TASK 1

Contents

[Goal 3](#_Toc204637897)

[Approach 4](#_Toc204637898)

[Implementation 6](#_Toc204637899)

[Testing 7](#_Toc204637900)

[All services 7](#_Toc204637901)

[Eureka dashboard 7](#_Toc204637902)

[Testing - Service 2: hello service 8](#_Toc204637903)

[Logging and Tracing 9](#_Toc204637904)

[Testing - Service 3: name service 10](#_Toc204637905)

[Logging and Tracing 10](#_Toc204637906)

[Testing - Service 1: main service 11](#_Toc204637907)

[GET /status - Get Main Service Status. 11](#_Toc204637908)

[POST /main - Final response from hello and name service combined. 12](#_Toc204637909)

[Negative Scenario Testing 15](#_Toc204637910)

[Service 2: hello service – is down during calling Service 1: main service call 15](#_Toc204637911)

[Service 3: name service – is down during calling Service 1: main service call 17](#_Toc204637912)

[Name tag missing from Input Request Json (All service UP) 19](#_Toc204637913)

# Goal

Create a three service microservice project to demonstrate orchestration, rest api invocation, error handling, tracing of logs in javaEE or Node.js Services should communicate with aws hosted URLs

1st Service) Expose two http methods, one get and one post (add swaggerUI).

From the get method return “Up” if service is up. The post method should return the concatenated responses of the Get call of Service 2 and the Post call of Service 3 using the same payload({The json})

2nd Service) It contains one get method which is called by the first service to fetch a string”Hello” wrapped with a spring response entity.

3rd Service) This exposes one post method which is called by first service to print/log the passed json and return the concatenated name elements as a string (example - “John Doe”)

Print logs before each method call with a traceID to trace the call flow.

The json.

{

“Name”: “John”,

“Surname”:”Doe”

}

Concatenated Response :-

{

“Hello John Doe”

}

Extra: Handle exception when passed Json in post calls is not valid

# Approach

**Total 3 services:**

* Service 1: main service
  + GET /status - Get Main Service Status.
  + POST /main - Final response from hello and name service combined.
* Service 2: hello service
  + GET /hello – return “Hello”
* Service 3: name service
  + POST /concatenate - concatenate name and surname.

**For inter service communication** 🡺 service discovery

Service 0: eureka-server

* Main, hello, name service to implement eureka client.

This will avoid hardcoding of inter service URLs. Client services can get URL dynamically from eureka-server and call it.

**Print logs before each method call with a traceID to trace the call flow.**

Log before each method call 🡺 Use AOP Interception

* Create annotation LogMethodParam
* Wherever @LogMethodParam annotation is used, before method call, logs will be printed.

**For TraceId:**

* For distributed tracing – “Micrometer-Tracing”
* TraceId to be logged in into interceptor method into @LogMethodParam annotation implementation.

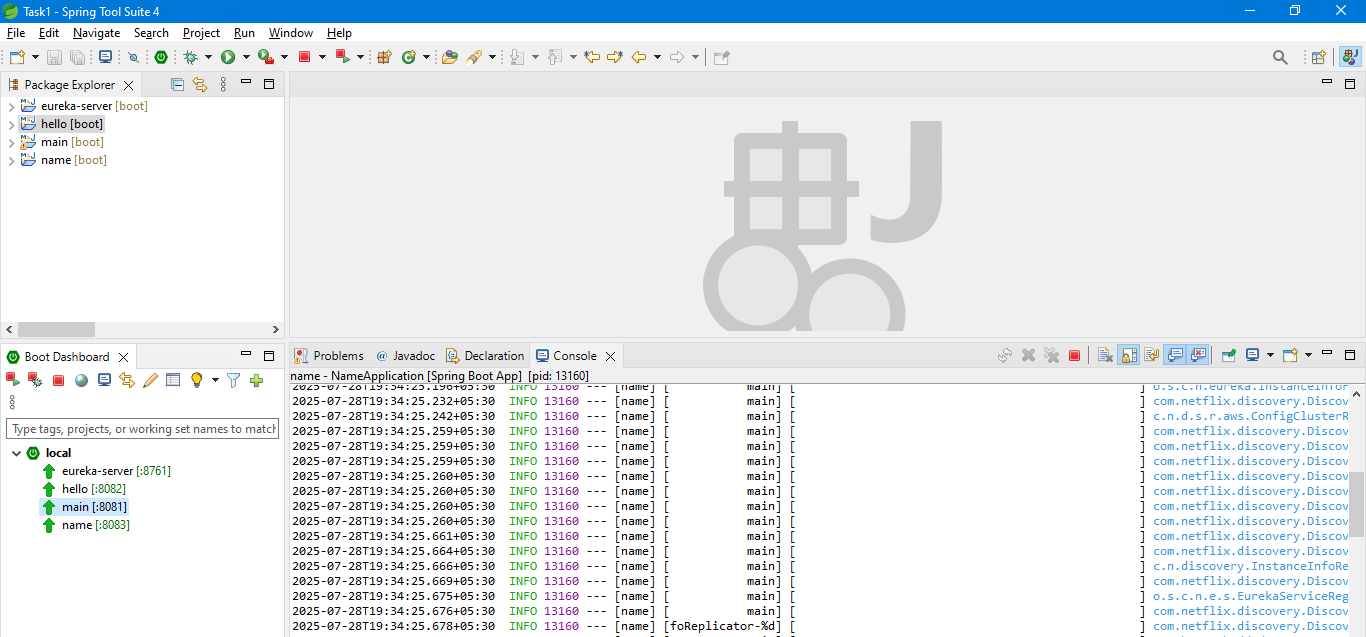
(Another approach could be to generate TraceId and keep placeholder for TraceId in header. But it is manual and less robust. So, Micrometer-Tracing is used.)

