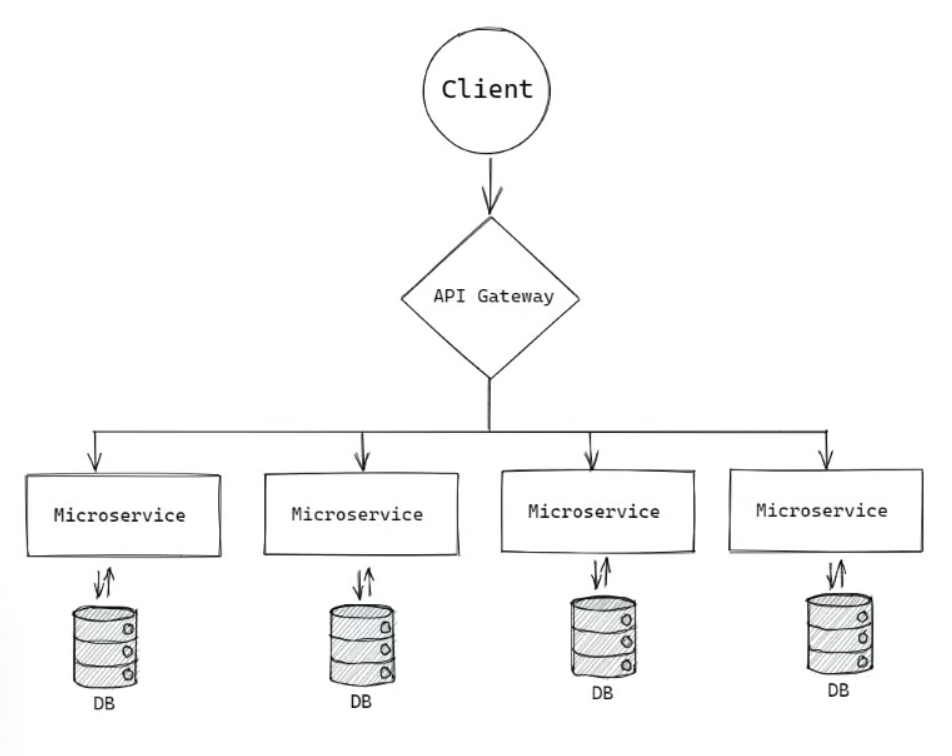
# **Spring Cloud Quickstart Guide**

‍Microservices architecture, or simply microservices, is a unique method of developing software systems that focuses on building single-function units with well-defined interfaces and operations.

In a microservices architecture, an application is decomposed into a set of autonomous services that communicate with each other through well-defined APIs (Application Programming Interfaces). Each microservice is built around a specific business domain and performs a specific task, such as user authentication, data storage, or payment processing.

**A Sample Microservices Architecture Diagram:**



In this project each microservice exposes a REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style.

A single microservice has its own set of responsibilities. Each one can only interact with its own database. With a microservice, businesses get the benefits of flexibility provided by loosely coupled systems. Each microservices architecture can work freely.

**Microservices Architecture is the perfect solution for:**

* Scaling
* Fault isolation
* Providing technology stack independence
* Ensuring data security

Microservices architecture has gained significant popularity due to its ability to create scalable, resilient, and independently deployable services. The Spring Framework, a robust and mature Java-based framework, provides comprehensive support for building microservices.

**Technology Uses for the project:**

* Java 17
* Spring Boot 3.xx
* Maven

**The architectural basic composition of microservices consists of:**

**Config sever:** a config-server for manage all config from a central location.

**Two microservice** – small independent service (MICROSERVICE-ONE, MICROSERVICE-TWO).

**Service discovery** – helps microservices to locate each other in an ecosystem.

**API gateway** – an open path between microservices and outside clients.

Together intercommunication and data exchanges form the functions of a complete application.

**Microservices with Spring Framework**

Spring Framework for Microservices: Empowering Scalable and Resilient Architecture.

