public class DemoApplication{

    // main Driver Method
    public static void main (string[] args){
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

(2) @SpringBootConfiguration annotation.

- Indicates that a class provides configuration for a SpringBoot application.
- Alternative to Spring's @Configuration.
- Included automatically with @SpringBootApplication.

(3) @EnableAutoConfiguration Annotation.

- This annotation auto-configures the bean that are present in the classPath.
- Enable's Spring Boot's auto-configuration mechanism.

(4) @ComponentScan Annotation.

- Tells Spring where to search for components (@Controller, @Service, @Repository, etc).
- Typically used with @Configuration.

Request Handling and Controller annotations

Some important annotations :-

(1) @Controller :-

- This annotation provides Spring MVC features
- It is used to create Controller classes and simultaneously it handles the HTTP requests.

Generally we use @Controller annotation with @RequestMapping annotation to map HTTP requests with methods inside a Controller class.

(2) @RestController :-

- This annotation is used to handle Rest APIs, and also used to create Restful web services using Spring MVC.
- It encapsulates @Controller annotation and @ResponseBody annotation with their default attributes.

@RestController = @Controller + @ResponseBody

(3) @RequestMapping Annotation :-

- Maps HTTPs requests to handler methods.
- Supports GET, POST, PUT, DELETE etc.

for handling specific HTTP requests, we can use

- @GetMapping, @PutMapping, @PostMapping,
- @PostMapping, @PatchMapping, @DELETEMapping

(4) @RequestParam Annotation :-

@RequestParam annotation is used to read the form data and binds the web request parameter to a specific controller method.

(5) @PathVariable Annotation :-

This annotation is used to extract the data from the URL path. It binds the URL template path variable with method variable.

(6) @RequestBody Annotation :-

This annotation is used to convert HTTP requests from incoming JSON format to domain objects directly from request body. Here method parameters binds with the body of the HTTP request.

(7) @ResponseBody Annotation :-

This annotation is used to convert the domain object into HTTP request in the form of JSON or any other text. Here, the return Type of the method binds with the HTTP respone body.

8. @ModelAttribute Annotation :-

This annotation refers to model object in Spring MVC. It can be used on methods or method arguments as well.

JSON full form ?

[JAVAScript Object Notation]

⇒ JNSON Stands for JavaScript Object Notation, used to transfer data between client and server.

Jackson (library) web, dev Tool (Dependency)
↓
JSON

DATE
PAGE

★ Product → /product

- → Create → Post
- → Get →
  /{id} → GET
- → delete → /{id} → DELETE
- → list →
  / → GET

@Controller.
@PostMapping("/product")
Public String save(){
Sys.o.p("Product saved--");
return "product saved";
}

TOC isko view name na samjhe iske liye Hum, is method par. @RequestBody annotation lga denge, taki vo isko as a respone hi lega.

Why we use postMan?
for testing the APIs. (Popular API testing tool used to:-
- Send HTTP requests, • Test REST APIs, • Check response from server.
- form-data.
- It helps developers test APIs without frontend.
- x-www-form-urlencoded.
- raw

key = " "
key must be in String form

{@RequestBody Product p}

Jo Jsion se data h, or-
Jo hum save karwa rahe hai To vo same Hona chahiye

@RequestParam("name") String name.
Use, when we send the data in key-value pair

★ Uses of Postman

Postman is used to :-

(1) Test GET, POST, PUT, DELETE APIs.
(2) Send request data (body, headers, params).
(3) Check JSON / XML response.
(4) Debug backend issues
(5) Automate API testing

⇒ Spring Boot API ko directly test karna before frontend. ready.

★ What happens in Postman ?
Postman acts as a client, and your backend (Spring Boot) is the server

★ Body Types in PostMan (important)
when we send data using POST / PUT, we choose body type.

(1) form-data :-

- Sends data as key-value pairs
- Used when sending files + text

⇒ where used ?
- file upload, - Image, PDFs, - Bonile Picture upload

(2) x-www-form-urlencoded :-

- Sends data in URL encoded format
- Same as HTML form submission

Ex:- username = neha & password = 1234

where used ?