# Implementation

<https://github.com/sudhanshu-y/microservice/tree/main/Task1>

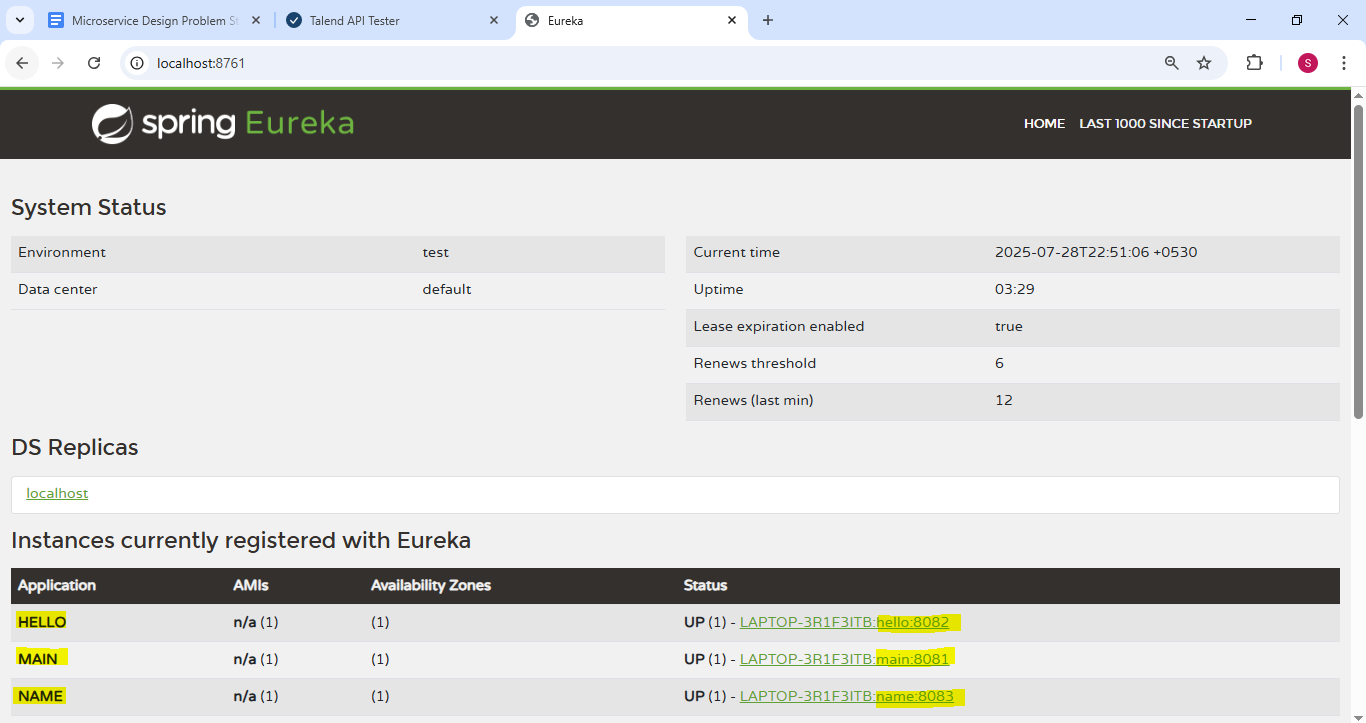
# Testing

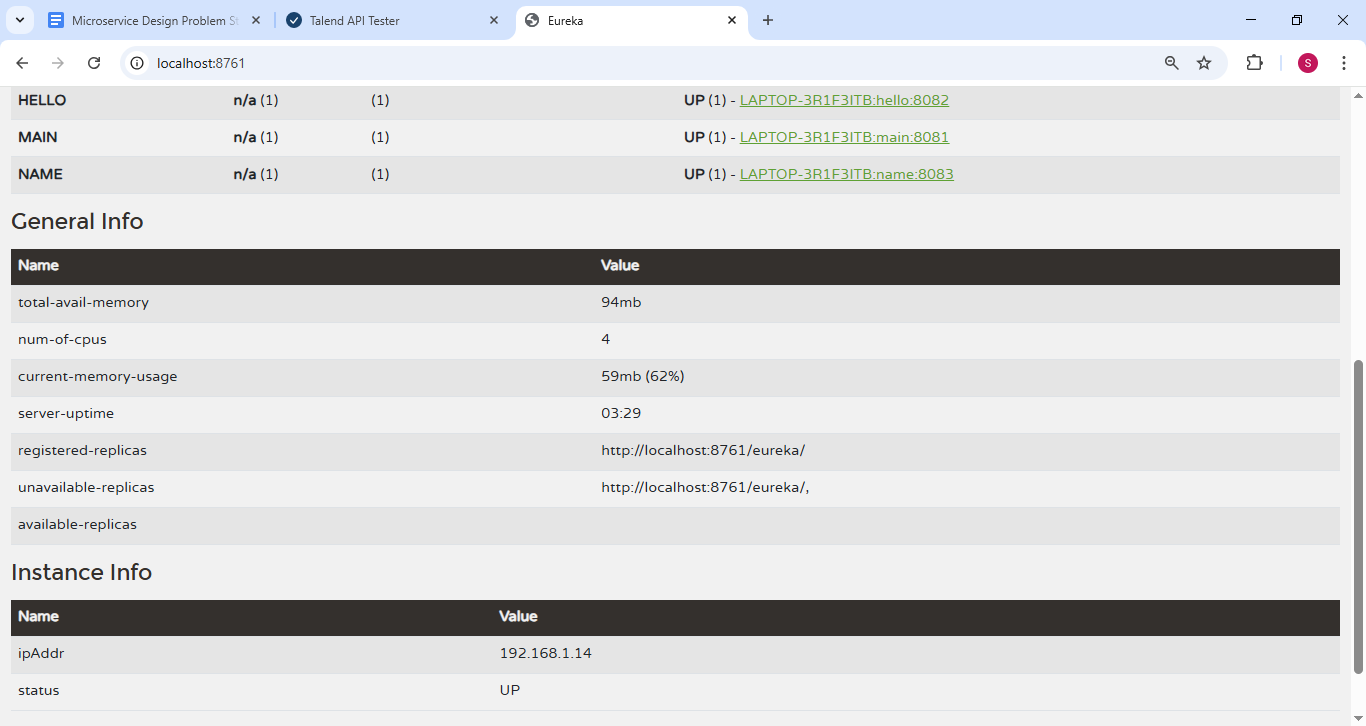
## All services



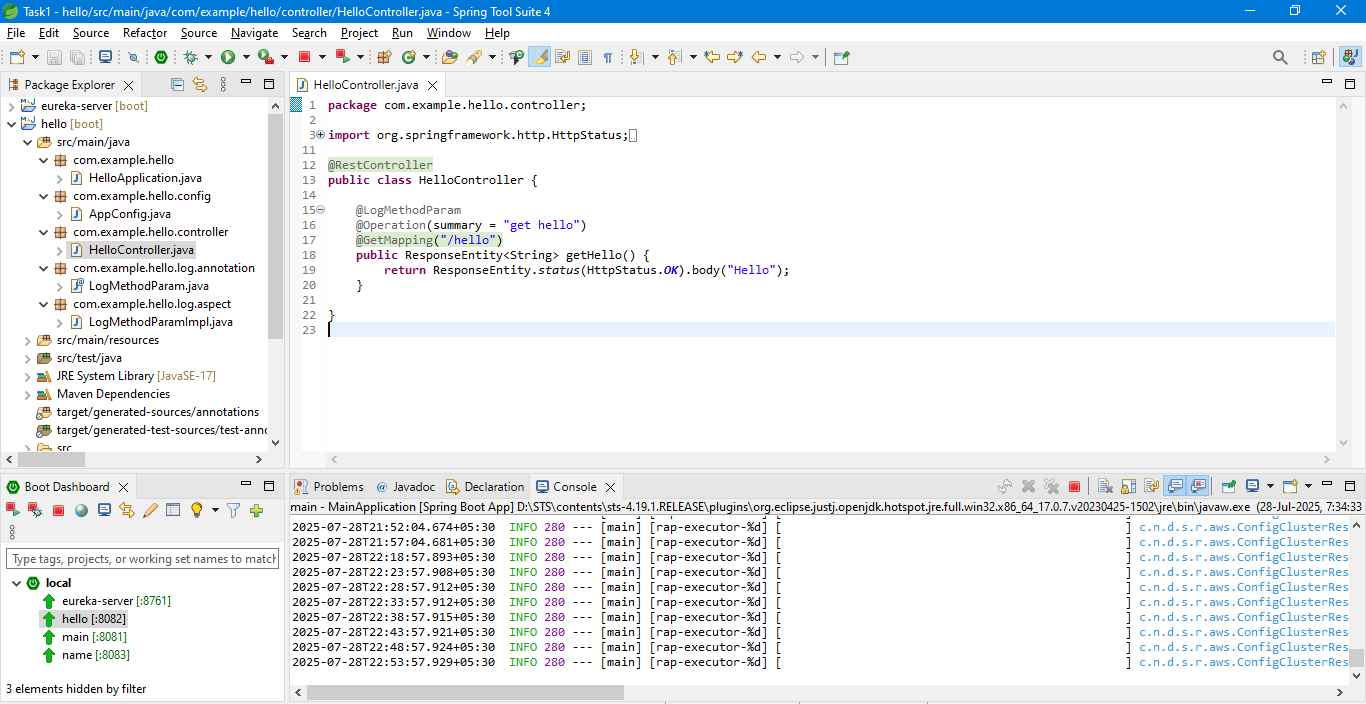
## Eureka dashboard

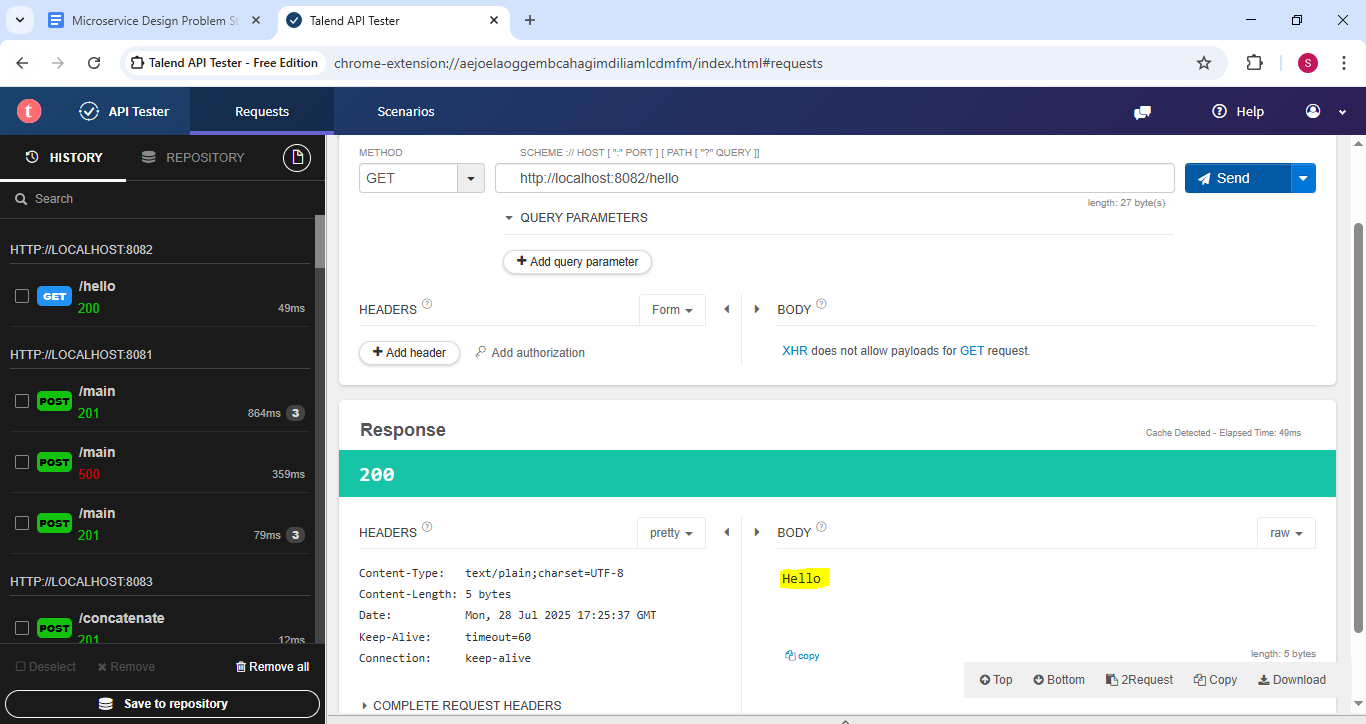