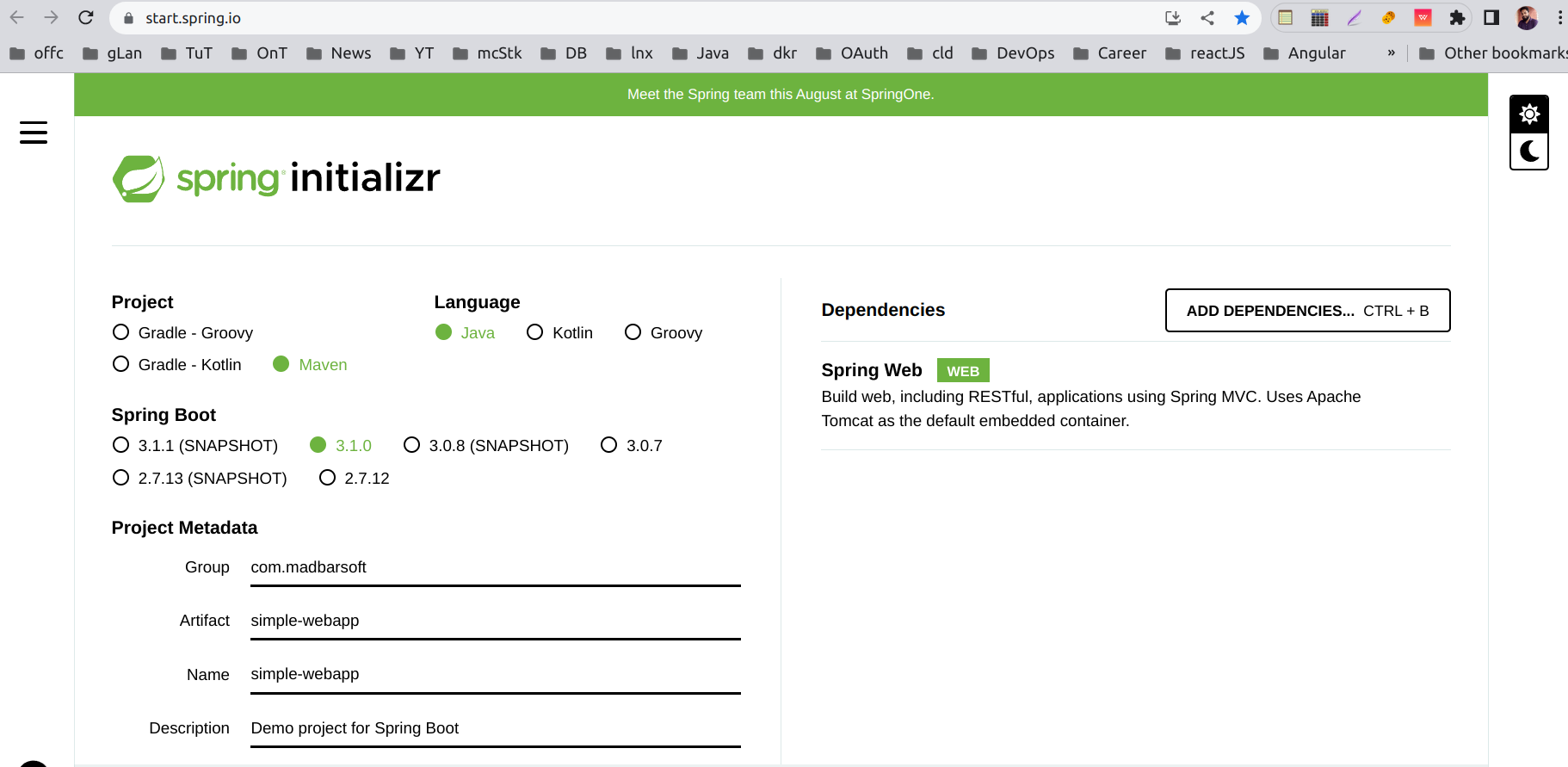
Required Spring Tools for Development Microservices Project as below:

**Spring Boot:**

Spring Boot is an extension of Spring, which eliminates the boilerplate configurations required for setting up a Spring application. Featuring default codes and annotation based configuration, Spring Boot enables a faster and more efficient development ecosystem.

