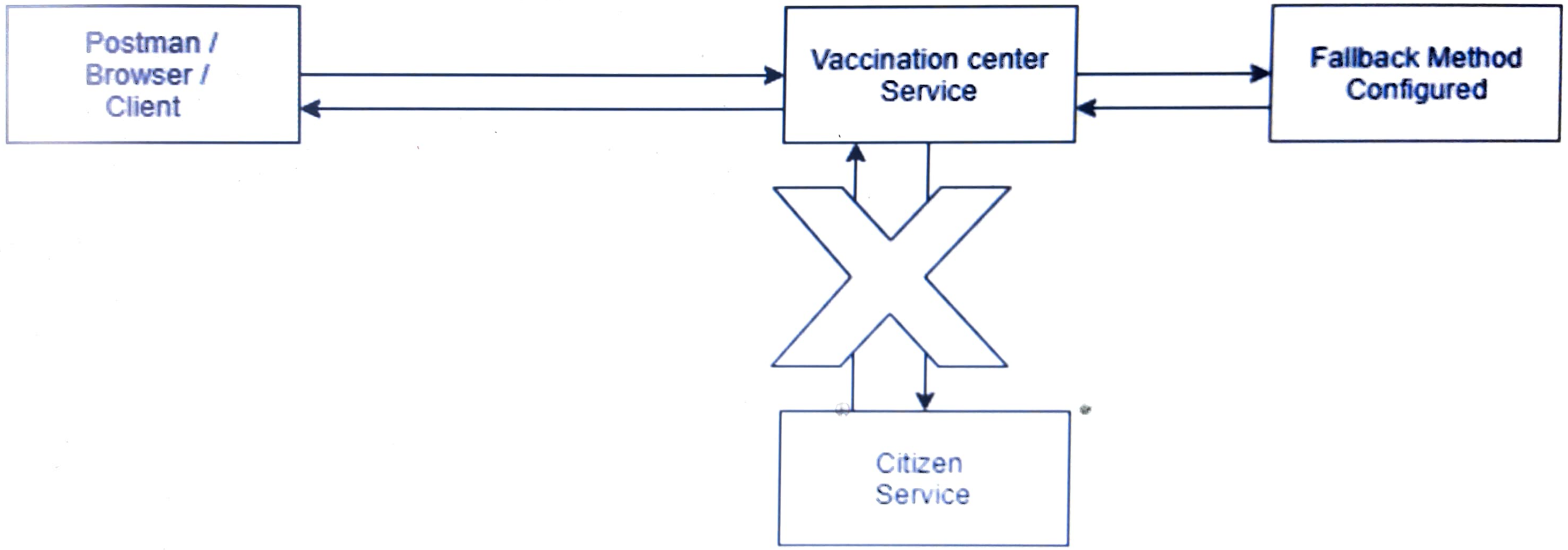
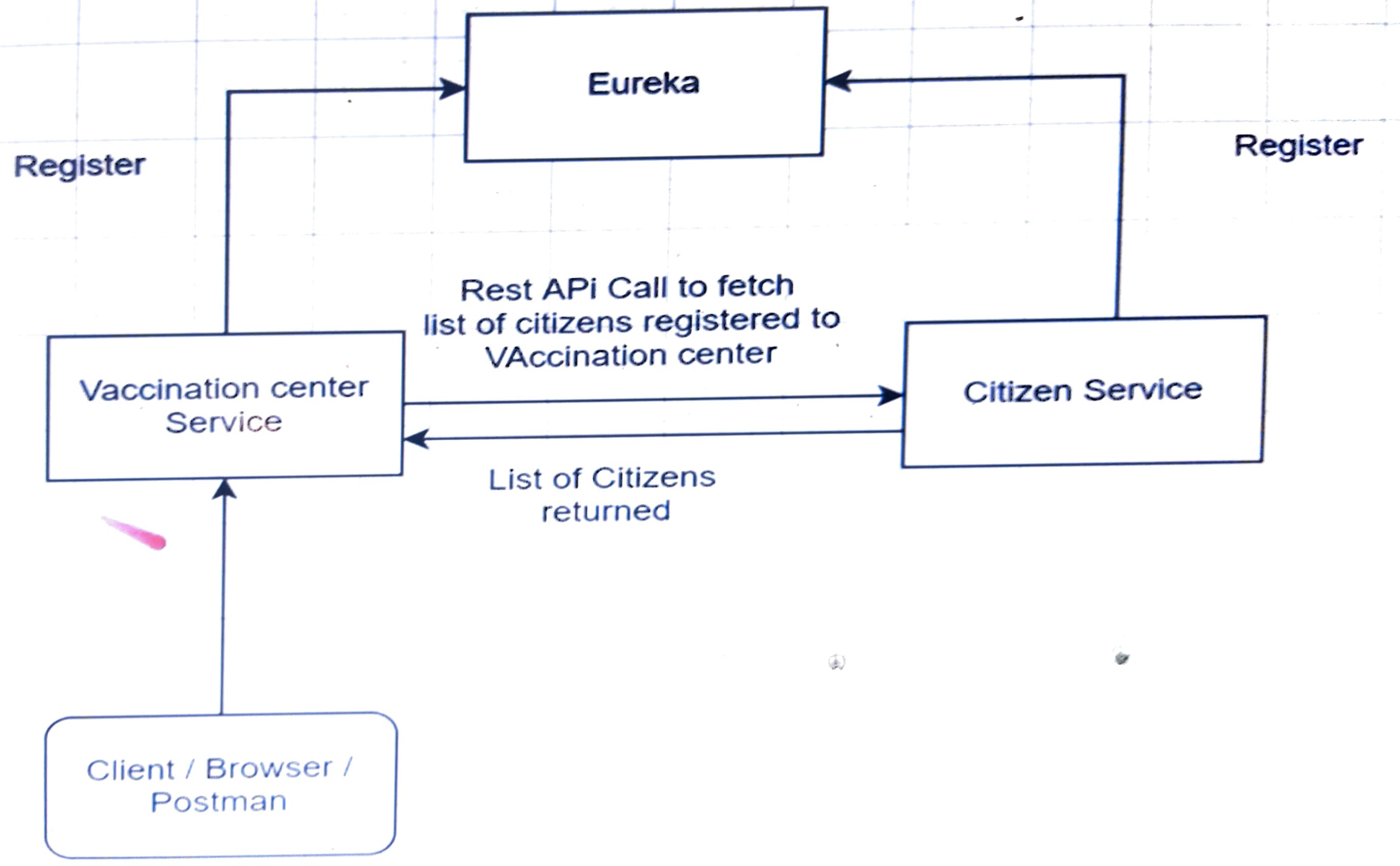