<http://localhost:8761/>





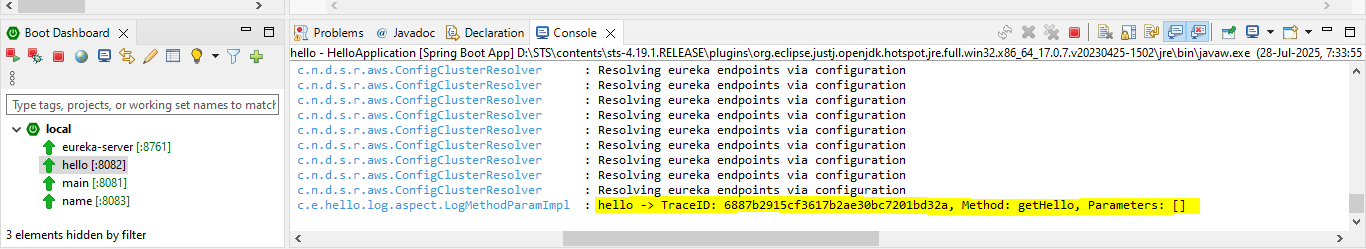
## Testing - Service 2: hello service



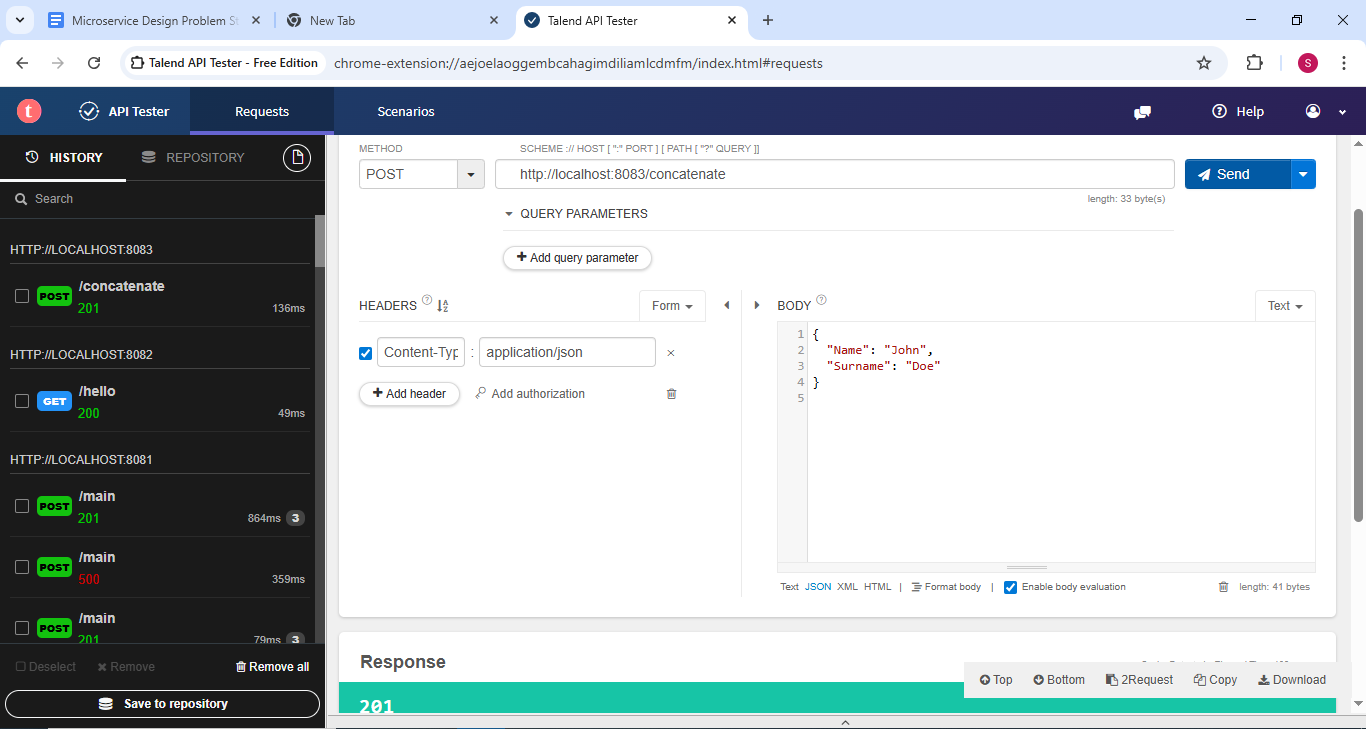


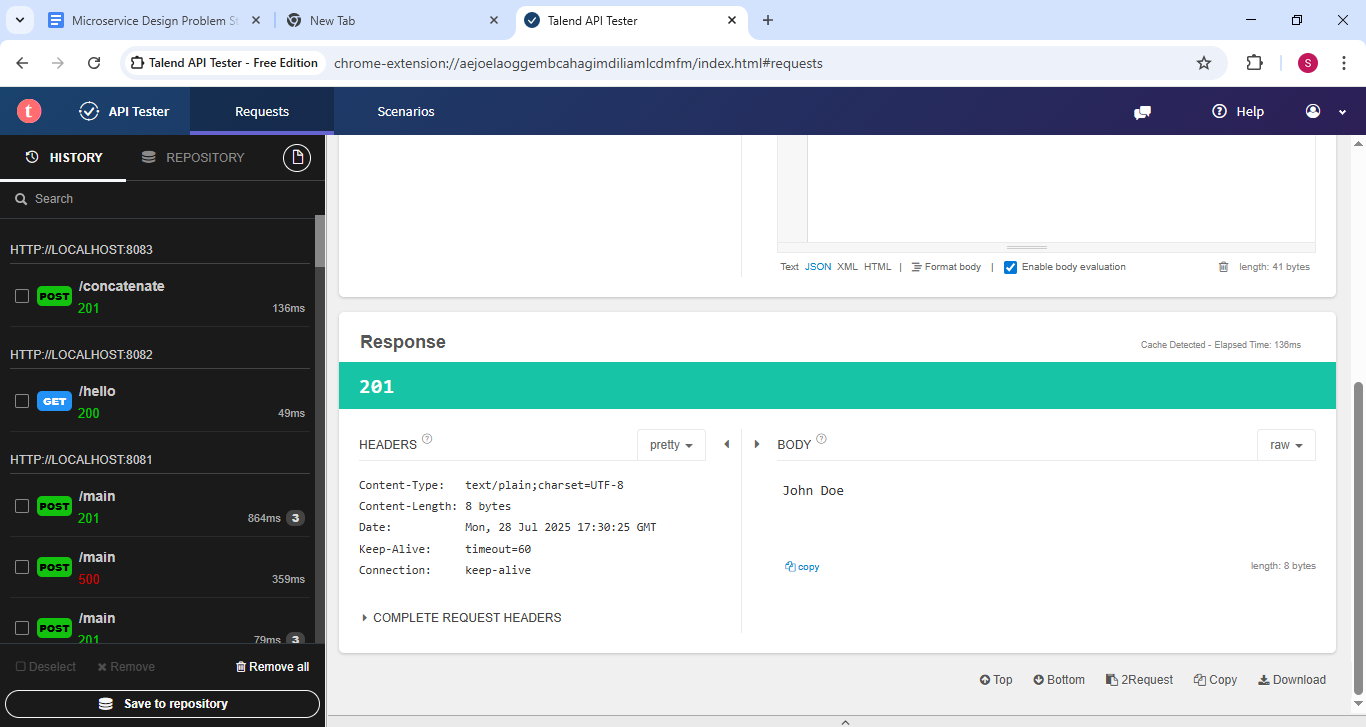
### Logging and Tracing

**hello -> TraceID: 6887b2915cf3617b2ae30bc7201bd32a, Method: getHello, Parameters: []**



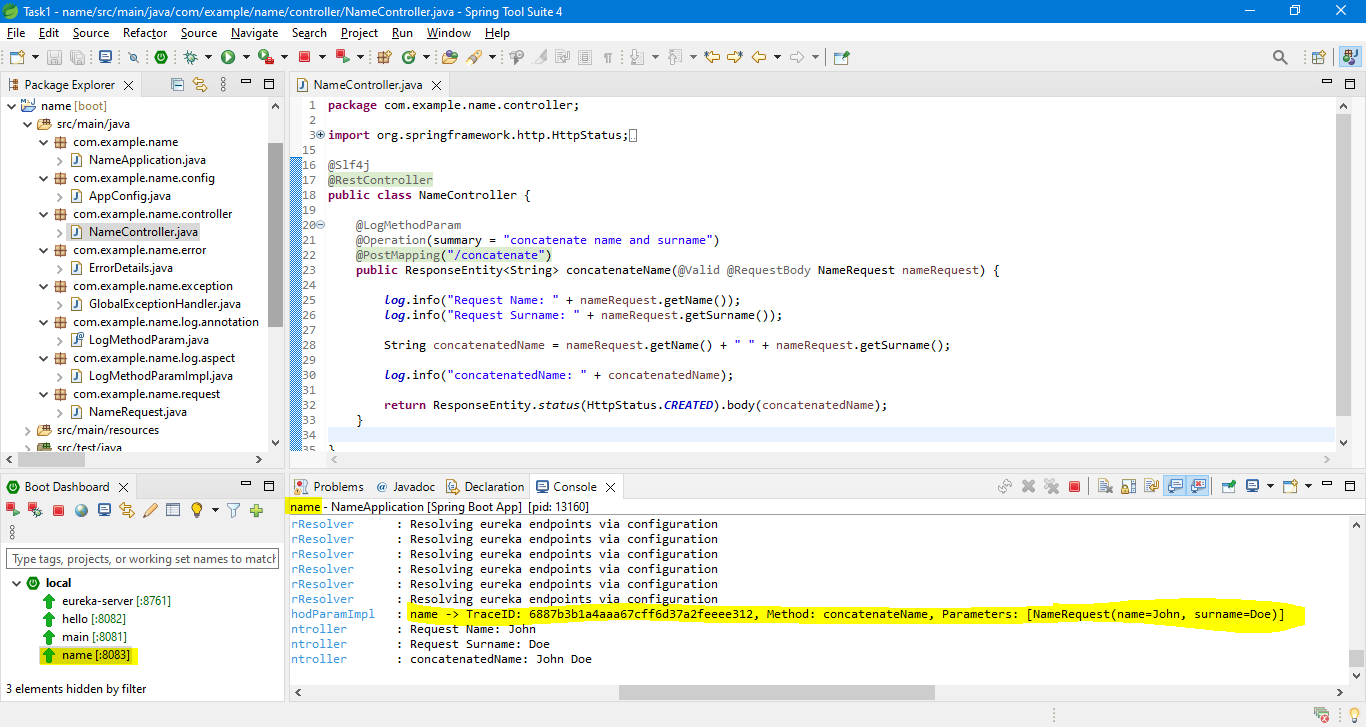