Starting a simple stateless web-service by Spring Boot:

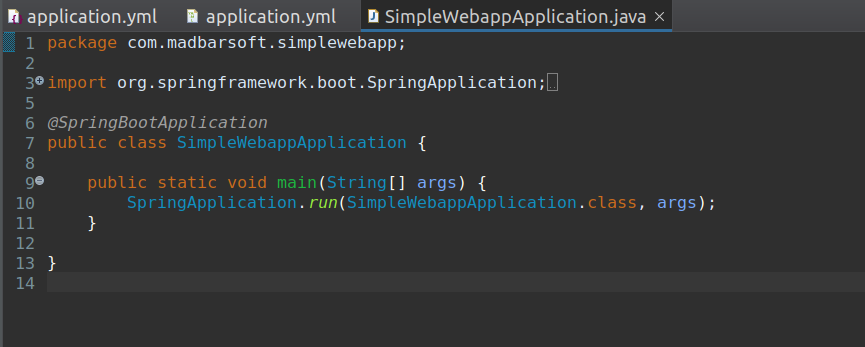
<https://start.spring.io/>, This is Spring Official site for initiation, this **sample-webapp** we user only **Spring Web** Starter with java 17, maven as build tools to keep it sample.



As select the option download the project extract it and input you favorite idea as a maven project.

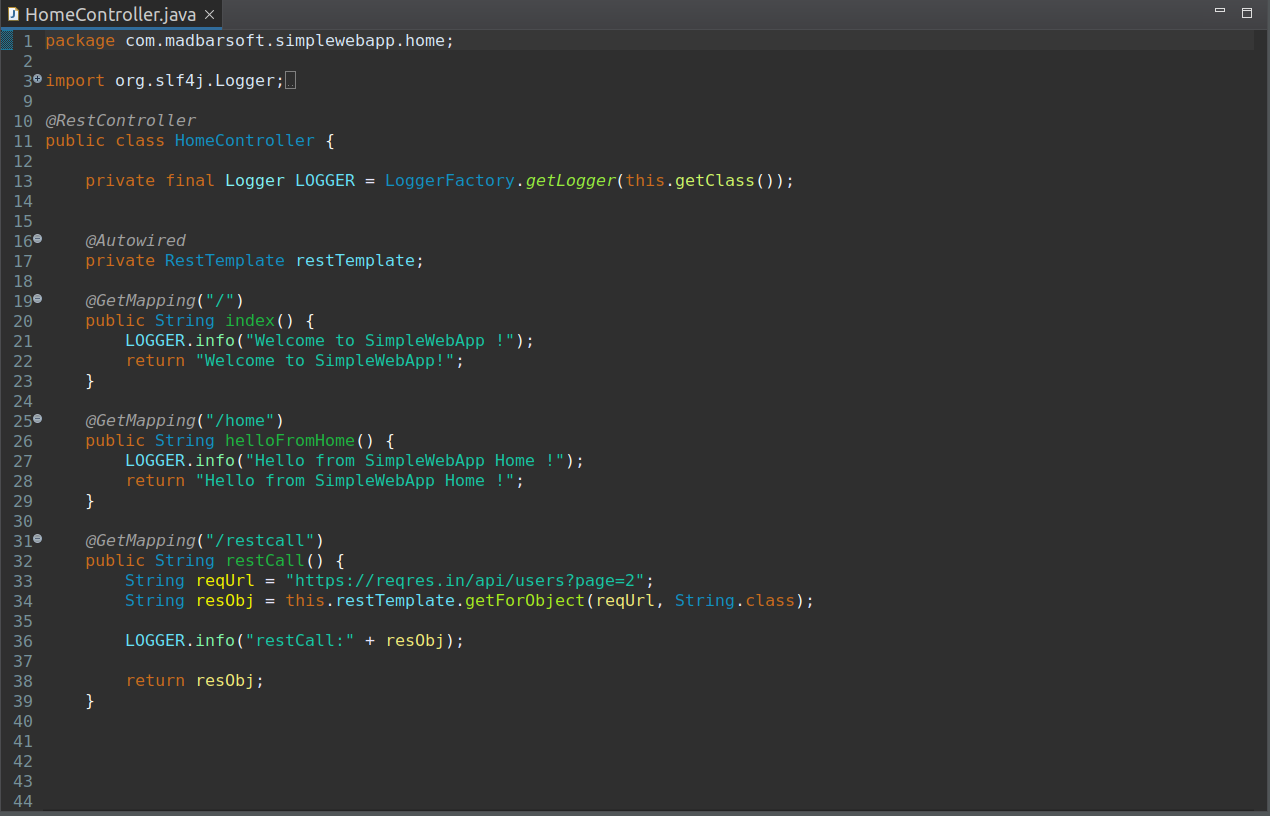
For Running our first project, find ths main method from package what is the starting point or entry point to the project, method annotated with **@SpringBootApplication.**

The project wii run on 8080 (default) port in you host machine, and can be access the RESTful-API.