- Login forms, • Simple form submission,
- No file upload.

(3) Raw :-

Sends raw data in Request body
mostly used for JSON

Example :- (JSON)

```
{
    "name" : "Neha",
    "email" : "neha@gmail.com"
}
```

where used ?

- REST APIs, • SpringBoot + @RequestBody,
- Microservices communication.

```
@PostMapping ("/save")
public String saveUser (@RequestBody User user){
    return "saved";
}
```

form-data vs x-www-form-urlencoded vs raw

Features	form-data	x-www-form-urlencoded	raw
Data format	Key-value	URL encoded	JSON/XML/Text
file upload	✓ Yes	✗ No	✗ No.
used for	files + text	Simple forms	REST APIs.
Spring annotation	@RequestParam	@RequestParam	@RequestBody.

① Postman :-
Postman is an API testing tool used to send HTTP requests and validate responses.

② form-data :-
used to send form fields along with files.

③ x-www-form-urlencoded :-
used to send simple key-value form data.

④ raw :-
Used to send raw data like JSON in REST APIs

Database ke ander jo table hai = Resource
like Product, Category

DATE
PAGE

PUT	PATCH.
Completely update the resource.	when, we want to update partially resource.

Product — Primarykey (must be unique)

Task	Method	Route	Parameter
Create	POST	/product	{ "name" : " " "price" : " " }
		← Path variable	
fetch.	GET	/product/{id} ← Dynamic route	id.
DELETE	DELETE	/product/{id}	id.
update	PUT / PATCH	/product/{id}	id + { = } =
fetchAll	GET	/product	

DELETE function, me return-type jisko delete karte hai wahi rakh dete hai, Taki UNDO ki functionality de sake (like Product, Category etc).

HTTP Status Code → 200, 201, 204, 404, 500 } ResponseEntity (class)

@ControllerAdvice.

HTTP Status Code :-

An HTTP Status Code is a number sent by the Server in response to a client request (browser, frontend app, Postman).
It tells :-

- whether the request was successfull.
- or why if failed.

Status Code Categories :-

Range	Meaning
2XX	Sucessfull
4XX	Client Error
5XX	Server Error

(A) 2XX - Success Code

(1) **200 - Ok**

- Request processed successfully
- Data is returned

Use when :-

- fetching data
- Updating data successfully.

(2) 201 → Created

- A new resource is created successfully

Use when :-

- Creating a new record (register, user, save data)

(3) 204 → No Content

- Request Successfull
- No response body

use when :-

- Deleting the resource

(B) 4XX :- Client Error Codes

(1) 400 - Bad Request

- client sent invalid data

Example :-
- Missing required fields
- Invalid JSON Data.

(2) 401 - Unauthorized

- Authentication required or token missing/ Invalid.

Example :-
- JWT token not provided

(3) 403 - forbidden

- User is authenticated but doesn't have Permission

Ex :- USER role trying to access ADMIN APIs

(4) 404 - Not found

- Requested resource or URL doesn't exist.

(C) 5xx - Server Error Codes

(1) 500 - Internal Server Error

- Something went wrong on the Server

Ex:- NullPointer Exception, Database error

(2) 502 - Bad Gateway

- Gateway / Server received invalid response.

Ex:- API Gateway → Microservice down.

(3) 503 - Service Unavailable

- Server is temporarily unavailable

Ex:- Server overloaded
Maintenance mode.

In Constructor Injection :- two constructors are not recommanded, just, increase the parameters.

SpringBoot automatically throw 500 error. (like SQL Server error, DB related sever error.)

for Handling this, we create one global Exception Handler class (instead of writting try-catch).

To tell Spring-container that this class is global Exception Handler class, we use.

⇒ @RestControllerAdvice

⇒ On method, which is Handle the Exception, we use

⇒ @ExceptionHandler

@ExceptionHandler(Exception.class)

But jo property seardalize nhi hogi, To vo persist bhi nhi Hogi.

