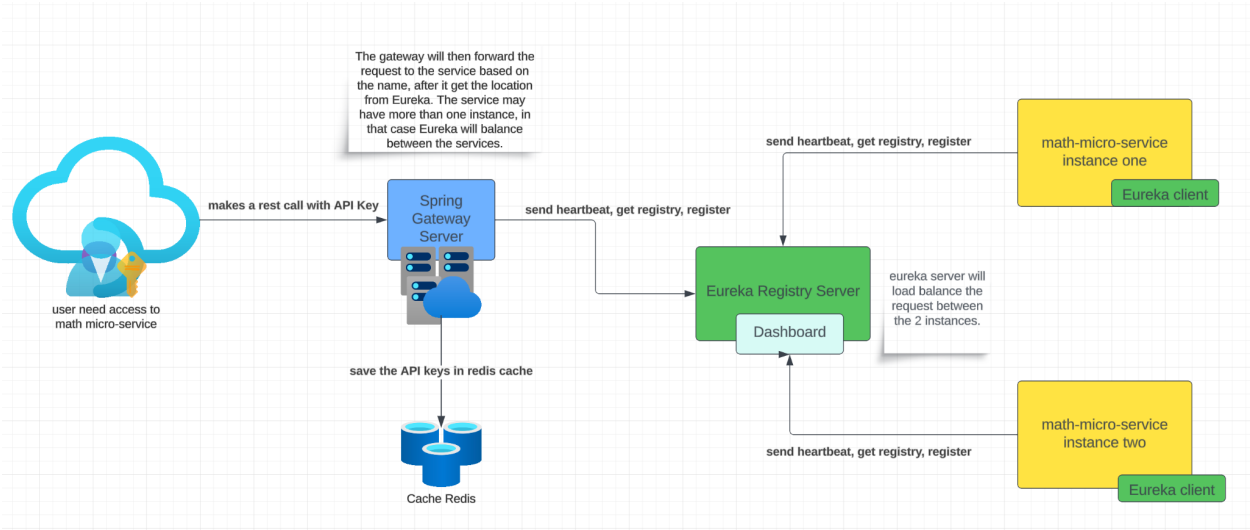
Design for secure, load balanced microservice system



The system is simple.
It is composed of 6 parts, 3 backend servers (Redis, Eureka and Gateway) back end servers with 2 Spring Boot micro services and one front end Angular UI support client.

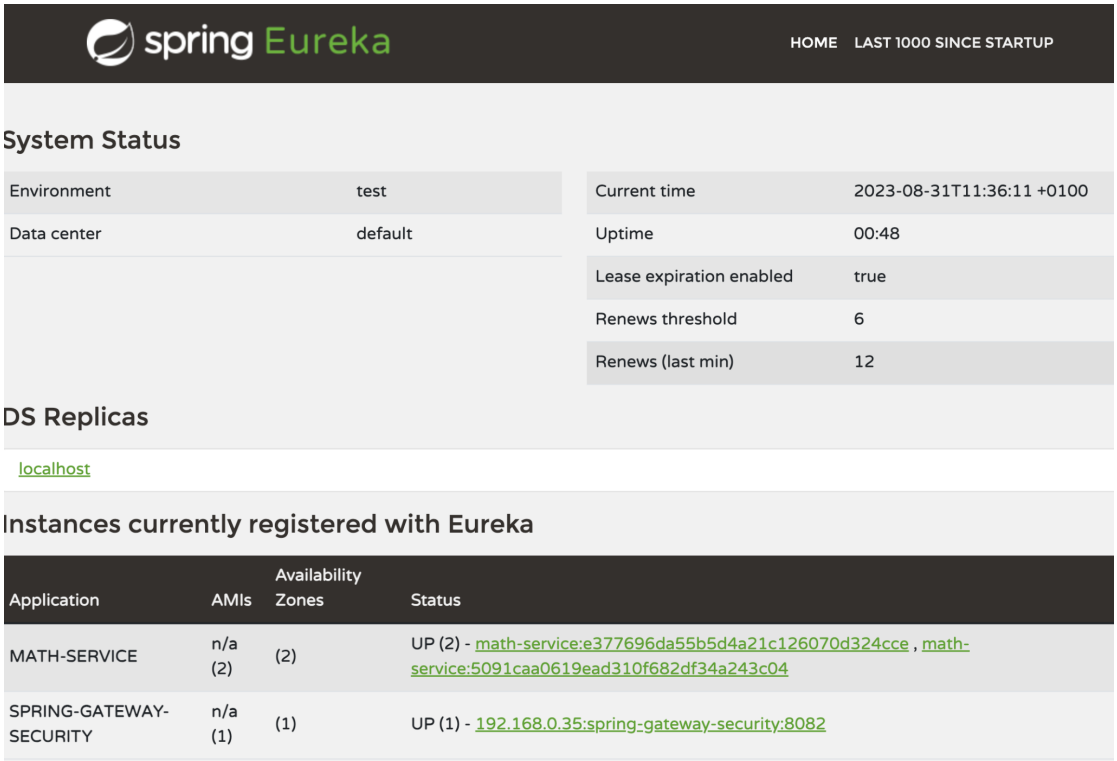
The Gateway role is in 2 parts:
1 - Authenticate the user with API key and Authorize for a route to a server
2 - forward the request to another path to a microservice