To access the service: $crul <http://localhost:8080/home>, or throw browser.

Now create a sample Controller for routing user request, as below with several end-point.This end-point represent each one is one of resource in microservices ready to served user request.



**Spring Cloud:**

Spring Cloud, a sibling project of Spring Boot, provides essential tools and libraries for building microservices-based applications. It offers features like service discovery, intelligent routing, load balancing, fault tolerance, and distributed configuration management. Spring Cloud integrates seamlessly with popular service registry technologies like Netflix Eureka and Consul.

We just need some different Starter as dependency for different component of microservices like: Discovery eureka, Discovery client, Gateway, Config-server etc.

Starting with config server.

**Spring Cloud Config:**

Spring Cloud Config is Spring's client/server approach for storing and serving distributed configurations across multiple applications and environments.

This configuration store is ideally versioned under Git version control and can be modified at application runtime. While it fits very well in Spring applications using all the supported configuration file formats together with constructs like Environment, PropertySource, or @Value, it can be used in any environment running any programming language.

Spring cloud config-sever starter:

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

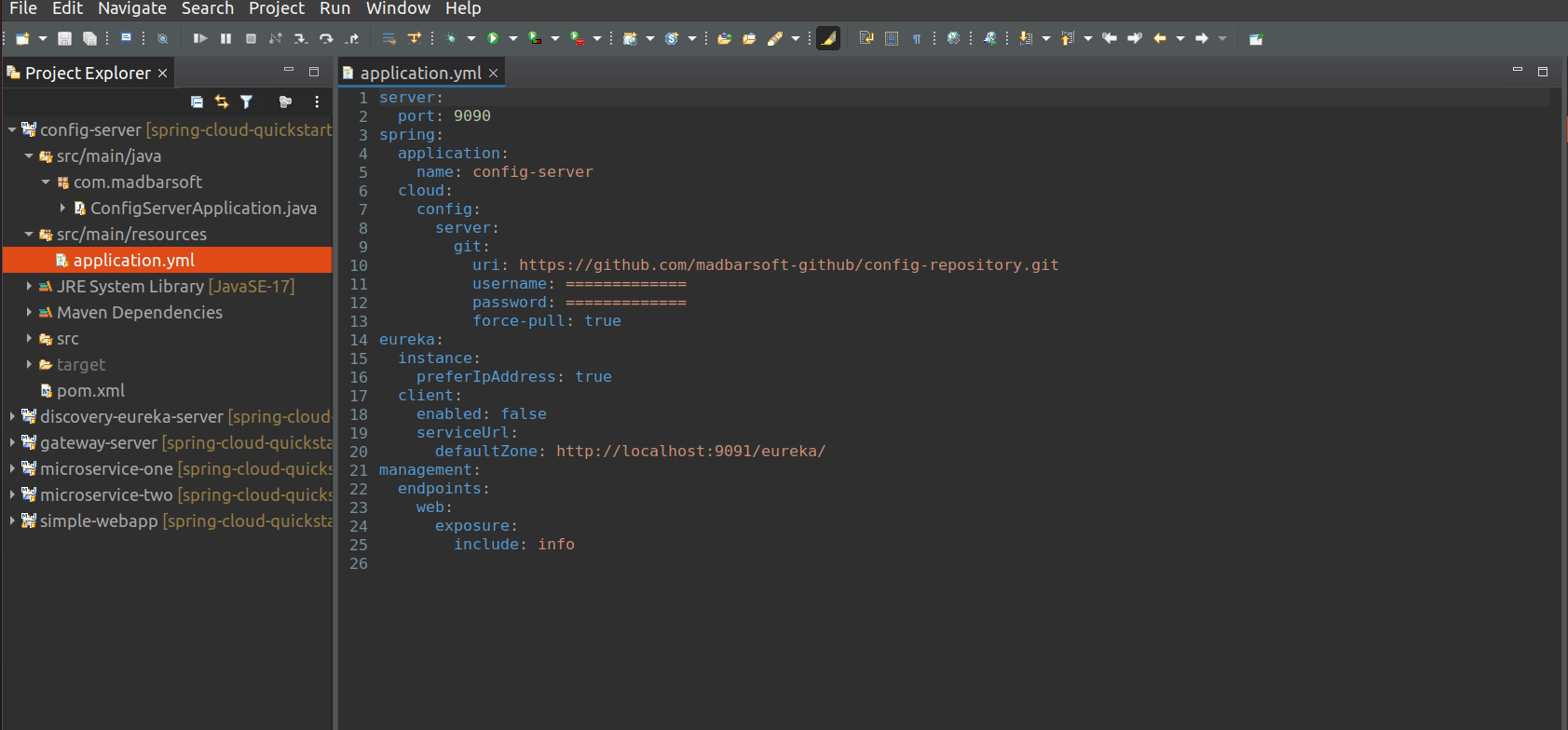
*<artifactId>spring-cloud-config-server</artifactId>*