## Testing - Service 3: name service





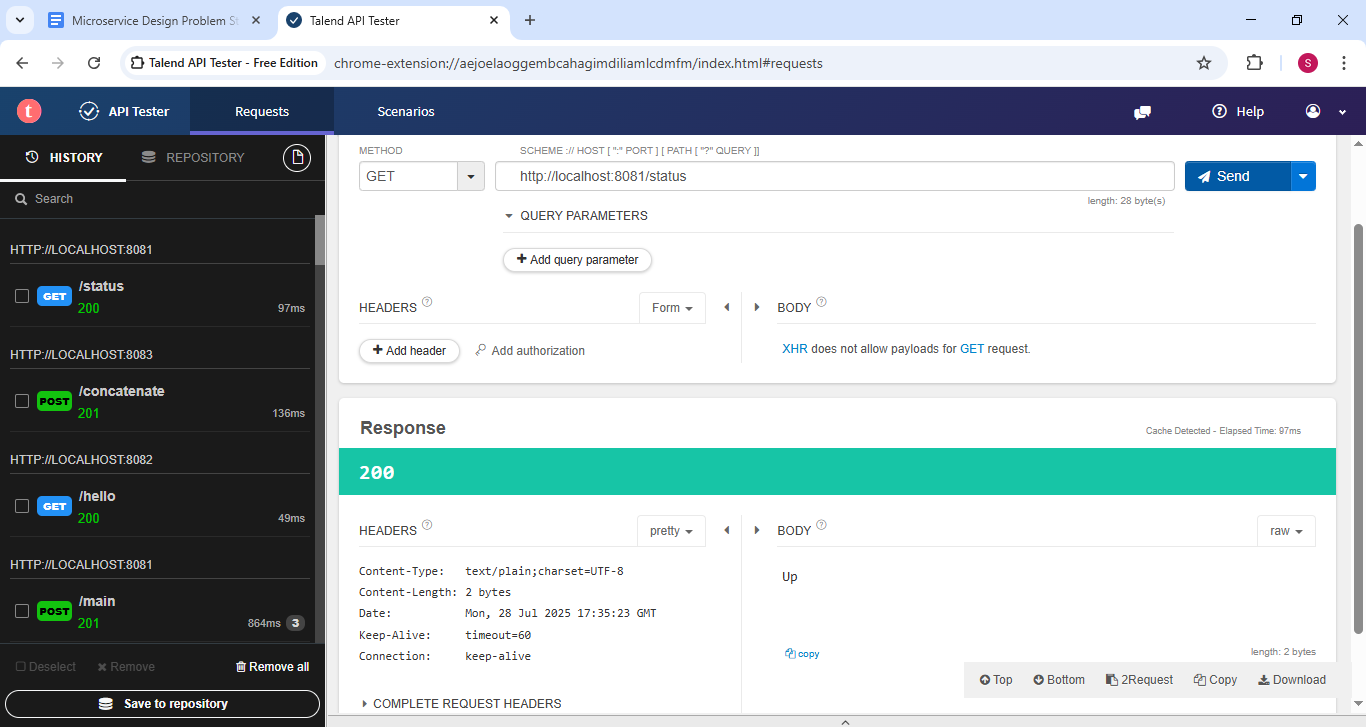
### Logging and Tracing

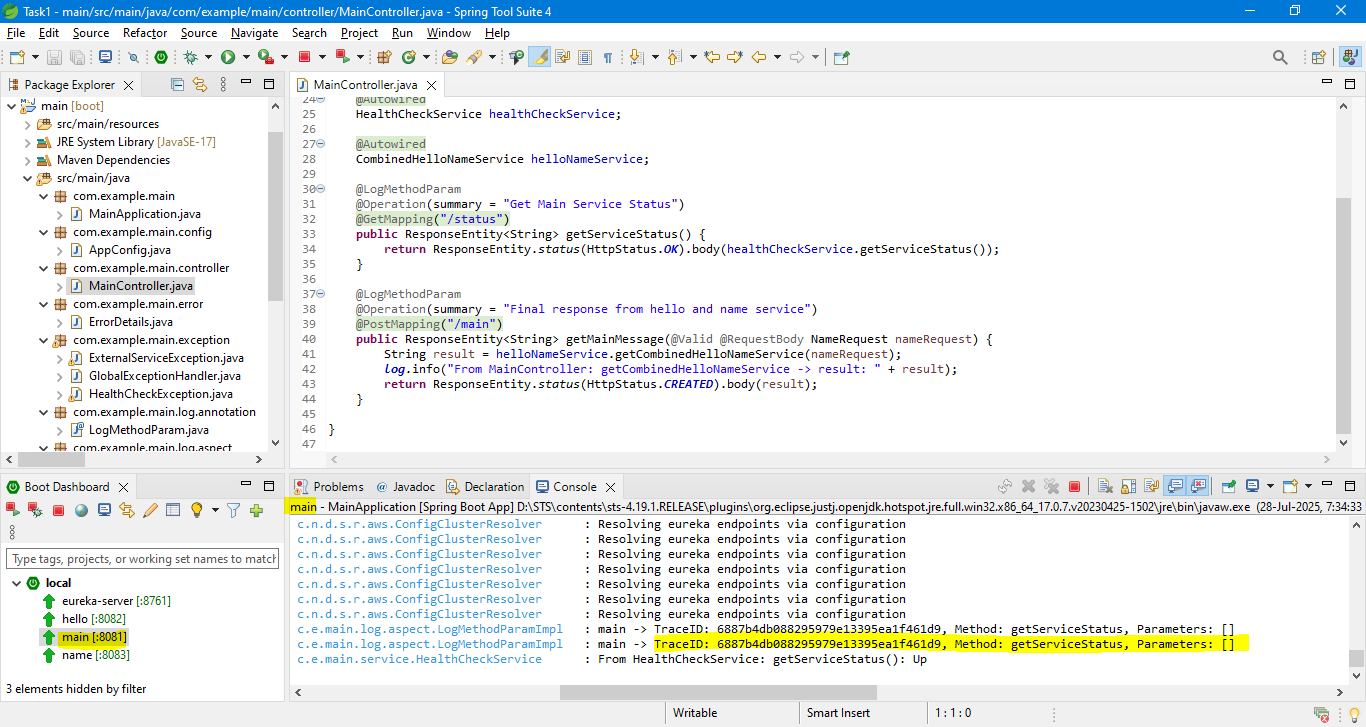
**name -> TraceID: 6887b3b1a4aaa67cff6d37a2feeee312, Method: concatenateName, Parameters: [NameRequest(name=John, surname=Doe)]**