DATE
PAGE

Solution :- JSON

~~To~~ To Not Expose the properties of n objects

Three ways to do this :-

① @JSON Ignore
② @ JSON Ignore property } NOT recomandded (Bcoz, property not seardalize and not persist in DB.

③ DTO (Data Transfer Object is used to transfer data between layers) (Recommanded)

④ JSONIgnore, @JSON Ignore Property
property-level, Class-level.

3) DTO :-

⇒ we create a separate class for DTO, And sends the DTO in respone, in any controller Method.

⇒ Mapping the entity parallely entity itself.

Spring Security Module :-

Spring auto configure the end-points in our Application, when we add spring Security in pom.xml (class path)

@EnableWebSecurity :- Enables Spring Security.

It is powerful model module use to secure Spring Boot Applications by Handling :-

- Authentication, Authorization, Protection against CSRF, CORS, etc.

Default - behaviour :-

It secure all the end-points By default

Backbone of Spring Security = filter

~~SecurityChainFilter~~ # SecurityFilterChain :-

we create SecurityChainFilter. method than it will returns object of SecurityChainFilter.

Hum iska object create karne ki responsibility to container than we use @Bean.

CSRF :-

CROSS-Side request Forgery :-
(for browser environment)

for preventing the unnecessary authentication, authorization. (

```
http.CSRF(CSRF->CSRF.disable()).authorizeHttpRequest(auth->auth.requestMatchers("/**").permitAll());
```

Sari APIs request ko allow kar diya Hai

- Disable CSRF (for REST APIs) :-

In Rest APIs (JWT based), CSRF is usually disabled. because APIs are Stateless :-

⇒ `http.csrf().disable();`

⇒ Matches() returns Boolean.

DATE
PAGE

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity
        http) throw Exception{