*</dependency>*

Configuration properties for a typical spring config-server with git url and repository.

This is a yaml file for arrange our config properties. All service on starting time ask configuration from this config-server.

Microservice are know where there config-server with server url and port in its application.yaml file.



**Service Registration and Discovery:**

Spring Cloud Netflix Eureka, a part of the Spring Cloud ecosystem, enables dynamic service registration and discovery. Microservices can register themselves with Eureka, making it easy to locate and communicate with other services. Eureka also supports high availability and fault tolerance, ensuring seamless service discovery in distributed environments. Add this below dependency on project POM file.

Spring cloud Discovery Starter:

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-config</artifactId>*

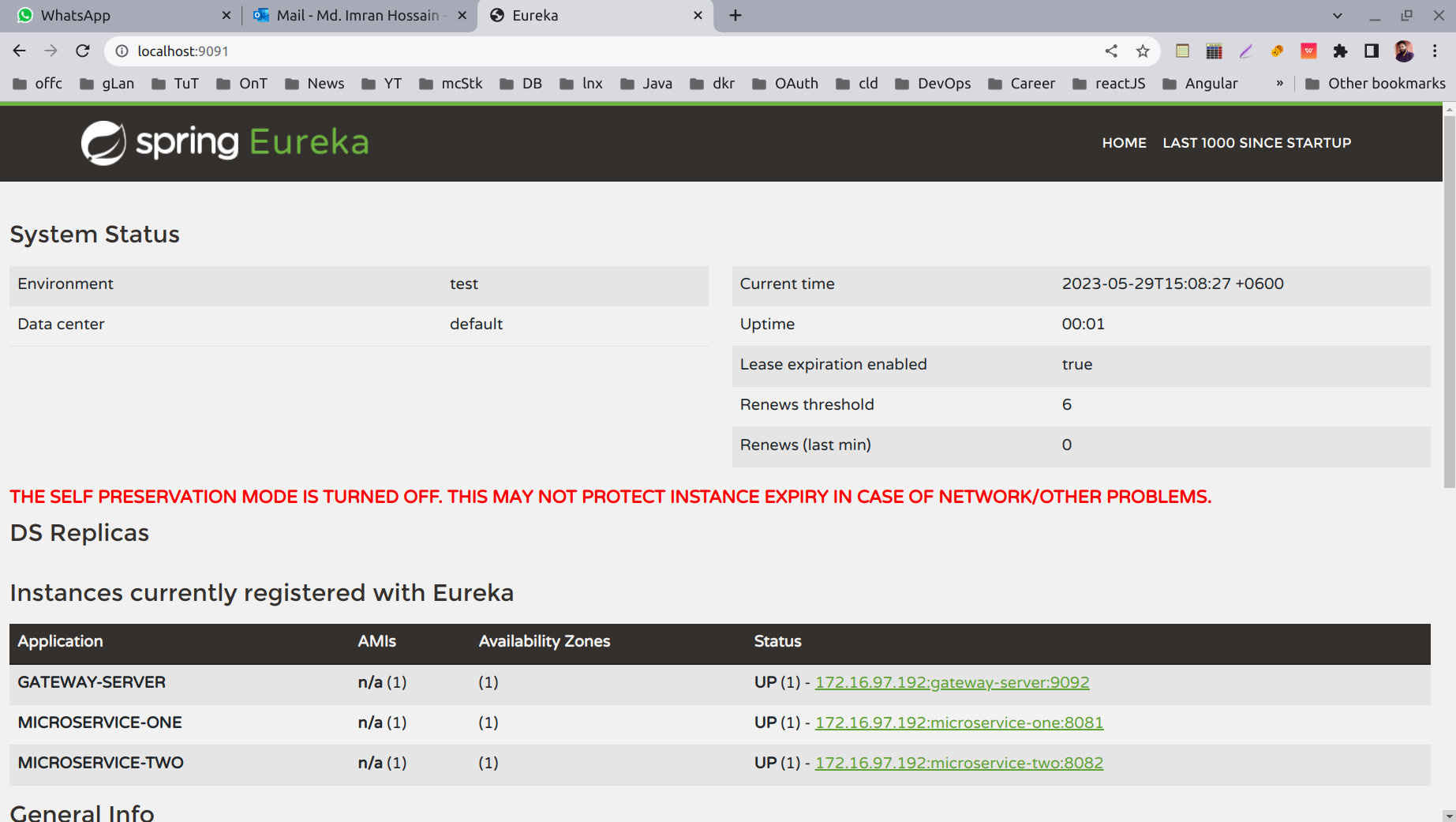
*</dependency>*

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>*

*</dependency>*



This is how look like Spring Discrovery server Dashboard, where all registered service will show with its general info like: name, ip, port, running status etc.