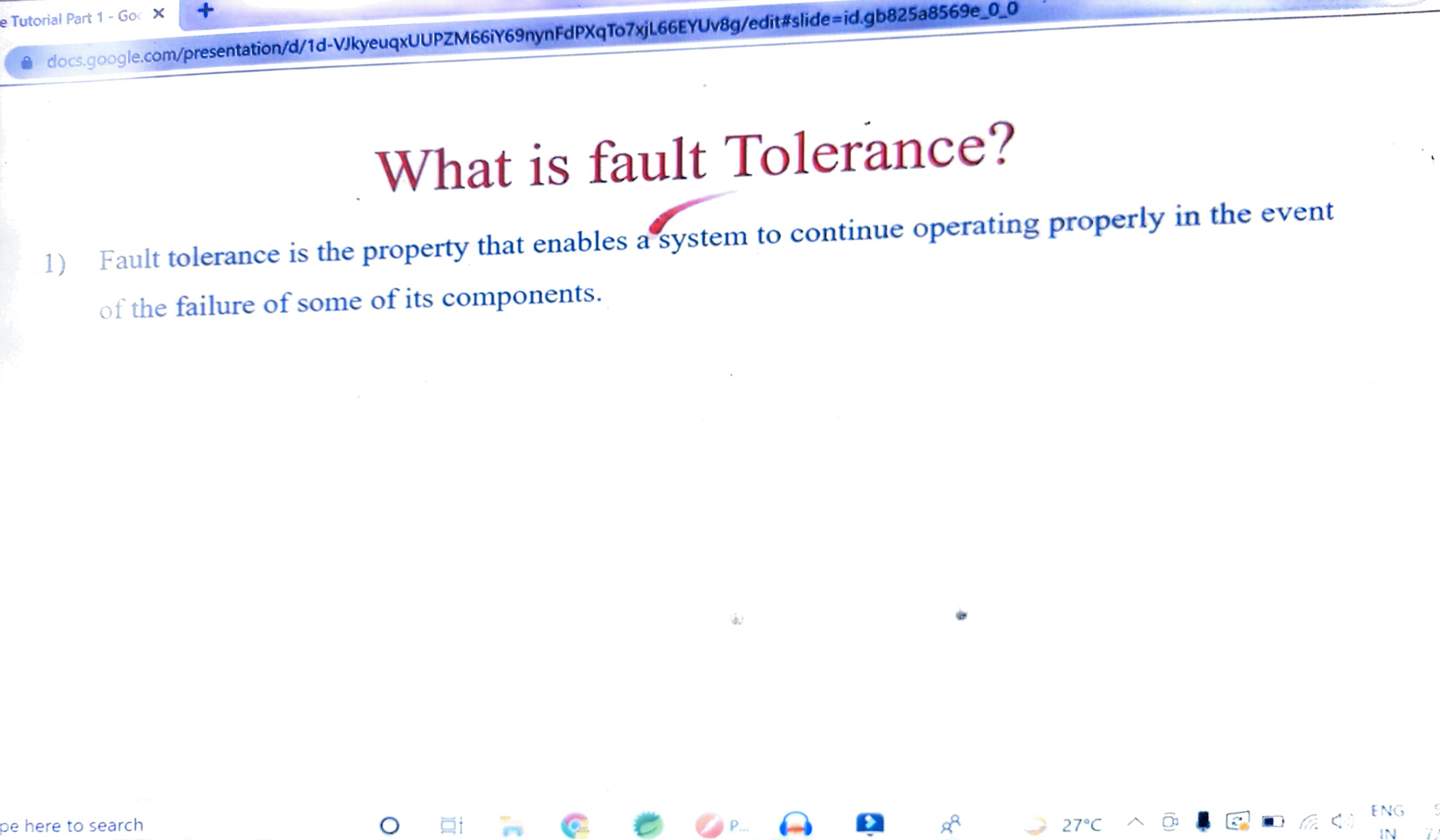
How to configure Circuit Breaker Design Pattern


