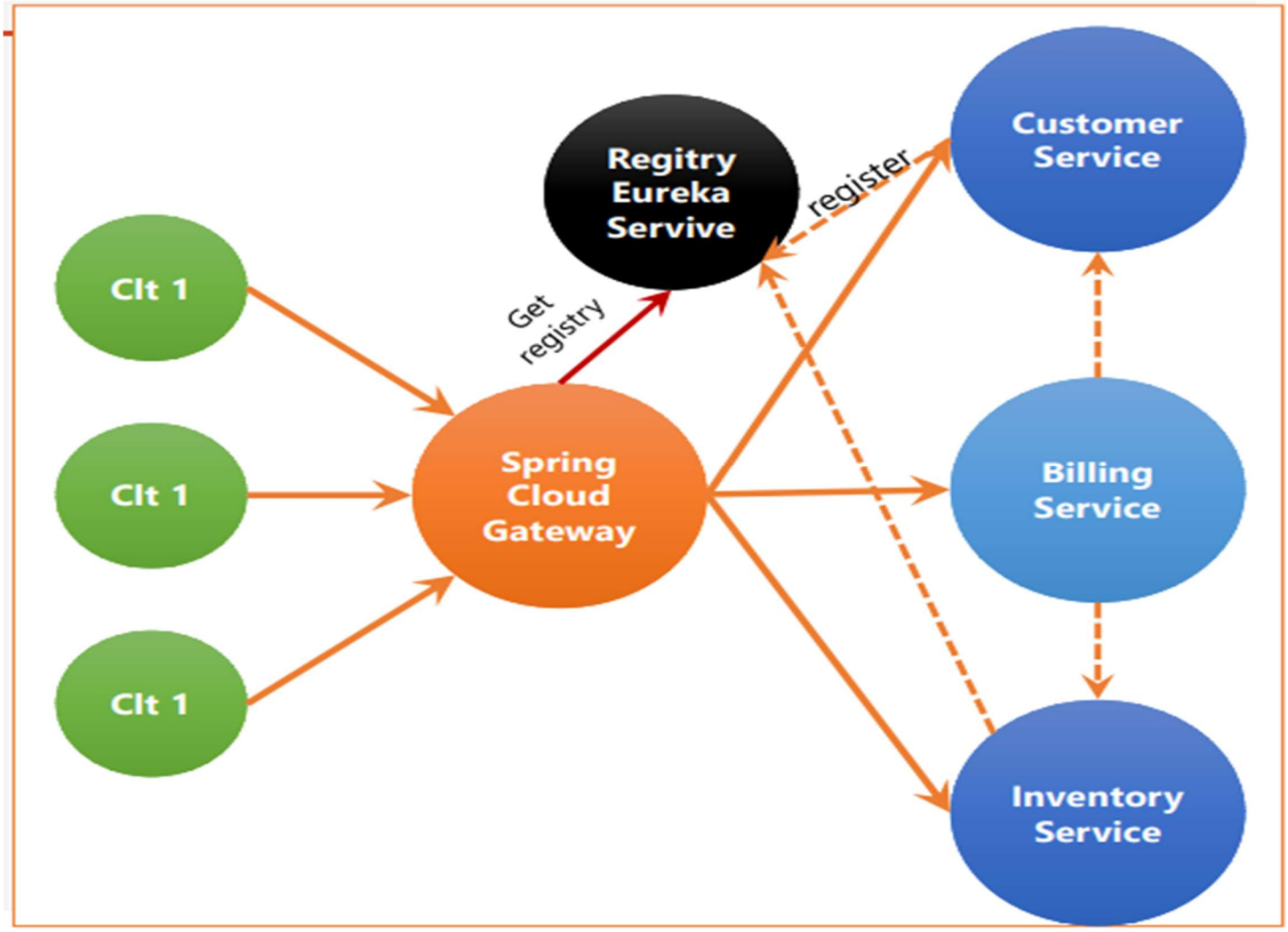
ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA



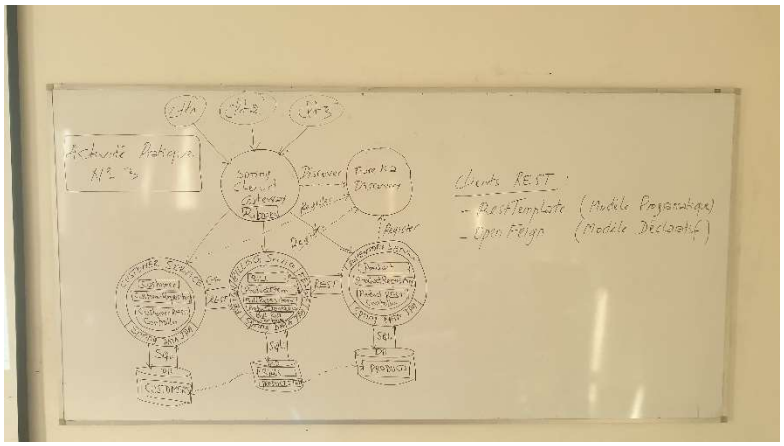
ACTIVITE PRATIQUE N° 2

ARCHITECTURES MICRO-SERVICES AVEC SPRING CLOUD

ABDERRAHMANE ETTOUNANI → IIBDCC-3 2023/2024

ENONCE

1. Créer le micro service Customer-service
   - Créer l'entité Customer
   - Créer l'interface CustomerRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
2. Créer le micro service Inventory-service
   - Créer l'entité Product
   - Créer l'interface ProductRepository basée sur Spring Data
   - Déployer l'API Restful du micro-service en utilisant Spring Data Rest
   - Tester le Micro service
3. Créer la Gateway service en utilisant Spring Cloud Gateway
   1. Tester la Service proxy en utilisant une configuration Statique basée sur le fichier application.yml
   2. Tester la Service proxy en utilisant une configuration Statique basée une configuration Java
4. Créer l'annuaire Registry Service basé sur NetFlix Eureka Server
5. Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micro services enregistrés dans l'annuaire Eureka Server
6. Créer Le service Billing-Service en utilisant Open Feign pour communiquer avec les services Customer-service et Inventory-service
7. Créer un client Angular qui permet d'afficher une facture



1 : MICRO-SERVICE CUSTOMER-SERVICE

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @ToString
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
}
```

```
@SpringBootApplication
public class CustomerApplication {

    public static void main(String[] args) {
        SpringApplication.run(CustomerApplication.class, args);
    }
    @Bean
    CommandLineRunner start(CustomerRepository customerRepository) {
        return args -> {
            customerRepository.save(new Customer(null, "Ettounani",
"tounani@gmail.com"));
            customerRepository.save(new Customer(null, "youssef",
"youssef@gmail.com"));
            customerRepository.save(new Customer(null, "Abderrahmane",
"abdo@gmail.com"));
            customerRepository.findAll().forEach(System.out::println);
        };
    }
}
```

```
server.port=8081
spring.application.name=customer-service
spring.datasource.url=jdbc:h2:mem:customer-db
spring.cloud.discovery.enabled=true
```

```
@RepositoryRestController
public interface CustomerRepository extends JpaRepository<Customer, Long> {}
```

2 : MICRO-SERVICE INVENTORY-SERVICE

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @ToString
public class Product {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
    private double quantite;
}
```

```
@RepositoryRestController
public interface ProductRepository extends JpaRepository<Product, Long> {}
```

```
@SpringBootApplication
public class InventoryApplication {
    public static void main(String[] args) {
        SpringApplication.run(InventoryApplication.class, args);
    }
@Bean
    CommandLineRunner start(ProductRepository productRepository) {
        return args -> {
                productRepository.save( new Product(null,"Computer", 8000, 3));
                productRepository.save( new Product(null,"Printer", 2000, 2));
                productRepository.save( new Product(null,"Smartphone", 6000, 5));
            productRepository.findAll().forEach(System.out::println);
        };
    }
}
```

```
server.port=8082
spring.application.name=product-service
spring.datasource.url=jdbc:h2:mem:product-db
spring.cloud.discovery.enabled=true
```