    http.CSRF(CSRF -> CSRF.disable()).authorizeHttp
        Requests(auth -> auth.requestMatchers("/**")
        .permitAll());
    return Http.build();
}
```

- Definition:-

CSRF (Cross-Site Request Forgery) is a security attack where a malicious website tricks a logged-in user's browser into performing an unauthorized action on another website without the user's knowledge.

#How CSRF Attack Works :—

① user logs into a trusted website.
↓
② Browser stores session & cookies.
③ User visits a malicious website.
④ That site sends a hidden request to the trusted site.
⑤ Browser automatically sends cookie.
⑥ Server thinks request is valid → Attack succeeds.

CSRF Protection in Spring Security.

Spring Security protects against CSRF by default.

⇒How?

- Generates a CSRF Token, • Token must be sent with every-eh state-changing request (post, Put, DELETE)
- Server validates token before processing request.

#problem :- Recursive JSON Loop.

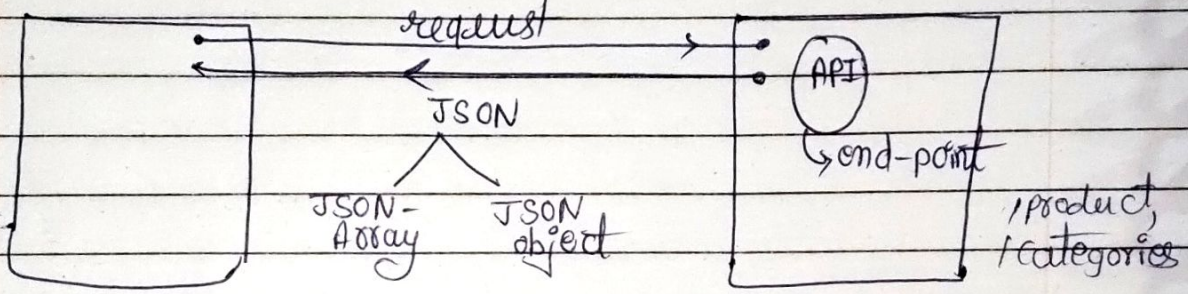
#Solution :- DTO, JSON ignore, JSONIgnoreProperties.

- Recommanded
- Risky.

Api route for pagination.

⇒ http://localhost:8080/api/product?page=0&size=5

JSON (JavaScript Object Notation)
Concept of JavaScript.



Why we create API ?

⇒ To provide the data to different users (like Browser client, IOS client, Android IOS).

⇒ API Developers (who writes develop).