Postman / Browser / Client
Vaccination center Service
Fallback Method Configured
Citizen Service

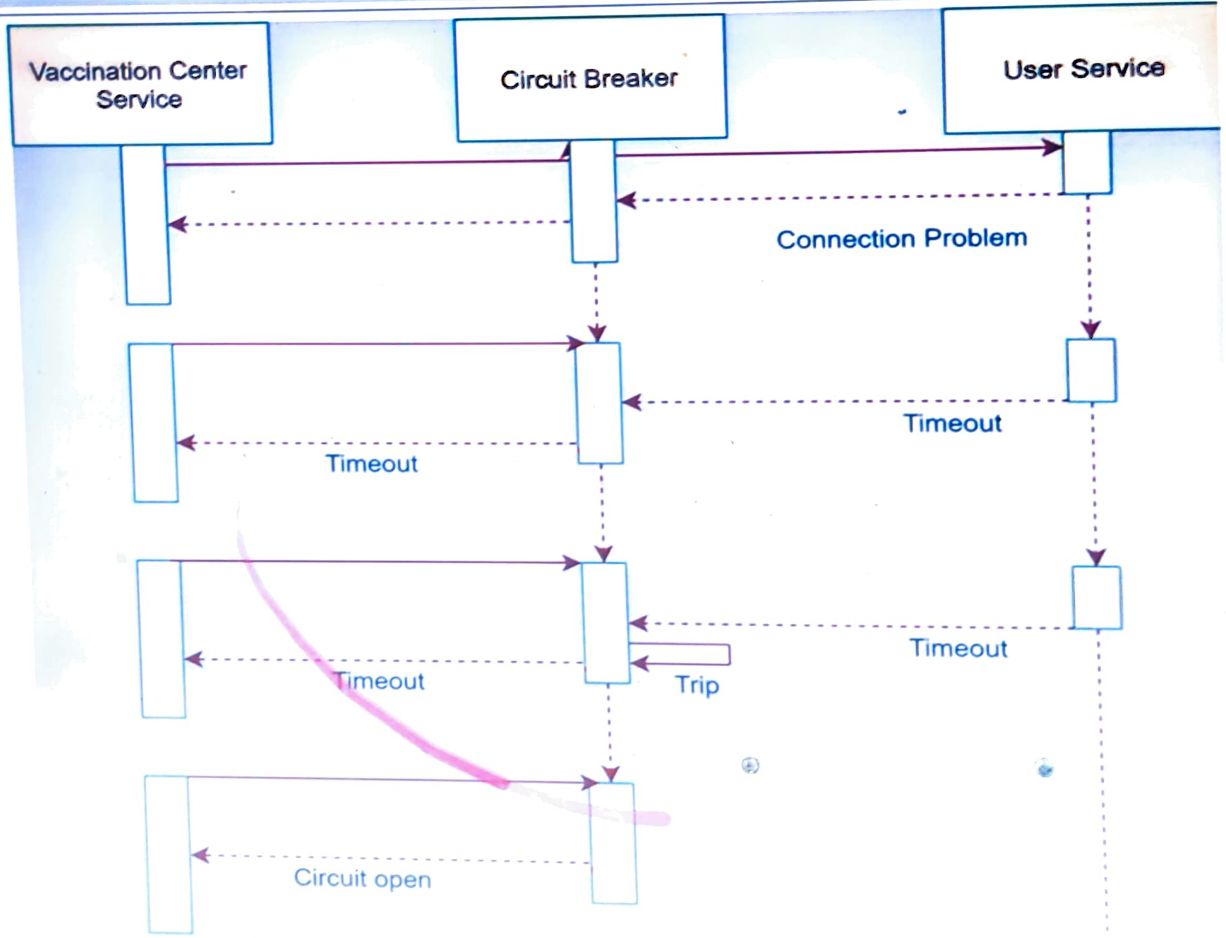

Eureka
Register
Register
Rest APi Call to fetch list of citizens registered to VAccination center
Vaccination center Service
Citizen Service
List of Citizens returned
Client / Browser / Postman


e Tutorial Part 1 - Goc
docs.google.com/presentation/d/1d-VJkyeuqxUUPZM66iY69nynFdPXqTo7xjL66EYUv8g/edit#slide=id.gb825a8569e_0_0