3 : GATEWAY SERVICE

```
@SpringBootApplication
public class GatewayyApplication {

    public static void main(String[] args) {
        SpringApplication.run(GatewayyApplication.class, args);
    }
    //@Bean
    public RouteLocator myRoutes(RouteLocatorBuilder builder) {
        return builder.routes()
                .route(p -> p.path("/customers/**").uri("lb://CUSTOMER-SERVICE"))
                .route(p -> p.path("/products/**").uri("lb://PRODUCT-SERVICE"))
                .build();
    }

    @Bean
    public DiscoveryClientRouteDefinitionLocator
definitionLocator(ReactiveCompositeDiscoveryClient discoveryClient,
DiscoveryLocatorProperties properties ) {

        return new DiscoveryClientRouteDefinitionLocator(discoveryClient, properties);
    }
}
```

```
server.port=8888
spring.application.name=gateway-service
spring.cloud.discovery.enabled=true
```

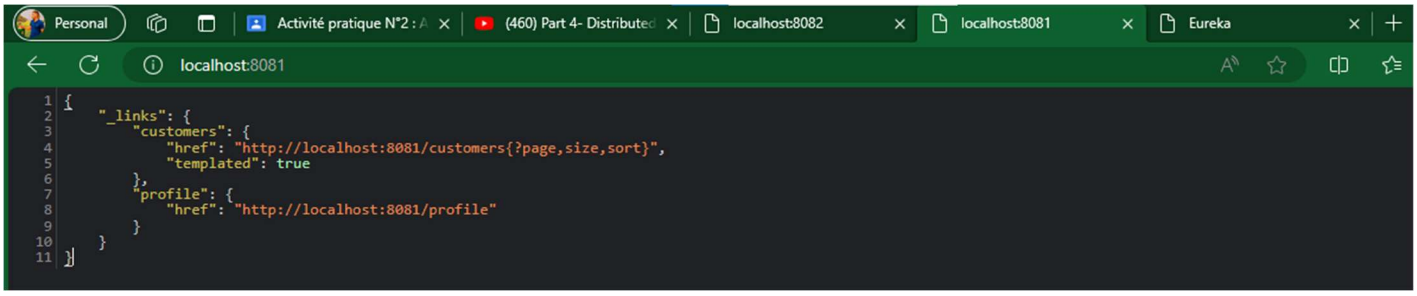
```
spring:
  cloud:
    gateway:
      routes:
        - id: r1
          uri: http://localhost:8081/
          predicates:
            - Path=/customers/**
        - id: r2
          uri: http://localhost:8082/
          predicates:
            - Path=/products/**
```

