Microservice Design Problem Statement

Use SpringBoot version 2+, Host below microservices in a aws free tier account.

TASK 1

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

TASK 2

Create
Db Connection:
Host a database in local or aws (postgres or mysql or inmemory db(H2)).
Configure your microservice to connect to db.
Create Hibernate Entity Class based on the below table requirement to auto create tables in hosted db.
Populate the db from a post request with the below data or directly insert into db.

Expose endpoints to fetch requests based on Id and complete list of objects.
While getting the complete table, Modify the result set(arrays or list) to form nested object structure.
Below is the nested output in json.Associate color to each object according to table.

Create a table with fields ID, Name, Color, ParentId and populate with below data.
Don't use ORM to map or form parent child relationships. Write an algorithm to form the relationship after fetching the list from db, in an efficient way.

id	parentid	name	color
1	0	Warrior	red
2	0	Wizard	green
3	0	Priest	white
4	0	Rogue	yellow
5	1	Fighter	blue
6	1	Paladin	lighblue
7	1	Ranger	lighgreen
8	2	Mage	grey
9	2	Specialist wizard	lightgrey
10	3	Cleric	red
11	3	Druid	green

12	3	Priest of specific mythos	white
13	4	Thief	yellow
14	4	Bard	blue
15	13	Assassin	lighblue

Nested Json response. Ignore Dangling Branches Scenario.

```
[
  {
    "Name": "Wizard",
    "Sub Classes": [
      {
        "Name": "Mage"
      },
      {
        "Name": "Specialist wizard"
      }
    ]
  },
  {
    "Name": "Priest",
    "Sub Classes": [
      {
        "Name": "Cleric"
      },
      {
        "Name": "Druid"
      },
      {
        "Name": "Priest of specific mythos"
      }
    ]
  },
  {
    "Name": "Warrior",
    "Sub Classes": [
      {
        "Name": "Fighter"
      },
      {
        "Name": "Paladin"
      },
      {
        "Name": "Ranger"
      }
    ]
  },
  {
    "Name": "Rogue",
    "Sub Classes": [
      {
        "Name": "Thief",
        "Sub Classes": [
          {
            "Name": "Assassin"
          }
        ]
      },
      {
        "Name": "Bard"
      }
    ]
  }
]
```

Create a method level Annotation @LogMethodParam which logs parameters passed to method.