pe here to search
27°C
ENG
IN

What is fault Tolerance?

1) Fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components.

docs.google.com/presentation/d/1d-VJkyeuqxUUPZM66iY69nynFdPXqTo7xjL66EYUv8g/edit#slide=id.gb825a8569e_0_0



Type here to search

Circuit Breaker Design Pattern

1) Idea behind circuit breaker is :

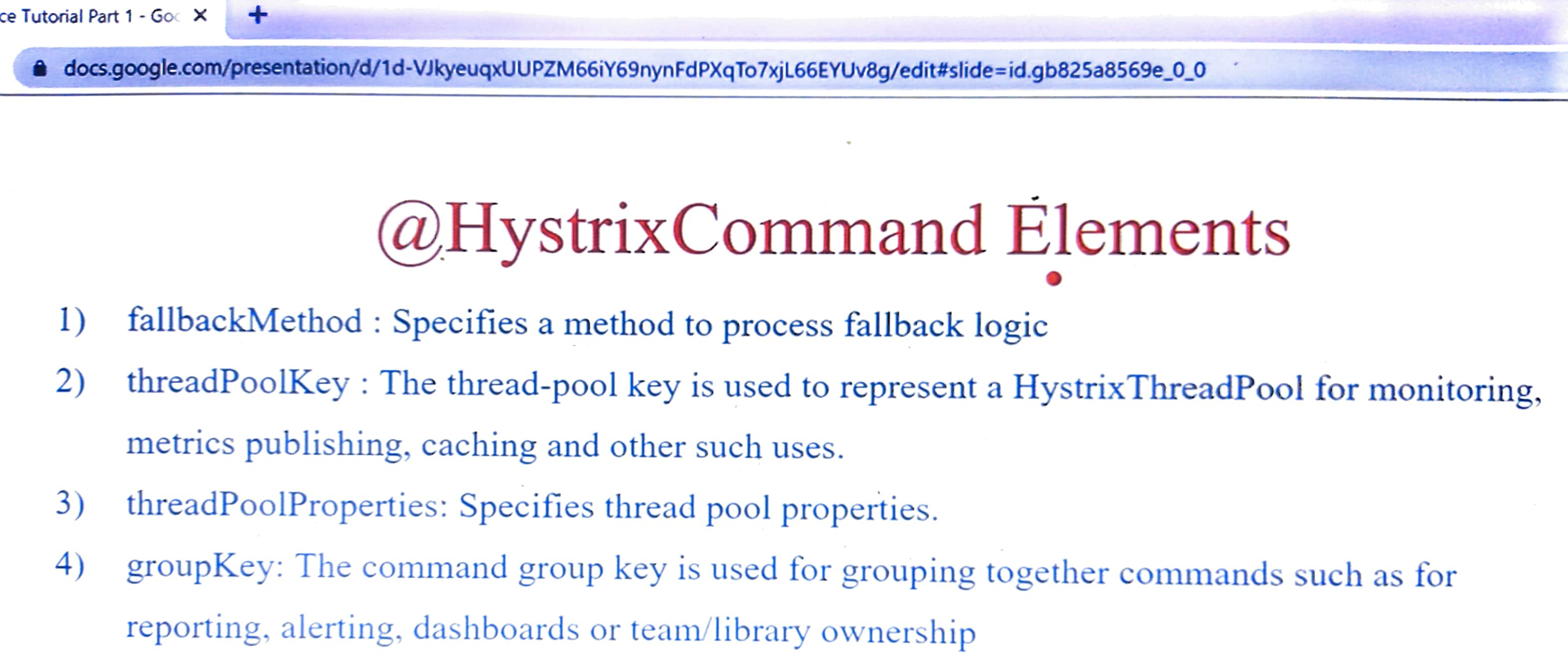
   a) Wrap your rest api call in circuit breaker object which monitors for failure