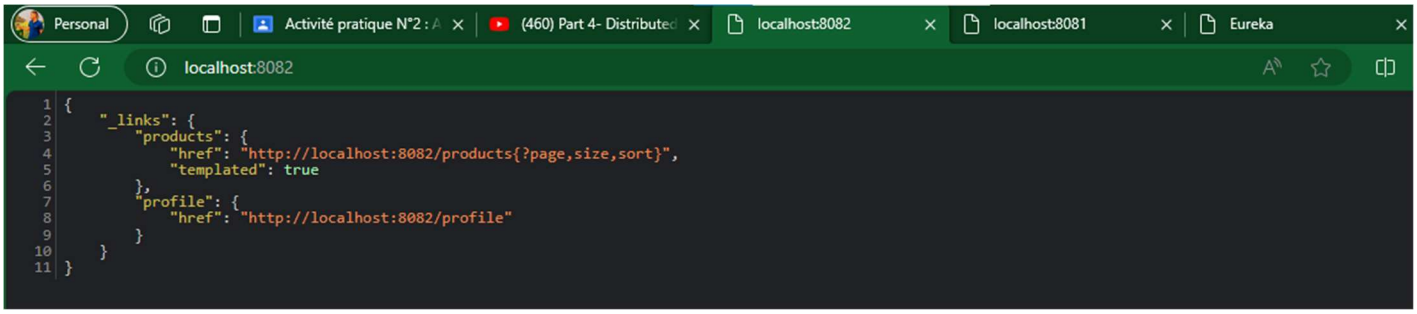
4 : NETFLIX EUREKA SERVER

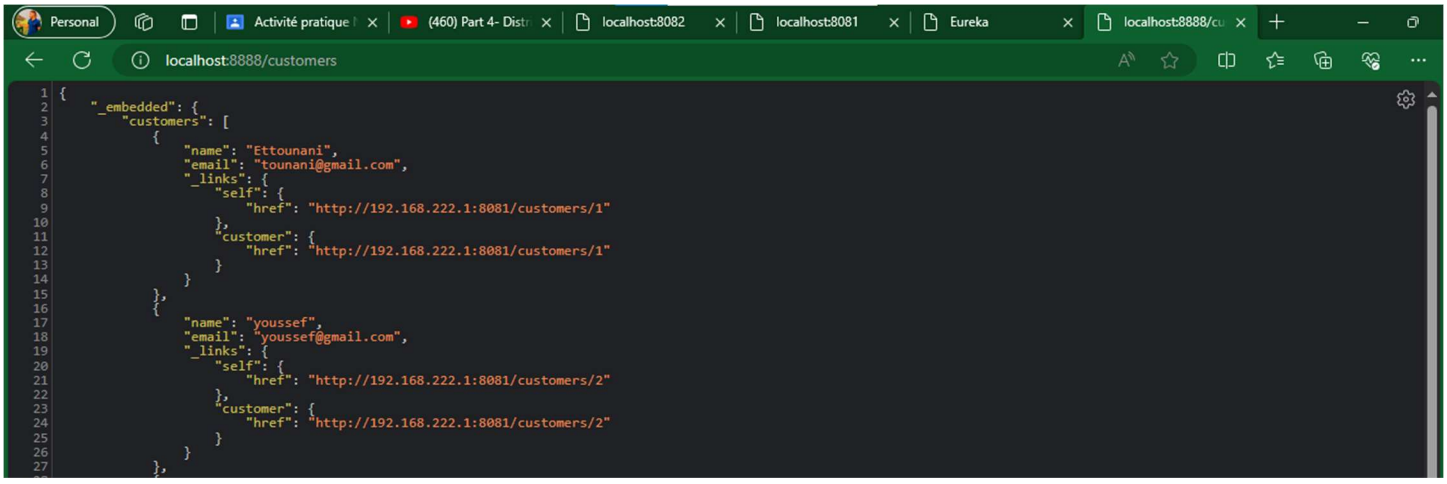
```java
@SpringBootApplication
@EnableEurekaServer // Enable eureka server
public class DiscoveryApplication {

    public static void main(String[] args) {
        SpringApplication.run(DiscoveryApplication.class, args);
    }

}
```