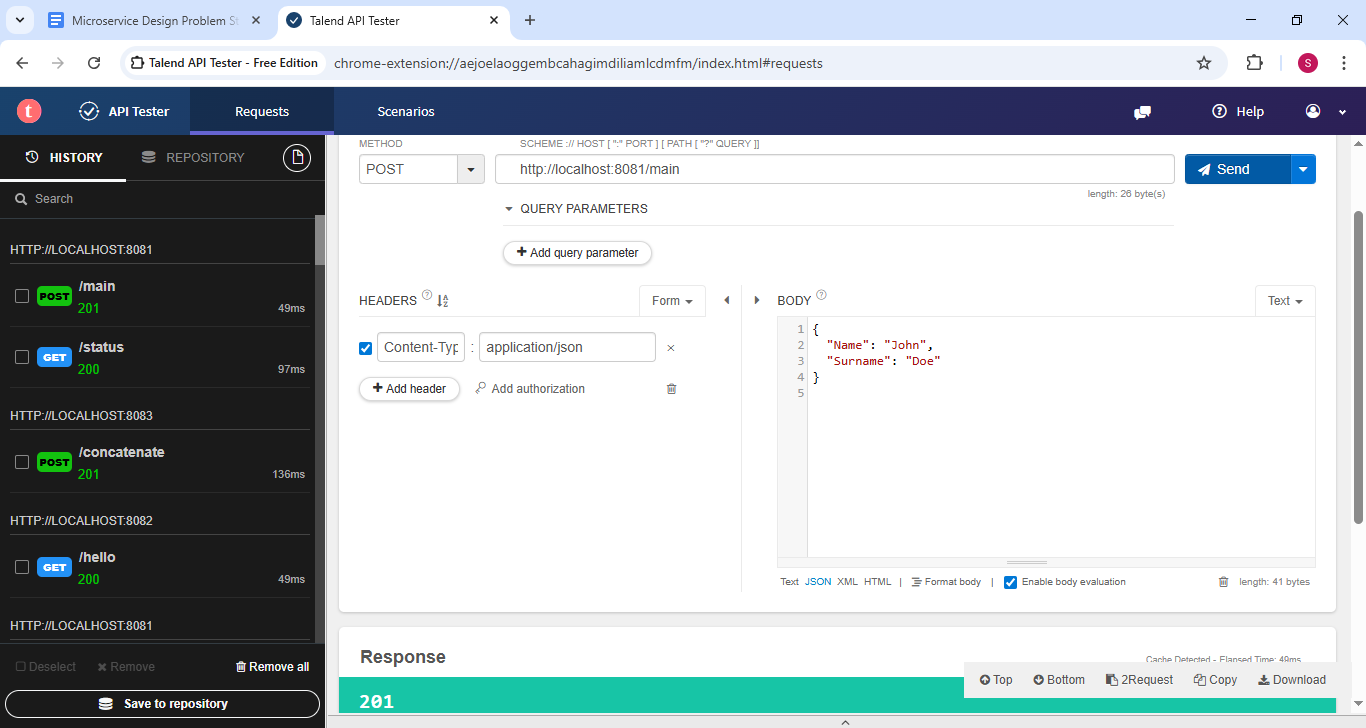
## Testing - Service 1: main service

### GET /status - Get Main Service Status.

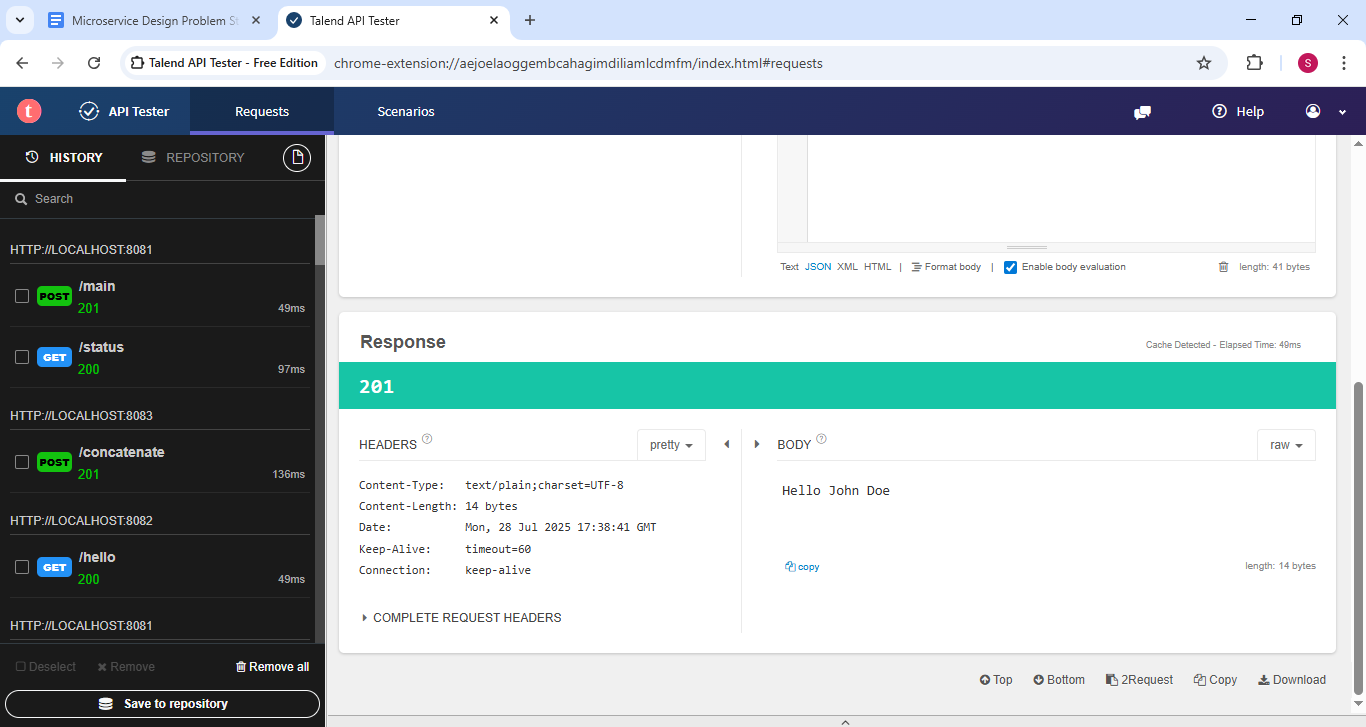


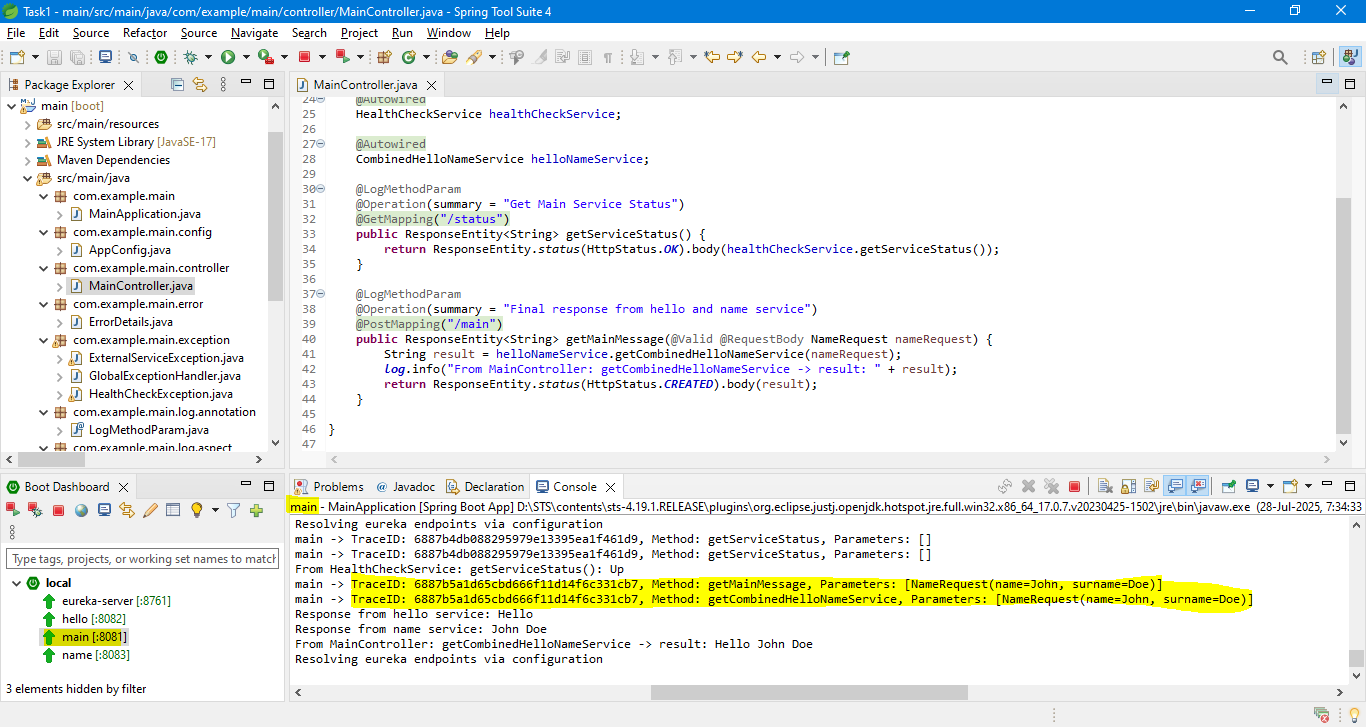


### POST /main - Final response from hello and name service combined.



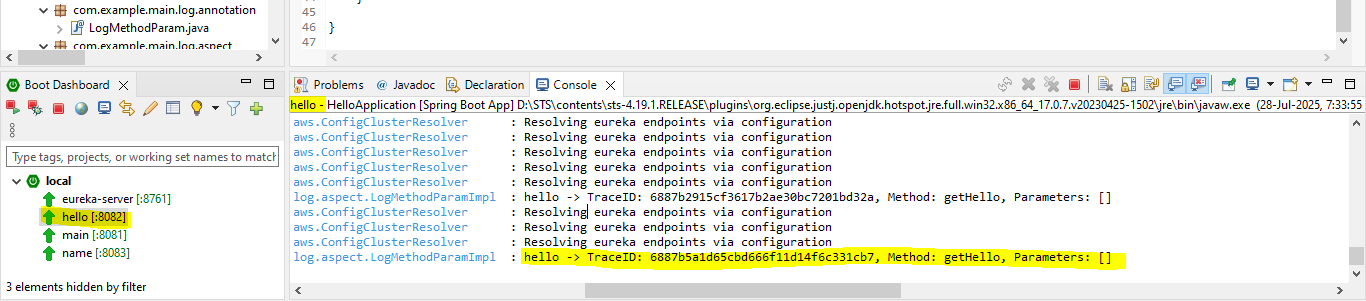
#### Logging and Tracing



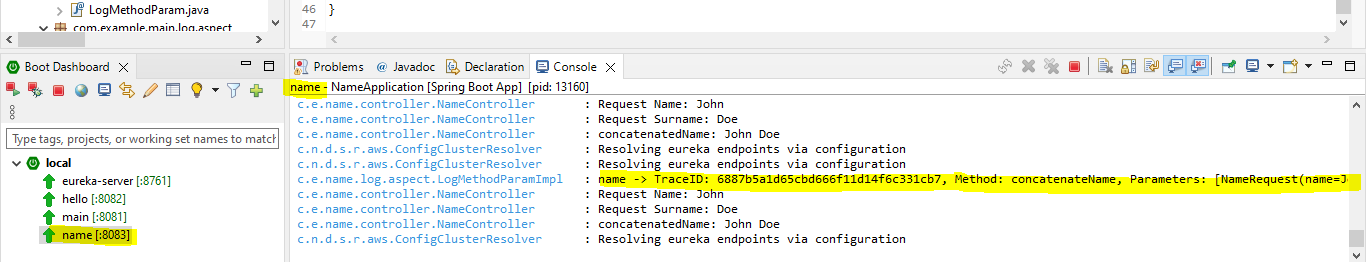


main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getMainMessage, Parameters: [NameRequest(name=John, surname=Doe)]