**Business Servcies:**

Well, a business service in Spring Framework is also known as service component and it also a spring boot project what is containe application business logic and API.

In this Quick Start Guide we build two microsrvice **“micrservice-one”** and **“microservice-two”**, there is no limit about this number for service it can be any base on your application requirement.

Spring cloud Starter for Discovery client:

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-config</artifactId>*

*</dependency>*

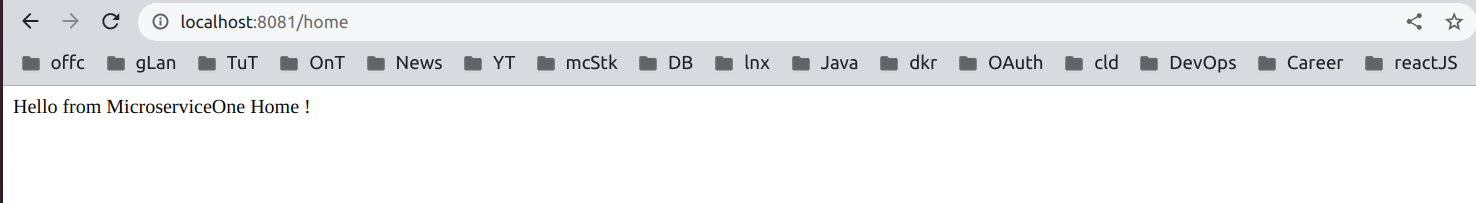
*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

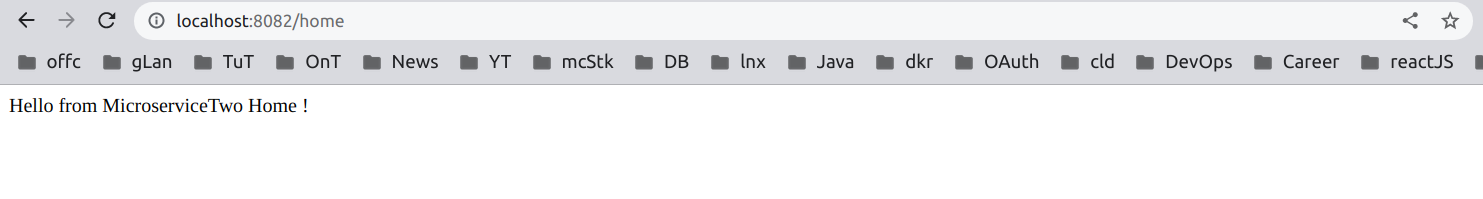
*<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>*

*</dependency>*

Microservice-one: url http://localhost:8082/home



Microservice-two: url http://localhost:8082/home



**API Gateway and Routing:**

Spring Cloud Gateway acts as a lightweight, scalable API gateway for microservices. It provides essential routing capabilities, request/response transformation, rate limiting, and authentication/authorization features. With Spring Cloud Gateway, you can implement cross-cutting concerns and enforce security policies consistently across microservices.