   b) Once the failures reach a certain threshold, the circuit breaker trips, and all further calls to the circuit breaker return with an error,

   c) Here comes our task: If it fails and circuit is open, configure a fallback method which will be executed as soon as circuit breaks or opens.

   d) In this way, your vaccination center service Rest controller is wrapped with a proxy class and monitor its calls. Everything is done internally handles everything for you.

pe here to search 27°C ENG IN



ce Tutorial Part 1 - Goc

docs.google.com/presentation/d/1d-VJkyeuqxUUPZM66iY69nynFdPXqTo7xjL66EYUv8g/edit#slide=id.gb825a8569e_0_0

@HystrixCommand Elements

1) fallbackMethod : Specifies a method to process fallback logic
2) threadPoolKey : The thread-pool key is used to represent a HystrixThreadPool for monitoring, metrics publishing, caching and other such uses.
3) threadPoolProperties: Specifies thread pool properties.
4) groupKey: The command group key is used for grouping together commands such as for reporting, alerting, dashboards or team/library ownership

@HystrixCommand Elements

5) commandProperties: Specifies command properties like

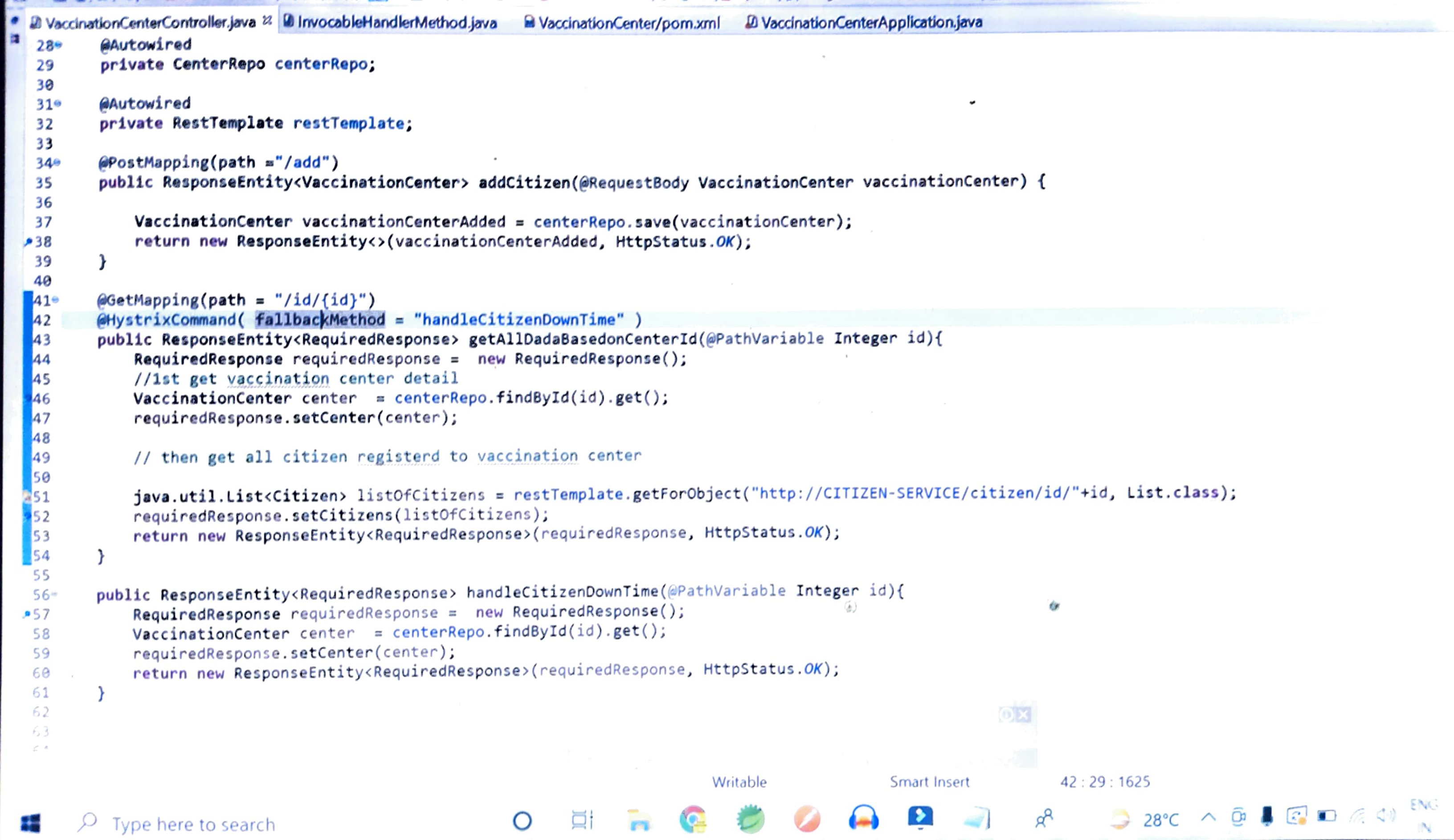
a) circuitBreaker.requestVolumeThreshold : number of requests reties before which cir
breaks

b) circuitBreaker.errorThresholdPrecentage eg 60%: If 60 % request fails out of total re
made then open the circuit

c) timeout s: Its like wait for milliseconds and if no response, break circuit

Steps we used

1) Add dependency spring cloud starter netflix hystrix.
2) @EnableCircuitBfeaker
3) @HystrixCommand to methods that needs breaking of circuit and configure it.On request mapped vaccination service method that calls user service , on that configure your Hystrix command.