main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getCombinedHelloNameService, Parameters: [NameRequest(name=John, surname=Doe)]



hello -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getHello, Parameters: []



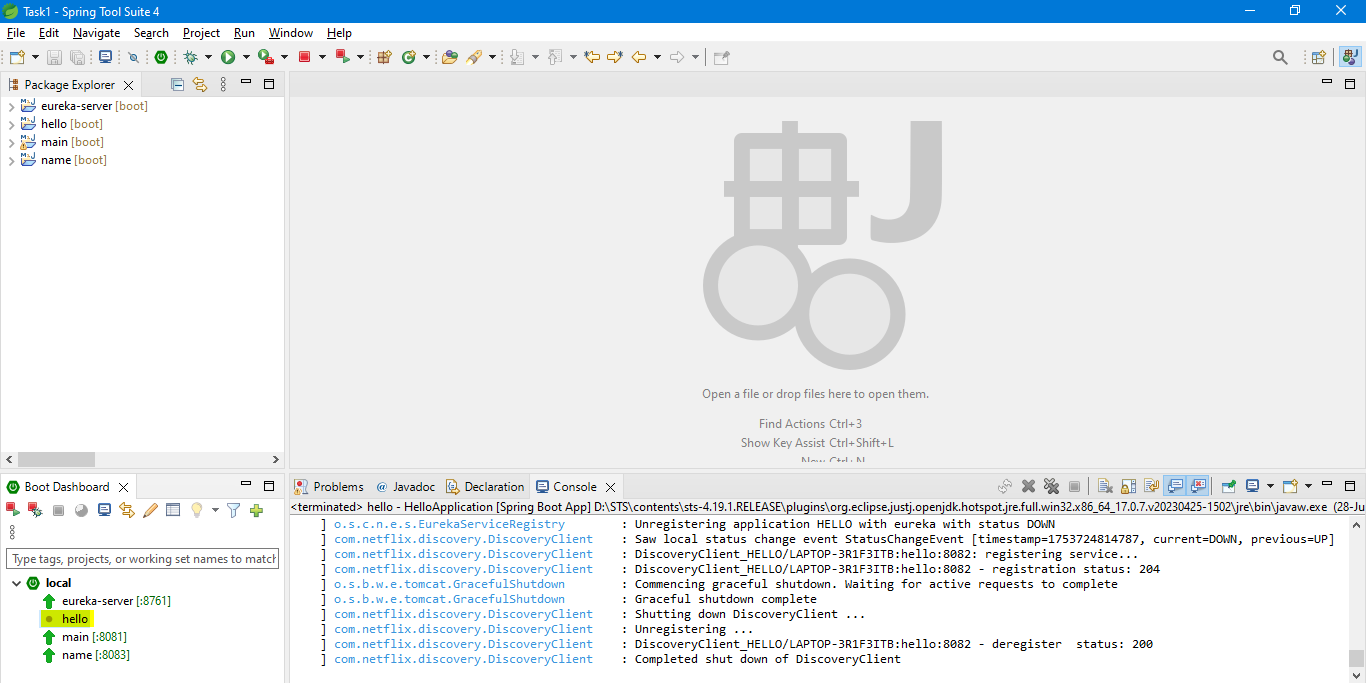
main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getCombinedHelloNameService, Parameters: [NameRequest(name=John, surname=Doe)]