Spring cloud Gateway Starter:

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-config</artifactId>*

*</dependency>*

*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-gateway</artifactId>*

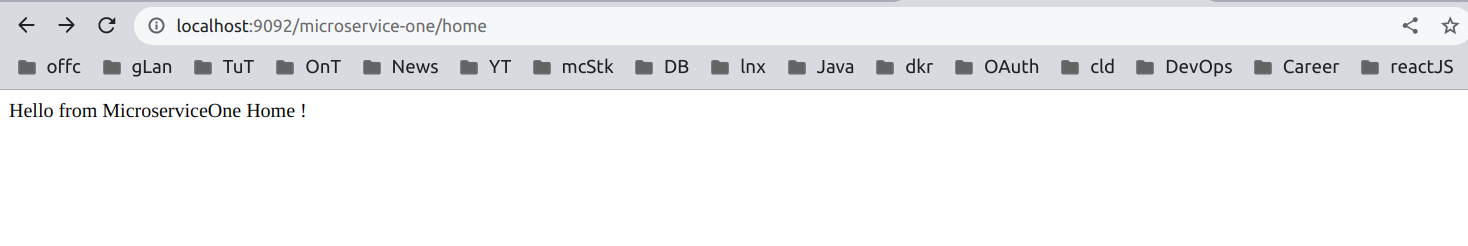
*</dependency>*

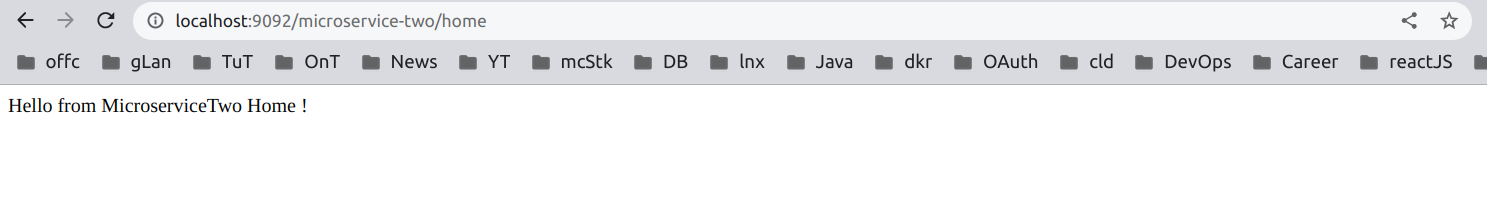
*<dependency>*

*<groupId>org.springframework.cloud</groupId>*

*<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>*

*</dependency>*





We can access all business microservice throw Gateway in a single ip and port, just append the name of service before api path.

As of now describe the fundamental component for a microservice architecture: config-server, service-registry, gateway and business services.

This is the required component of a basic microservices architecture project, to keep this project short and sample we end up with this required component only.

Some of another essential component for Moitoring, security and persistence layer describe as blow:

**Distributed Tracing and Monitoring:**

Spring Cloud Sleuth and Zipkin enable distributed tracing, allowing developers to trace requests across microservices. By collecting and correlating trace data, you can gain insights into the flow and performance of requests in complex architectures. Additionally, Spring Boot Actuator provides monitoring and management endpoints for metrics, health checks, and more.

**Data Access and Persistence:**

Spring Data offers a unified and consistent approach to accessing various data sources within microservices. It supports multiple databases, NoSQL stores, and message brokers. Spring Data JPA simplifies working with relational databases, while Spring Data MongoDB and Spring Data Redis provide seamless integration with respective data stores.

**Messaging and Event-Driven Architecture:**

Spring provides excellent support for event-driven communication patterns in microservices. With Spring Cloud Stream and Spring Integration, you can easily build event-driven architectures using messaging systems like RabbitMQ, Apache Kafka, or Apache Pulsar. This enables loose coupling, scalability, and resilience in microservices communication.

**Security and Authentication:**

Securing microservices is crucial, and Spring Security offers a comprehensive set of features for authentication, authorization, and securing endpoints. With Spring Security OAuth, you can implement robust and secure authorization workflows, allowing microservices to authenticate and interact securely.

**Testing and Deployment:**

Spring Framework provides extensive support for testing microservices. From unit tests using frameworks like JUnit and Mockito to integration tests with Spring Test, it simplifies the verification of microservices' behavior and interactions. Additionally, Spring's integration with popular deployment tools like Docker and Kubernetes streamlines the deployment and scaling of microservices.

**Conclusion:**

The Spring Framework offers a comprehensive ecosystem of tools, libraries, and features that greatly simplify the development, deployment, and management of microservices. With Spring Boot, Spring Cloud.