    a) Then add fallback method to handle circuit breaks.i.e fallback method takes method name.
    b) Configure according to project needs


VaccinationCenterController.java
InvocableHandlerMethod.java
VaccinationCenter/pom.xml
VaccinationCenterApplication.java

```java
    @Autowired
    private CenterRepo centerRepo;

    @Autowired
    private RestTemplate restTemplate;

    @PostMapping(path ="/add")
    public ResponseEntity<VaccinationCenter> addCitizen(@RequestBody VaccinationCenter vaccinationCenter) {

        VaccinationCenter vaccinationCenterAdded = centerRepo.save(vaccinationCenter);
        return new ResponseEntity<>(vaccinationCenterAdded, HttpStatus.OK);
    }

    @GetMapping(path = "/id/{id}")
    @HystrixCommand( fallbackMethod = "handleCitizenDownTime" )
    public ResponseEntity<RequiredResponse> getAllDadaBasedonCenterId(@PathVariable Integer id){
        RequiredResponse requiredResponse =  new RequiredResponse();
        //1st get vaccination center detail
        VaccinationCenter center  = centerRepo.findById(id).get();
        requiredResponse.setCenter(center);

        // then get all citizen registerd to vaccination center

        java.util.List<Citizen> listOfCitizens = restTemplate.getForObject("http://CITIZEN-SERVICE/citizen/id/"+id, List.class);
        requiredResponse.setCitizens(listOfCitizens);
        return new ResponseEntity<RequiredResponse>(requiredResponse, HttpStatus.OK);
    }

    public ResponseEntity<RequiredResponse> handleCitizenDownTime(@PathVariable Integer id){
        RequiredResponse requiredResponse =  new RequiredResponse();
        VaccinationCenter center  = centerRepo.findById(id).get();
        requiredResponse.setCenter(center);
        return new ResponseEntity<RequiredResponse>(requiredResponse, HttpStatus.OK);
    }
```