JSON ⇒ Technology

⇒ Data format that Platform And ~~client~~ Independent

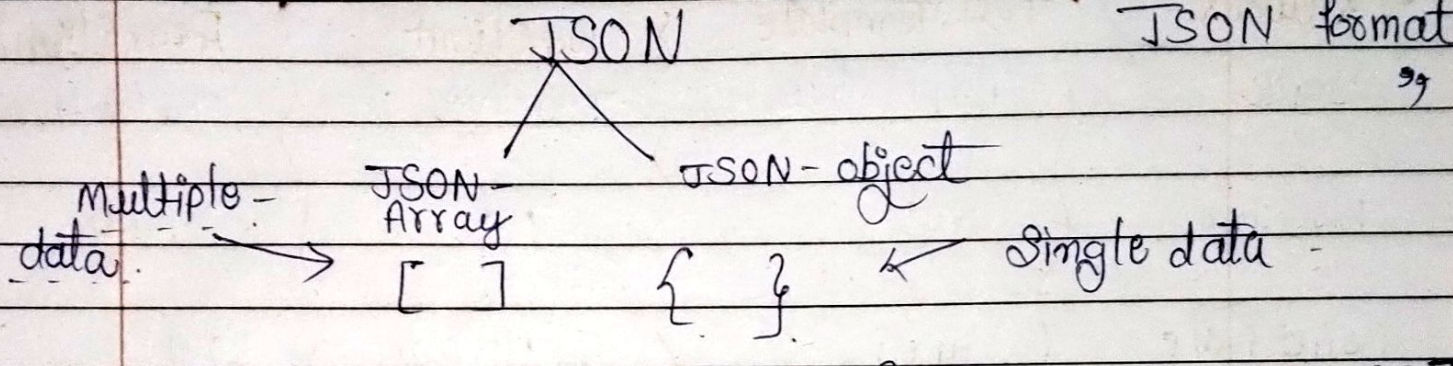
⇒ It is web-technique which is used to exchange the data b/w two parties.

⇒ It simply a text and string data.

⇒ JSON Data is easy to parse

we can also use XML

" API returns the reponse in JSON format "



[{ "id": 100, "title": - }, { }, { data: [{ }, { }] }]

API Type → Rest " It follows constraints "

→ SOAP

REST

R ⇒ Representation.

E ⇒

S ⇒ STATE

T ⇒ Transform

{ "id": 101,
"title": ...
"Stock": ... }

"States"

States changes according to time.

REST :—

Representational State Transfer

Online JSON Viewer :- To read the JSON

DATE
PAGE

RestTemplate / FeignClient / webClient
old & Blocking.

what is

Difference :-

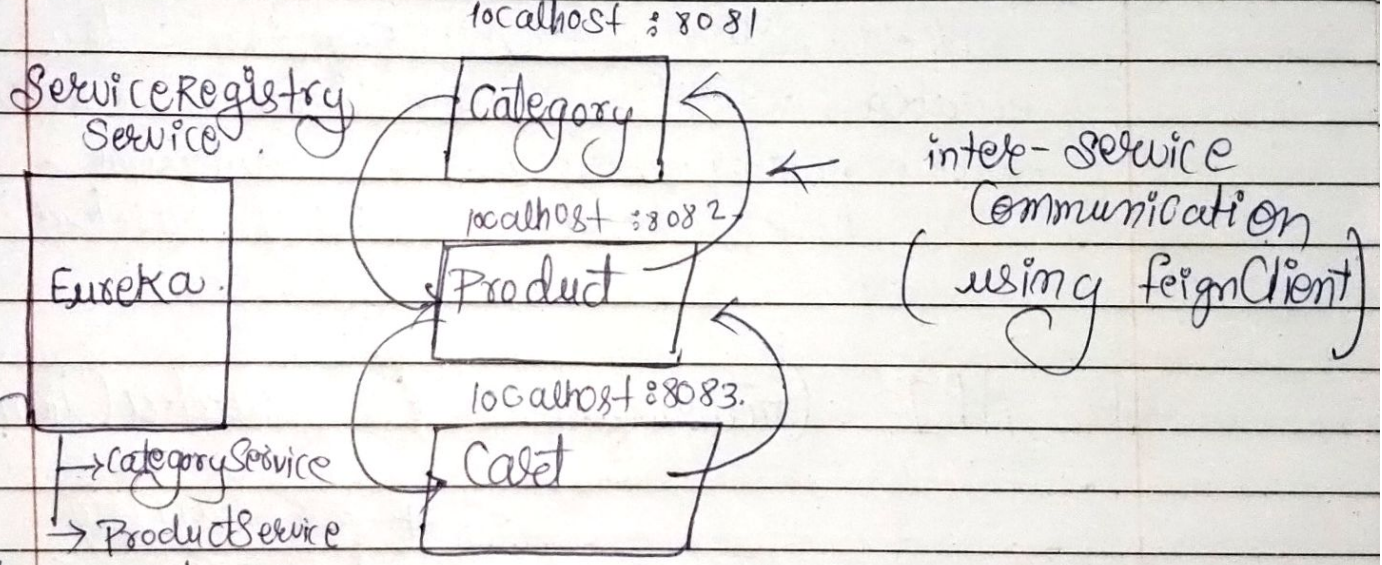
Industry-Standard

features	Rest Template	webClient	FeignClient
Style	Imperative	Reactive	Declarative
Blocking	yes.	NO	Depends.
Boilerplate	High	medium	LOW.
Microservices	X	Δ	Best
Eureka Support	X	X	Native (✓)
JSON Mapping	manual	manual	Automatic.
Learning Curve	Easy	medium	Easy.

Why we use FeignClient :-
If we want to call the API / third-Party API.

feignclient direct entity me convert nhi karta, Hume pehle dto banana pdega

★ @FeignClient(name = "category-feign-client", url = "https://dummyjson.com/")
public interface CategoryFeignClient{