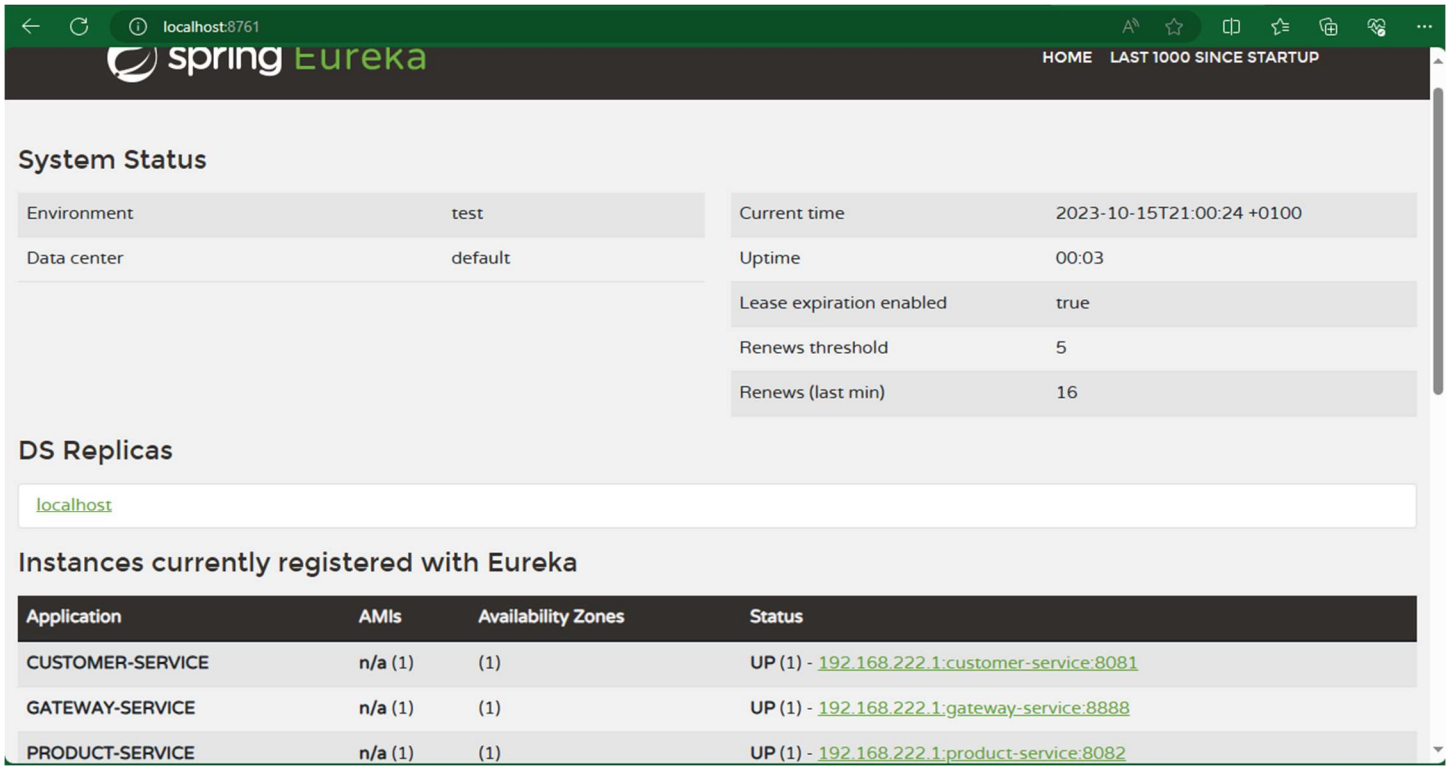
```yaml
server.port=8761
eurika.client.fetch-registry=false
eurika.client.register-with-eurika=false
```

5 : TEST









Selected dependencies

- **Spring Web** : Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
- **Spring Data JPA** : Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- **H2 Database** : Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
- **Rest Repositories** : Exposing Spring Data repositories over REST via Spring Data REST.
- **Lombok** : Java annotation library which helps to reduce boilerplate code.
- **Spring Boot DevTools** : Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- **Eureka Discovery Client** : a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.
- **OpenFeign** : Declarative REST Client. OpenFeign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.
- **Spring HATEOAS** : Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC.

6 : BILLING-SERVICE

```
@Entity
public class Bill {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date billingDate;
    @OneToMany(mappedBy = "bill")
    private Collection<ProductItem> productsItems;
    private Long customerID;
    @Transient
    private Customer customer;
}
```

```
@Entity
public class ProductItem {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private double quantity;
    private double price;
    private Long productId;
    @ManyToOne
    private Bill bill;
    @Transient
    private Product product;
}
```

```
@Data
public class Customer {
    private Long id;
    private String name;
    private String email;
}
```

```
@Data
public class Product {
    private Long id;
    private String name;
    private double price;
    private double quantity;
}
```

```
@FeignClient(name = "CUSTOMER-SERVICE")
public interface CustomerRestClient {
    @GetMapping(path = "/customers/{id}")
    public Customer findCustomerById(Long id);

}
```

```
@FeignClient(name = "PRODUCT-SERVICE")
public interface ProductItemRestClient {
    @GetMapping(path = "/products")
    PagedModel<Product> findAll(@RequestParam(value = "page") int
page,@RequestParam(value="size") int size);
    @GetMapping(path = "/products/{id}")
    Product findProductById(@RequestParam(value = "id") Long id);

}
```

```
@RepositoryRestResource
public interface BillRepository  extends JpaRepository <Bill, Long>{

}
```

```
@RepositoryRestResource
public interface ProductRepository  extends JpaRepository <ProductItem, Long>{
    public Collection<ProductItem> findByBillId(Long id);
}
```

```
@SpringBootApplication
public class BellingApplication {

    public static void main(String[] args) {
        SpringApplication.run(BellingApplication.class, args);
    }

}
```