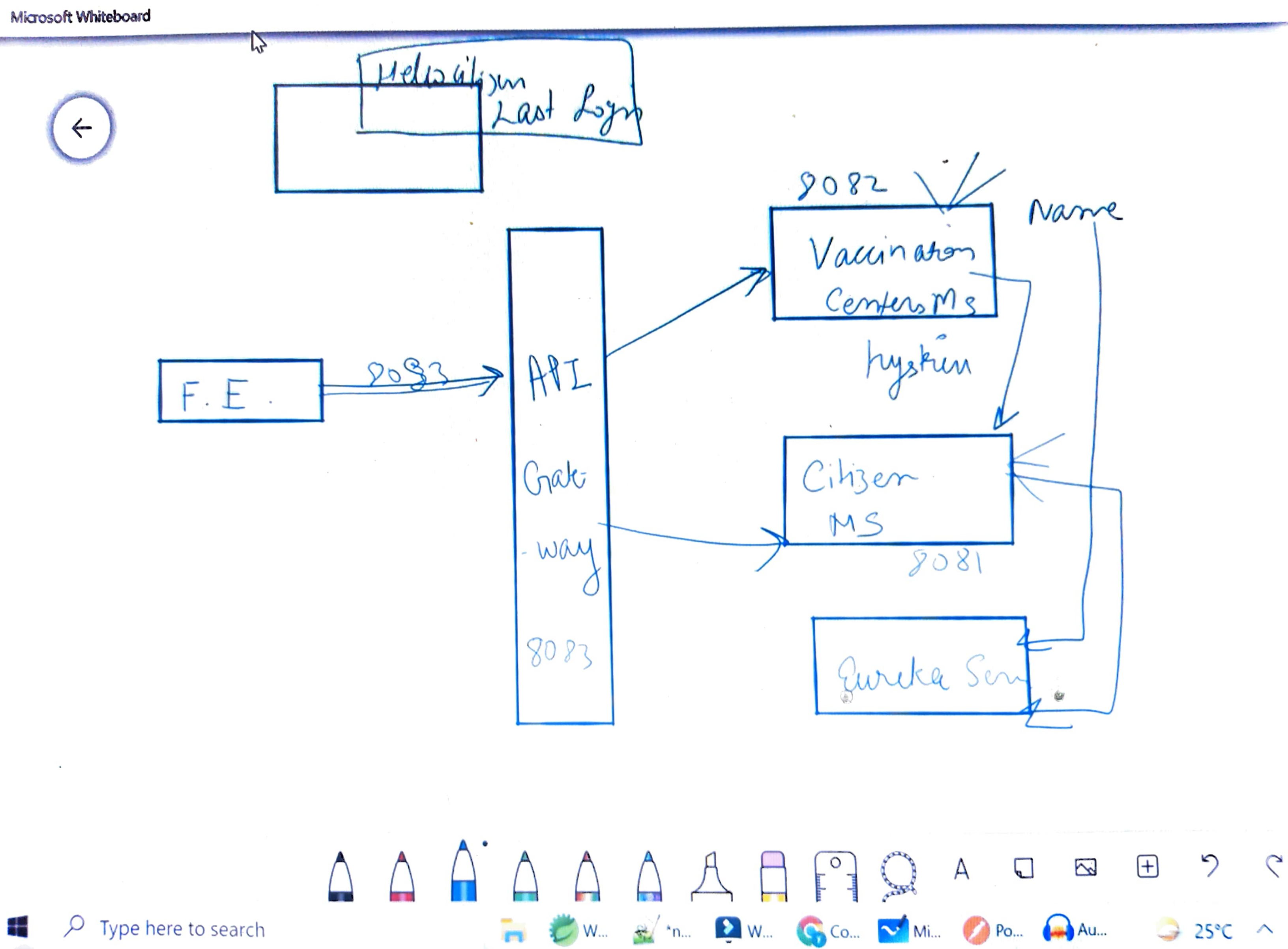
Why API Gateway

1) We had eureka server, hystrix implemented in Previous videos, Is that not enough ? Why do we need API Gateway?