@GetMapping("/products/categories")
List<object> getCategories();
}



↳ It automates loadbalancing

↳ It creates the multiple Instances of services.

By Default
Server.port = 8761

for Eureka Server Creation :—
⇒ cloud Bootstrap, eurekaServer,

⇒ Service ko name se call kar paye isliye eureka server banaya Hai, ⇒ register karke

add in microservices. ⇒ for making it eurekaClient

spring → add starters → eureka Discovery Client
Add Dependency

⇒ EurekaClient is By Default enable.
⇒ server.port = 8761
⇒ eureka-instance.hostname = localhost
⇒ eureka.client.register-with-eureka = false.
[Kahi or na register HO Jaye is liye]
⇒ eureka.client.fetch-registry = false.

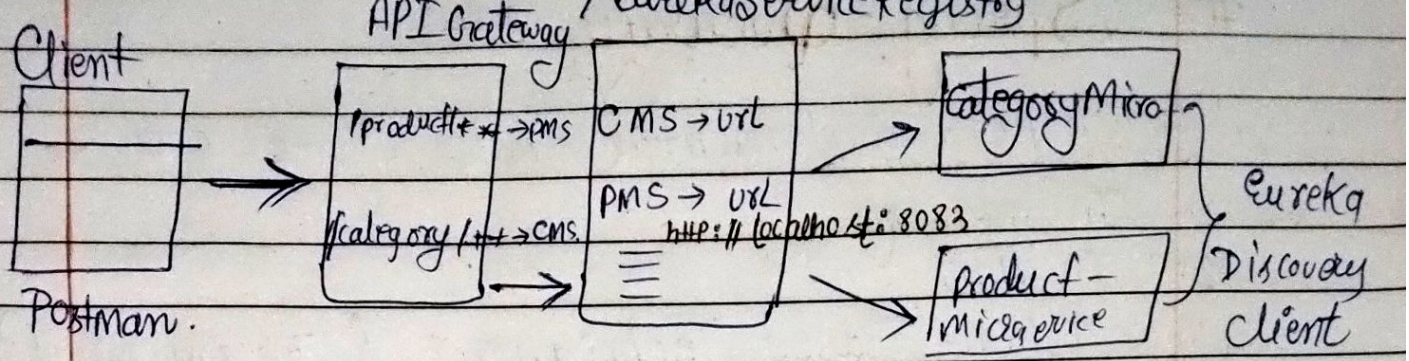
In MicroService
Property :—

eureka.instance.prefer-ip-address = true
eureka.client.fetch-registry = true.
eureka.client.register-with-eureka = true
eureka.client.service-url.defaultZone =
http://localhost:8761/eureka.

API Gateway (eurekaClient)

for running all microservices on a Single Port.
we will do only Configuration in API Gateway
we register this API Gateway in Service Registry Servic

Eureka Server.
/eurekaServiceRegistry



Request → /Product → /product/** → PMS → PMS-URL →
ProductMicroservice.

⇒ Dependencies....

1) Gateway × spring-cloud-starter-gateway
2) eurekaDiscoveryClient

⇒ API Gateway