The Gateway uses Redis to maintain record of key and service route allowed. At the start of the application the keys populate Redis.
Afterward the Gateway queries Redis for authorize clients access to a service

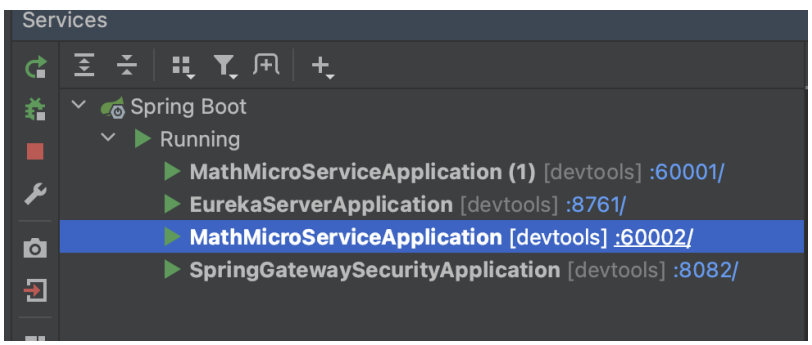
User is authenticated if the key has a mapping in Redis for a service and at the same time gives authorization to access a service.

Eureka server is a registry and load balancer.
In the picture below of the Registry dashboard, we have 2 instances of the math-service.

Using the Angular app we can visualize the load balancing when performing a multiplication multiple time



Eureka Dashboard



The application run ing inside intellij.

Design Patterns:

API Gateway Pattern

Since the application is microservice based, I used gateway to allow a single entry point for security purpose and Request's response are returned back to caller.
This design is good when there are many services behind the gateway that needs to be accessed.

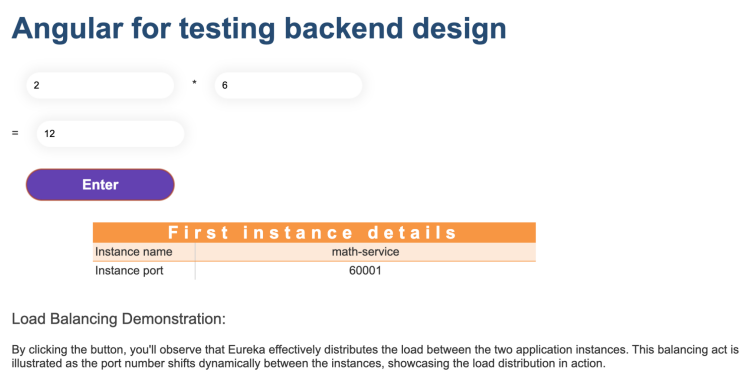
Gateway routing pattern

The application exposes a single endpoint to route requests to microservices.

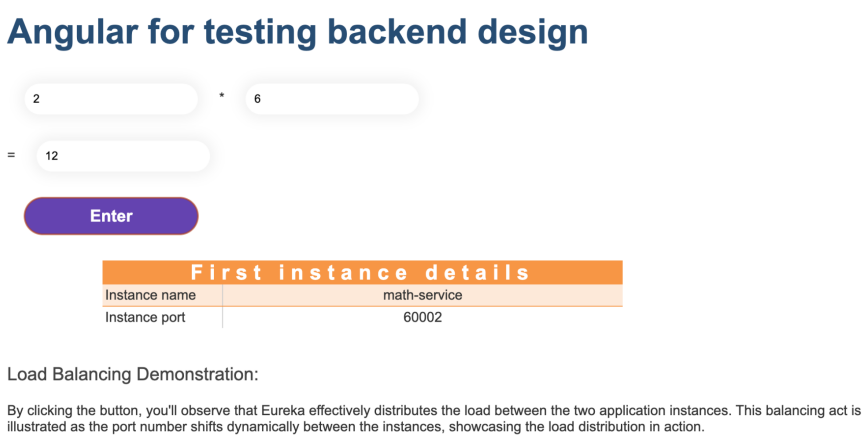
Service Registry pattern
The application uses Eureka for registry and discovery.

Microservice Architecture Pattern
Because we need multiple concurrent usage which calls for multiple instances, and the applications are small by nature, microservice architecture will be well served here.
as a set of independently deployable, loosely coupled components or services.

The characteristics of the application are few design patterns, API key infrastructure, Eureka registry and an angular UI support application.



And if you click enter again… this time the request is sent to the second instance of the service, verified by the changing of port



RUNNING THE APPs

1. **Start the Eureka**
   as a Spring Boot Registry server first to allow the application clients to register in this case, the Gateway, the 2 services instances.
2. **Second start Redis** (assume Redis is already installed locally)
3. **Start the Gateway**
4. **Start the 2 math micro services** instances (same application just duplicate)
5. **Start the angular client** by running “ng serve” that assume that you have nodejs installed, npm package manager and angular cli installed and configured before hand.

All 5 apps are Spring Boot applications and the front end app is Angular version 6

Fred Assi