2) The reasons are many :

   a) When we scale up our microservices, clients needs to now exactly which microservice they need. Client/ FE Dont need such information that backend developer is developing which microservice for which functionality. To over come this we have API Gateway to create an abstraction layer between FE and Backend developers . Thus FE developers need not depend on Backend people anymore. This is as good as implementing Abstraction principle in Java OOps.

   b) It becomes harder to change microservice without affecting the client. In reality, the client doesn't need to know microservice and its implementation behind.


Microsoft Whiteboard
Hello Citizen
Last Login
F.E.
8083
API Gateway
8083
8082
Vaccination Centers MS
Name
hystrix
Citizen MS
8081
Eureka Server
Type here to search
25°C

a) With API gateway we can centralise common functionalities like authentication, logging, and authorization. Now API will authenticate all requests hence ~~each MS~~ need not implement the same. Thus reduces code redundancy.

b) Multiple Versions of Service : Our MS is updated with the new changes, but the updated service is not compatible with the mobile client. Our mobile client will continue using version-V1 while other clients will move to the version-V2. This can be done using API Gateway Routing Technique

2) To address these issues, the architecture now contains another layer between the client and the microservices. This is API Gateway. API Gateway acts like a proxy that routes the request to the appropriate microservices and returns a response to the client. Microservices can also interact with each other through this Gateway.

Different Implementations of API Gateways Available?

1) There are a number of API Gateways available and one can use any of these based on the needs.
   a) Netflix API Gateway (Zuul)
   b) Amazon API Gateway
   c) Mulesoft
   d) Kong API Gateway
   e) Azure API Gateway

In Real world you will see most of the projects using Netflix API Gateway (Zuul), But we will be implementing API Gateway using Spring Cloud Gateway. Why??

here to search W... Co... Mi... Po... Au... 25°C ENG

Why Netflix Era came to an end and we all have to move on to Spring Cloud technologies?

Whenever if speak about MS, the first thing came to our mind was NETFLIX OSS support tools like Eureka, Zuul, Hystrix etc. But recently we see that many of these are now in maintenance mode. This means that there won't be any new features added to these modules, and the Spring Cloud team will perform only some bug fixes and fix security issues. The maintenance mode does not include the Eureka module, which is still supported.

Hystrix has been already superseded by the new solution for telemetry called Atlas.

The successor of Spring Cloud Netflix Zuul is Spring Cloud Gateway.

here to search W... *n... W... Co... Mi... Po... Au... 25°C

Why Not Zuul? Why Spring Cloud Gateway.

Zuul is a blocking API. A blocking gateway api makes use of as many threads as the number of incoming requests. So this approach is more resource intensive. If no threads are available to process incoming request then the request has to wait in queue.

Spring Cloud Gateway is a non blocking API. When using non blocking API, a thread is always available to process the incoming request. These request are then processed asynchronously in the background and once completed the response is returned. So no incoming request never gets blocked when using Spring Cloud Gateway.

ype here to search 25°C ENG IN

How it Works - Spring Cloud Gateway.

Clients make requests to Spring Cloud Gateway. If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler. This handler runs the request through a filter chain that is specific to the request. The reason the filters are divided by the dotted line is that filters can run logic both before and after the proxy request is sent. All "pre" filter logic is executed. Then the proxy request is made. After the proxy request is made, the "post" filter logic is run.

URIs defined in routes without a port get default port values of 80 and 443 for the HTTP and HTTPS URIs, respectively.

ere to search W... *n... W... Co... Mi... Po... Au... 23°C

Getting Started with Spring Cloud Gateway.

To include Spring Cloud Gateway in your project, use the starter with a group ID of org.springframework.cloud and an artifact ID of spring-cloud-starter-gateway.

Also you need to make it eureka client

If you include the starter, but you do not want the gateway to be enabled, set spring.cloud.gateway.enabled=false.

pe here to search W... *n... W... Co... Mi... Po... Au... 23°C