~~eureka~~ Server. Port = 8080

@EnableDiscoveryClient

Spring.cloud.gateway.routes[0].predicates[0] = Path/products/**

spring.cloud.gateway.routes(0).filters[0] = RewritePath/products/(?.*)/${segment}

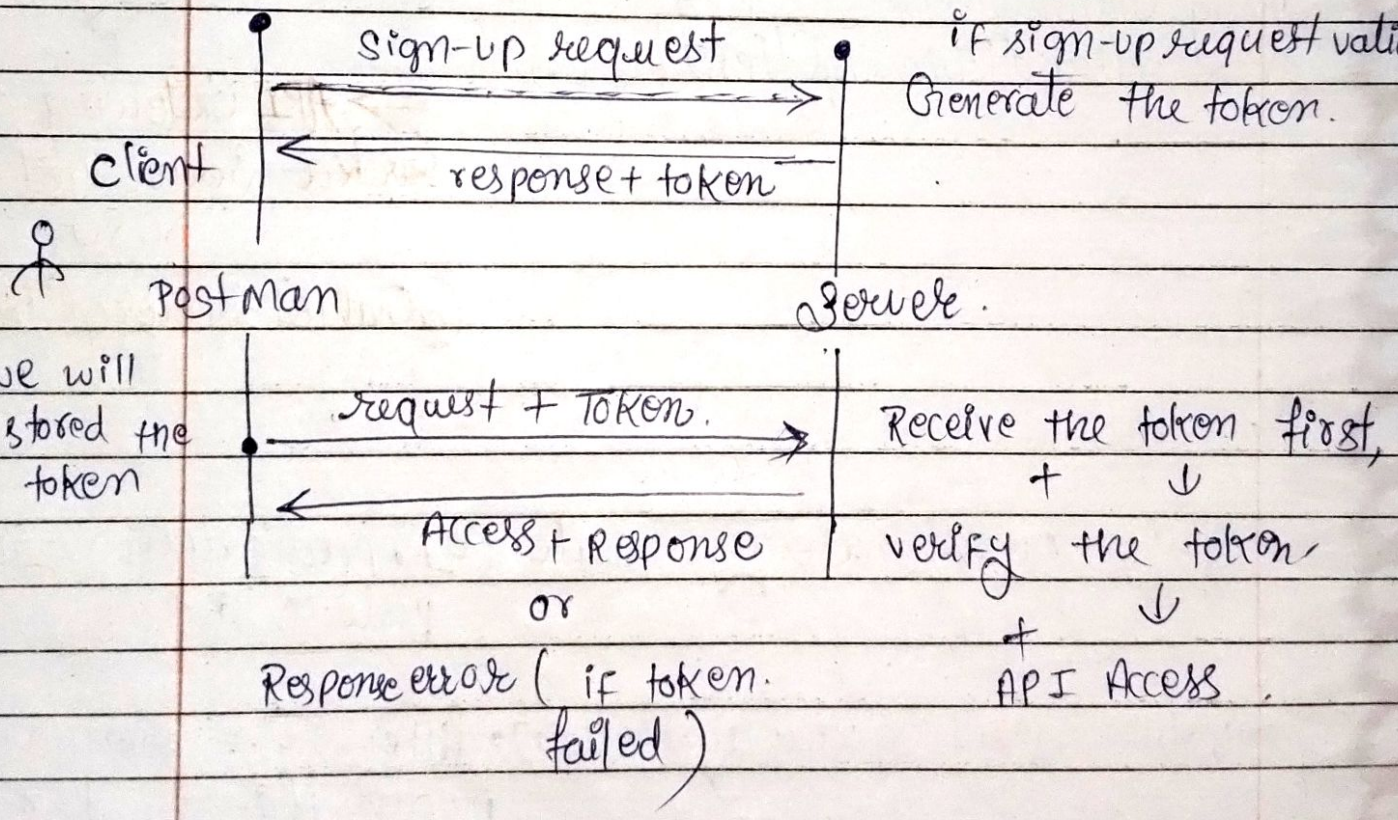
this line, converts the client requested URL into the controller URL,
agar humari microservice me ~~do~~ ya do se jyada Controller ho to.

Spring-Security :-

Spring-Security. Can be implemented into three ways.

(1) In memory
(2) In dao
(3) Token-Based Technique (Recomanded).

~~JSON~~ (~~JSON~~
:- JWT (JSON web Token)



format :-

format of JWT

XXXX-YYYY-ZZZZ.

↓ ↓ → Secret_key.

Header Payload / claim

(1) Header :-

- Algorithm [main Component of Header]

Symmetric Assymmetric.

Header Contains some important data like Algorithm

Symmetric Algorithm :-

Token works in → Generate, Verify.

when we generate the token, the Secret key and the verify key the key is same.

Same key for generation and Verification

⇒ HS256 (Symmetric Algorithm)

Use when, small-scale Application Development And Monolithical Architecture.

Algorithm.

(2) Assymetric ~~key~~ :-

RS256. (Popular Algorithm)

⇒ Different_key for Generation, verification.

⇒ Use when we use Microservices Architecture.

(2) Payload :-

Some information related to the Users.