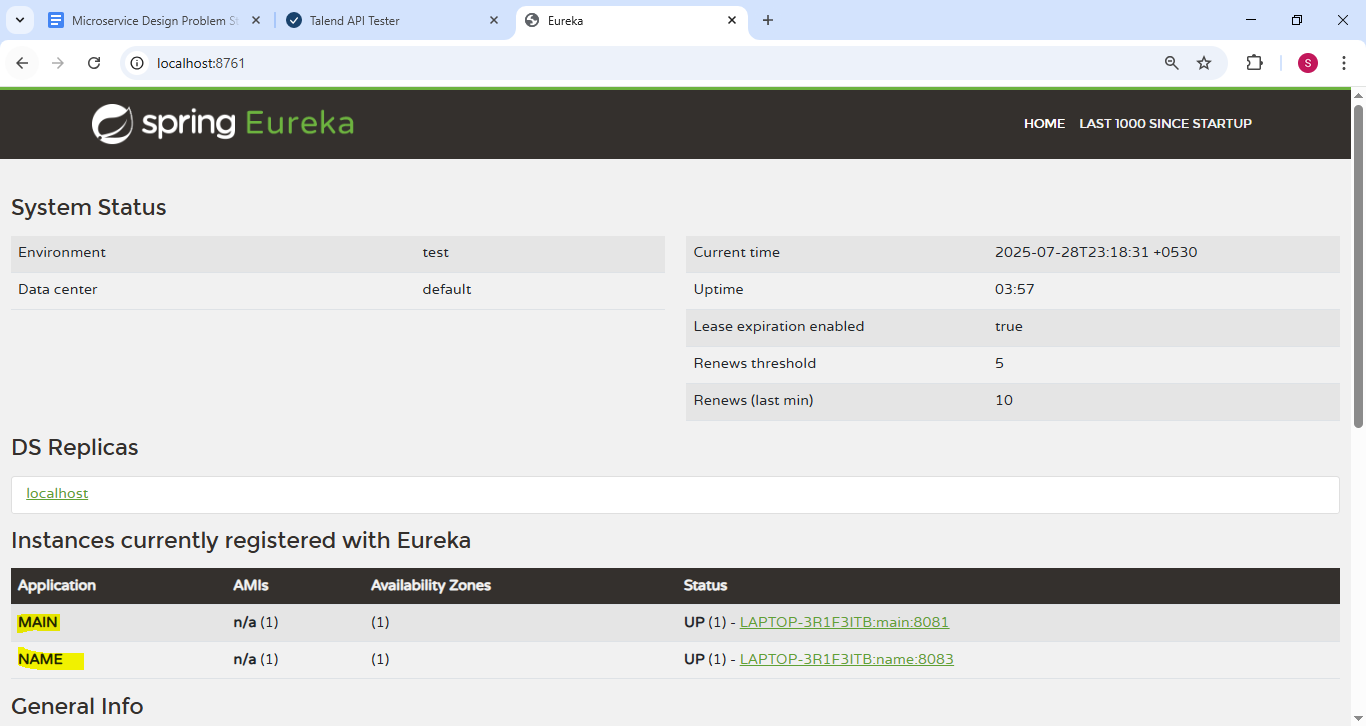
**All TraceId are same:**

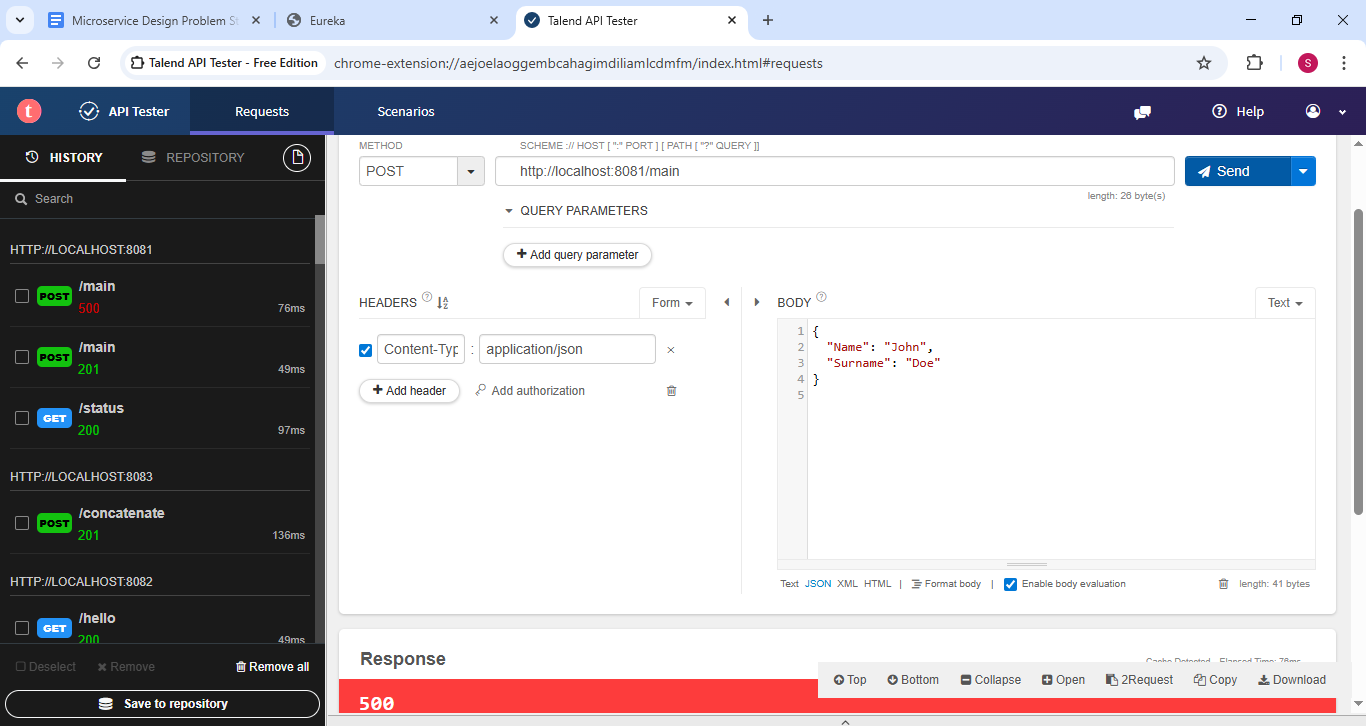
|  |
| --- |
| main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getMainMessage, Parameters: [NameRequest(name=John, surname=Doe)]  main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getCombinedHelloNameService, Parameters: [NameRequest(name=John, surname=Doe)] |
| hello -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getHello, Parameters: [] |
| main -> TraceID: **6887b5a1d65cbd666f11d14f6c331cb7**, Method: getCombinedHelloNameService, Parameters: [NameRequest(name=John, surname=Doe)] |

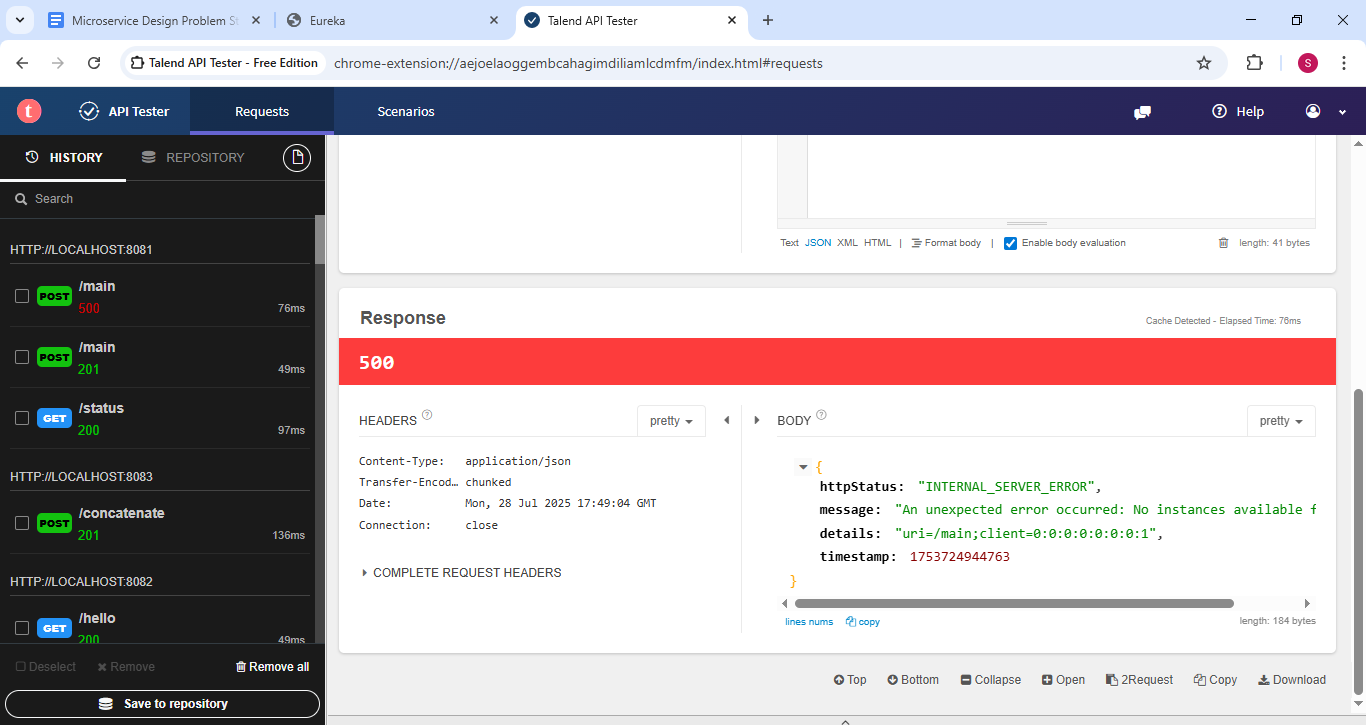
# Negative Scenario Testing

## Service 2: hello service – is down during calling Service 1: main service call









{

"httpStatus": "INTERNAL\_SERVER\_ERROR",

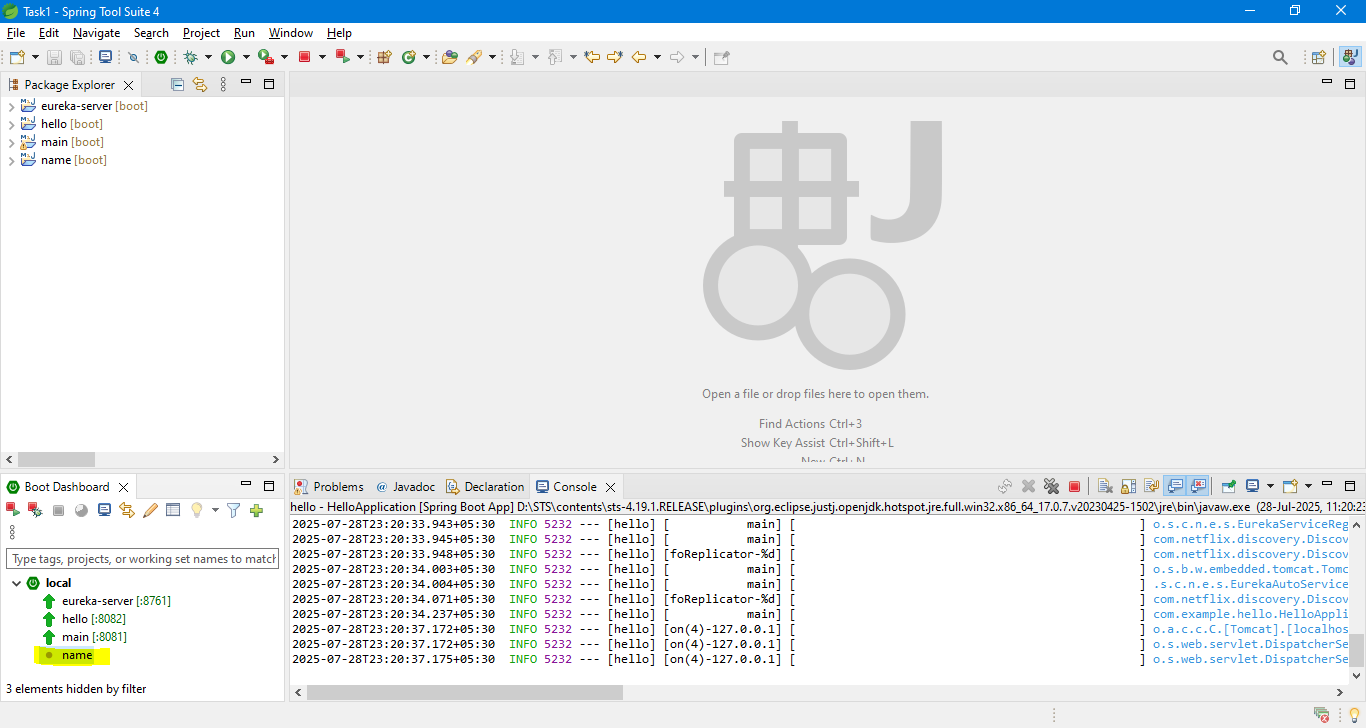
**"message": "An unexpected error occurred: No instances available for hello",**