It is Basically a Claim Object [data] we will mix it into our Token, So it will make our Token unique.

Bcoz, Header and Secret_key is same for all, so to make the Token unqie for everyone, we will mix payload in our token.

⇒ Payload may be email etc.

API Verification.

⇒ Token verification in API-Gateway

⇒ Token Generation in User-Microservices.

Filter Class (Apply / create in API-Gateway)

(1) GatewayFilter (2) GlobalFilter

we must do extra configuration and apply filter on separate and each API

It Automatically secure All the routes of API

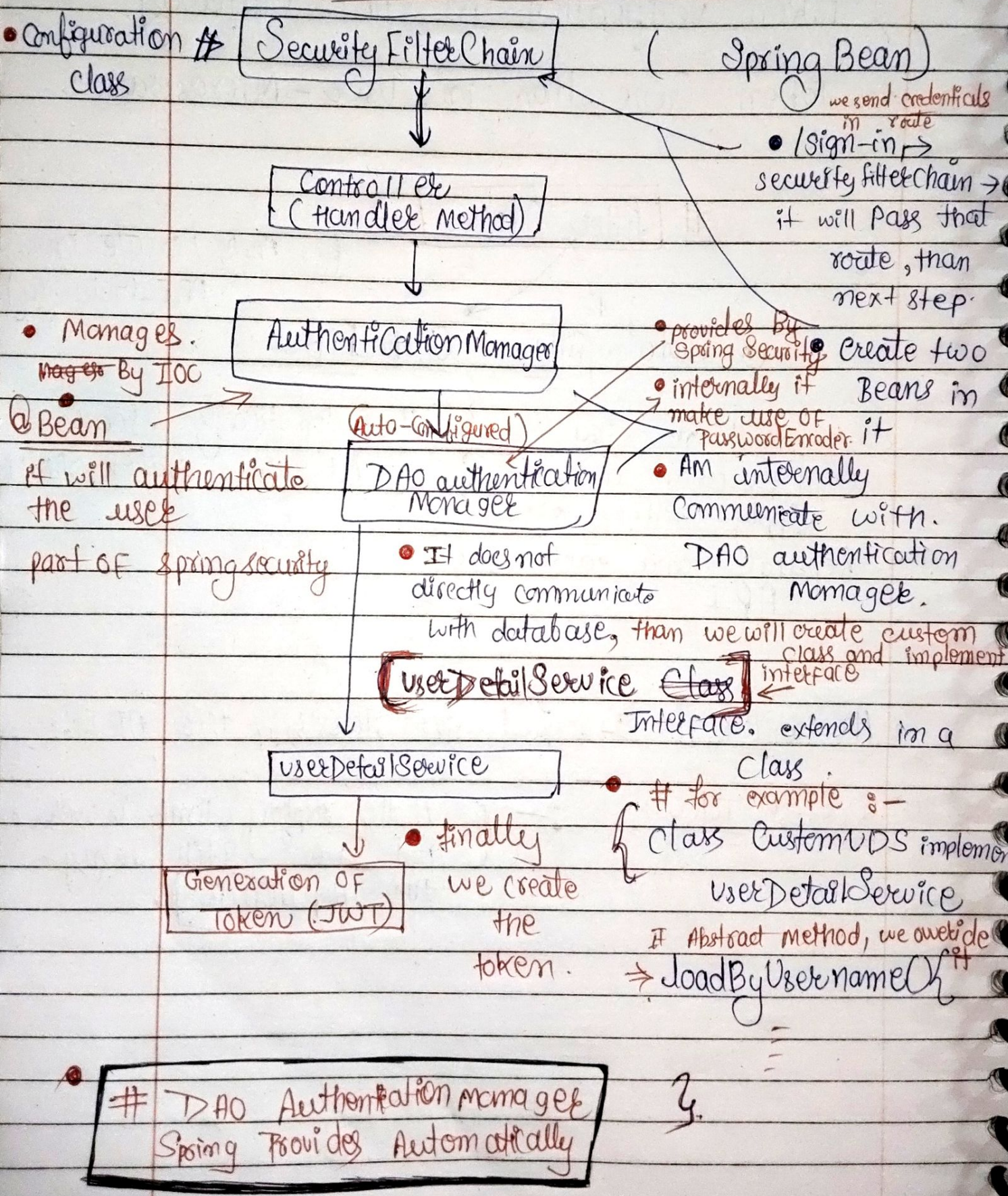
(1) Access Token :- Just for Accessing the APIs.

(2) Refresh Token :- (If the expiry time is over, But we still using the Application).

- ⇒ Credentials :- email, Password etc.
- ⇒ By Default All routes are Secured.

Flow of JW Token.

* Token Generation WorkFlow *



- # DAO Authentication manager Spring Provides Automatically

load Balancing ✓

DATE
PAGE

for the Vadi validation OF Token.

◉ Authorization → Bearer Token → than Paste the token.

⇒ Usee ke Corresponding, jo token Generate Ho raha tha, Jo vo kuch time ke liye hota hai, To vo us kuch time Hota, To us time ke exceed Hone ke bad new Token Generate Hoga.

Q. If the interviewer ask you that How you generate the token & valid the token.

⇒ than you have to explain the Generation and validation ~~Code~~ token workflow.