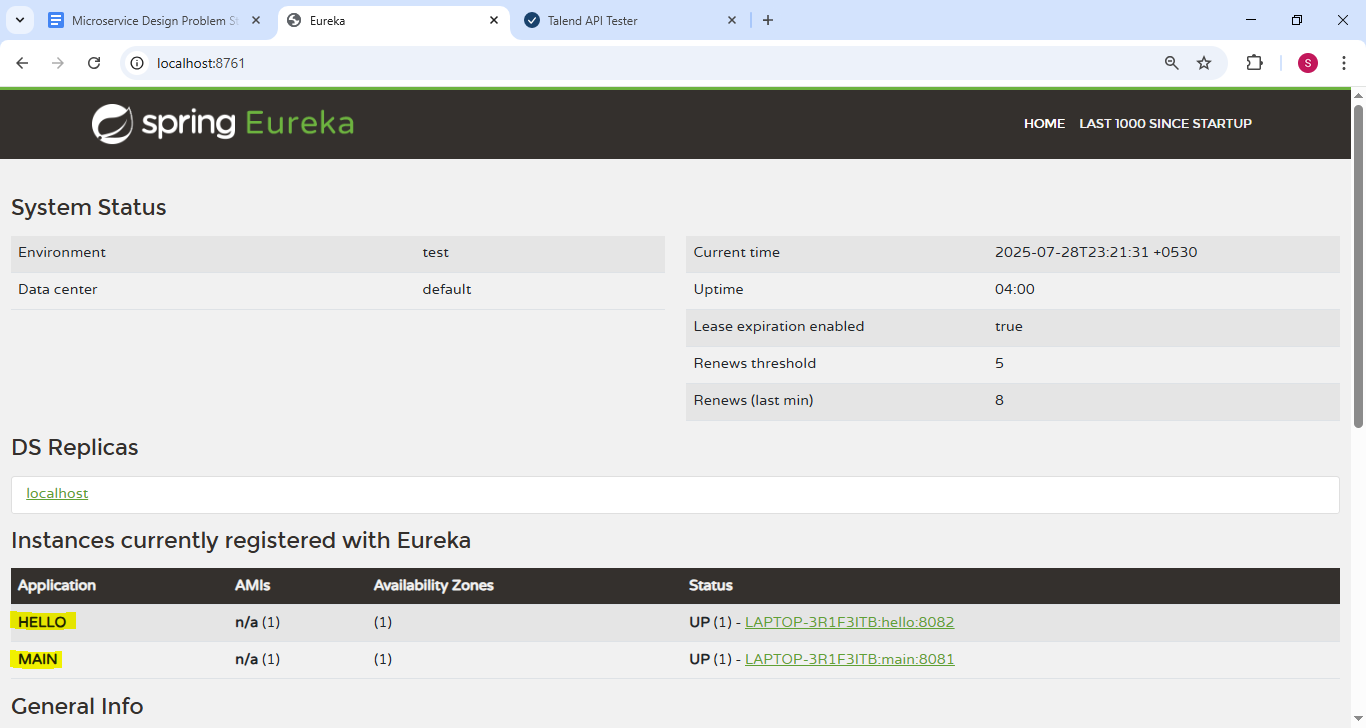
"details": "uri=/main;client=0:0:0:0:0:0:0:1",

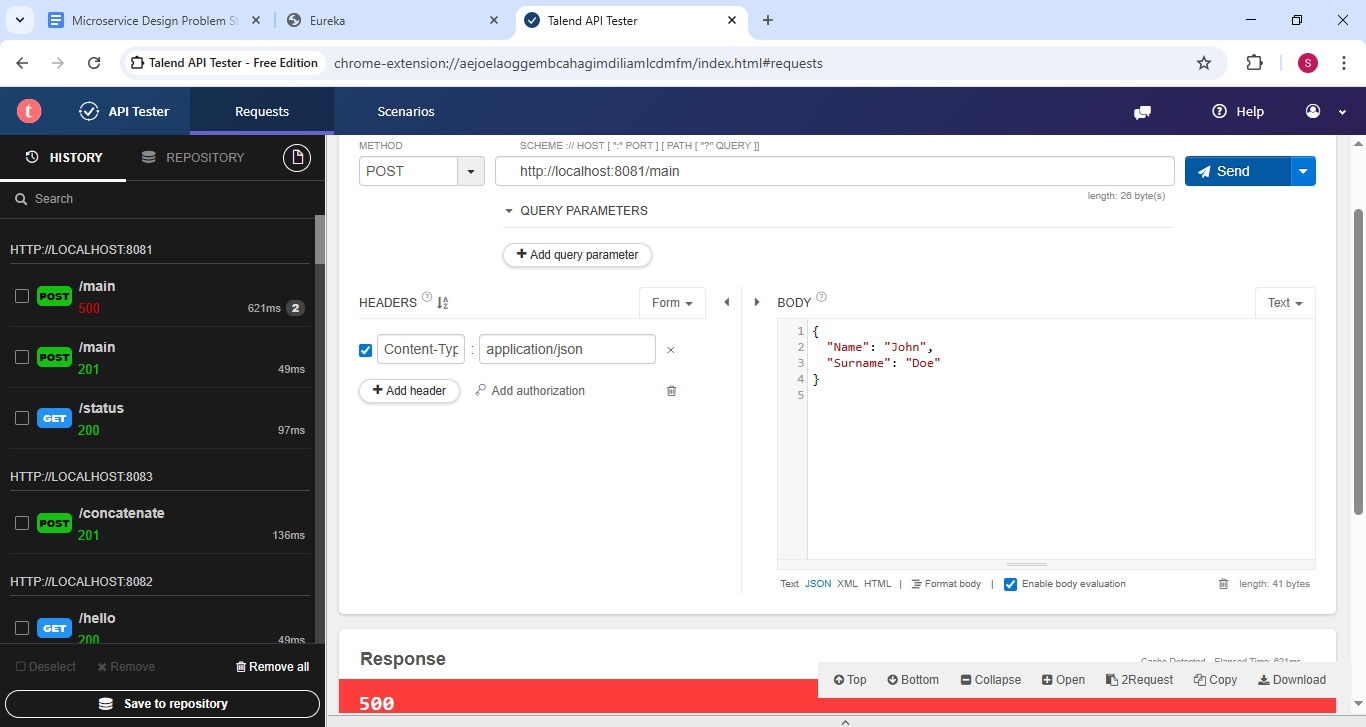
"timestamp": 1753724944763

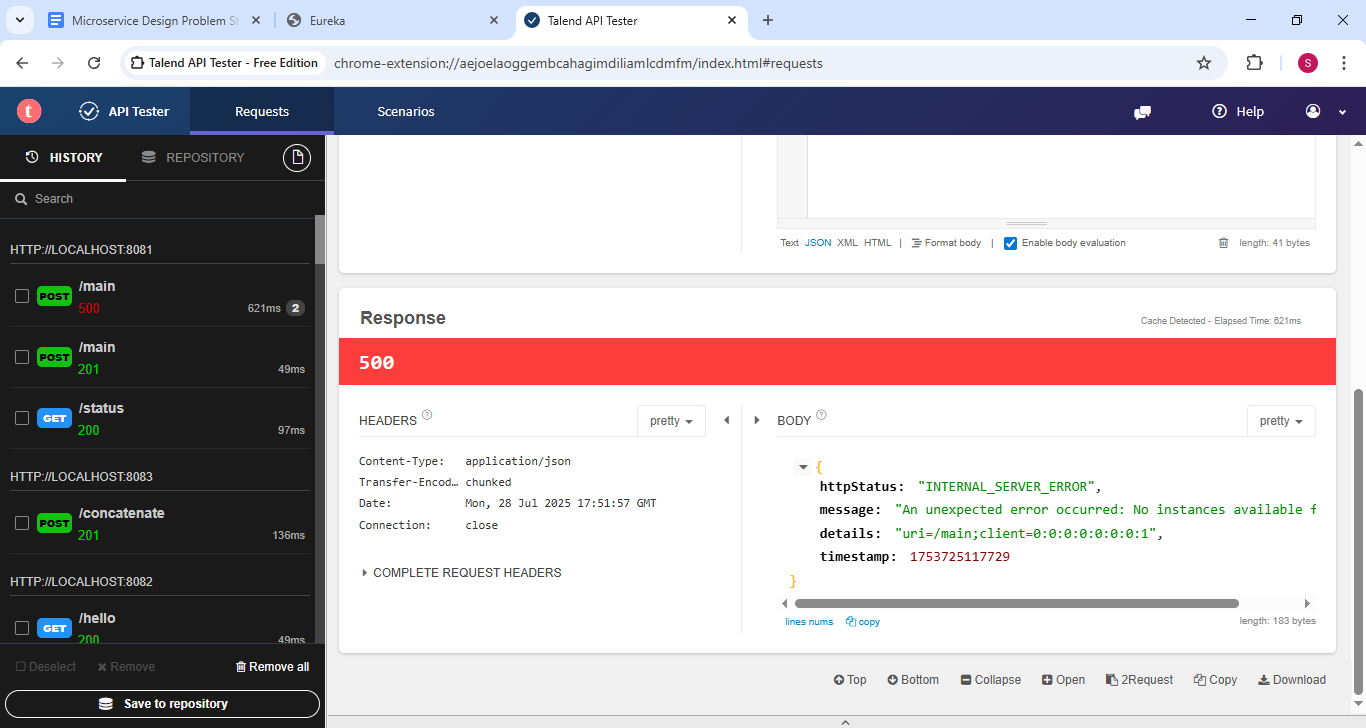
}

## Service 3: name service – is down during calling Service 1: main service call









{

"httpStatus": "INTERNAL\_SERVER\_ERROR",

**"message": "An unexpected error occurred: No instances available for name",**

"details": "uri=/main;client=0:0:0:0:0:0:0:1",

"timestamp": 1753725117729

}

## Name tag missing from Input Request Json (All service UP)

