**Département Mathématiques et Informatique**

**Filière :**

**« Ingénierie Informatique : Big Data et Cloud Computing »**

**II-BDCC**

**Mise en œuvre d'une architecture micro-services**

Réalisé par :

Soukaina EL KAMOUNI

**Année Universitaire : 2022-2023**

# Consignes :

Suivre les vidéos Bonnes pratiques de Architectures micro-services :

1. **Vidéo 1** : Concepts de bases
2. **Vidéo 2 à 5** : Mise en œuvre d'une application distribuée basée sur deux micro-services en utilisant les bonnes pratiques :

* Couches DA0, Service, Web, DTO
* Utilisation de MapStruct pour le mapping entre les objet Entités et DTO
* Génération des API-DOCS en utilisant SWAGGER3 (Open API)
* Communication entre micro-services en utilisant OpenFeign
* Spring Cloud Gateway
* Eureka Discovery Service

# Table of Contents

[Consignes : 2](#_Toc118065585)

[1. Project Customer Service: 3](#_Toc118065586)

[a. Entities: 3](#_Toc118065587)

[b. Repository: 3](#_Toc118065588)

[c. DTO: 3](#_Toc118065589)

[d. Services: 3](#_Toc118065590)

[e. Mappers: 4](#_Toc118065591)

[f. Web Service: 5](#_Toc118065592)

[g. Main App: 5](#_Toc118065593)

[h. Résultats: 5](#_Toc118065594)

[2. Project Billing Server: 8](#_Toc118065595)

[a. Entities: 8](#_Toc118065596)

[b. DTO: 8](#_Toc118065597)

[c. Repository: 8](#_Toc118065598)

[d. Services: 9](#_Toc118065599)

[e. Web Service: 10](#_Toc118065600)

[f. Mapper: 11](#_Toc118065601)

[g. Open Feign: 11](#_Toc118065602)

[h. Main App: 11](#_Toc118065603)

[i. Résultats: 11](#_Toc118065604)

[3. Project Eureka: 14](#_Toc118065605)

[a. Configuration: 14](#_Toc118065606)

[b. Résultat: 15](#_Toc118065607)

[4. La Gateway: 16](#_Toc118065608)

[a. La configuration: 16](#_Toc118065609)

[b. Résultat: 16](#_Toc118065610)

# Project Customer Service:

## Entities:

* Customer:

@Entity  
@Data @NoArgsConstructor @AllArgsConstructor  
public class Customer {  
 @Id  
 private String id;  
 private String name;  
 private String email;  
}

## Repository:

public interface CustomerRepository extends JpaRepository<Customer,String> {}

## DTO:

* Customer Request DTO:

@Data  
@NoArgsConstructor @AllArgsConstructor  
public class CustomerRequestDTO {  
 private String id;  
 private String name;  
 private String email;  
}

* Customer Response DTO:

@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class CustomerResponseDTO {  
 private String id;  
 private String name;  
 private String email;  
}

## Services:

* Interface:

public interface CustomerService {  
 CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO);  
 CustomerResponseDTO getCustomer(String id);  
 CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO);  
 List<CustomerResponseDTO> listCustomers();  
}

* Implementation:

@Service  
@Transactional  
public class CustomerServiceImpl implements CustomerService {  
 private CustomerRepository customerRepository;  
 private CustomerMapper customerMapper;  
  
 public CustomerServiceImpl(CustomerRepository customerRepository, CustomerMapper customerMapper) {  
 this.customerRepository = customerRepository;  
 this.customerMapper = customerMapper;  
 }  
  
 @Override  
 public CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO) {  
  
 /\*  
 Mapping Objet Objet  
 \*/  
 Customer customer=customerMapper.customerRequestDTOToCustomer(customerRequestDTO);  
 Customer saveCustomer=customerRepository.save(customer);  
 CustomerResponseDTO customerResponseDTO = customerMapper.customerToCustomerResponseDTO(customer);  
 return customerResponseDTO;  
 }  
  
 @Override  
 public CustomerResponseDTO getCustomer(String id) {  
 Customer customer=customerRepository.findById(id).get();  
 return customerMapper.customerToCustomerResponseDTO(customer);  
 }  
  
 @Override  
 public CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO) {  
 Customer customer=customerMapper.customerRequestDTOToCustomer(customerRequestDTO);  
 Customer updatedCustomer=customerRepository.save(customer);  
 return customerMapper.customerToCustomerResponseDTO(updatedCustomer);  
 }  
  
 @Override  
 public List<CustomerResponseDTO> listCustomers() {  
 List<Customer> customers=customerRepository.findAll();  
 List<CustomerResponseDTO> customerResponseDTOS=customers.stream()  
 .map(cust->customerMapper.customerToCustomerResponseDTO(cust))  
 .collect(Collectors.*toList*());  
 return customerResponseDTOS;  
 }  
}

## Mappers:

@Mapper(componentModel = "spring")  
public interface CustomerMapper {  
 CustomerResponseDTO customerToCustomerResponseDTO(Customer customer);  
 Customer customerRequestDTOToCustomer(CustomerRequestDTO customerRequestDTO);  
}

## Web Service:

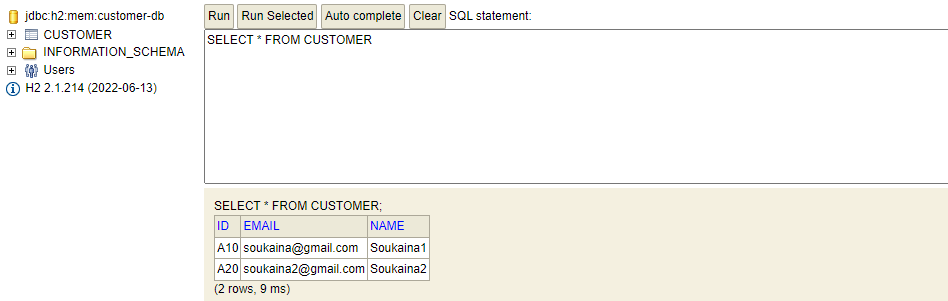
@RestController  
@RequestMapping(path = "/api")  
@Slf4j  
public class CustomerRestAPI {  
 private CustomerService customerService;  
  
 public CustomerRestAPI(CustomerService customerService) {  
 this.customerService = customerService;  
 }  
  
 @GetMapping(path = "/customers")  
 public List<CustomerResponseDTO> customers(){  
 return customerService.listCustomers();  
 }  
  
 @PostMapping(path = "/customers")  
 public CustomerResponseDTO save(@RequestBody CustomerRequestDTO customerRequestDTO){  
 customerRequestDTO.setId(UUID.*randomUUID*().toString());  
 return customerService.save(customerRequestDTO);  
 }  
  
 @GetMapping(path = "/customers/{id}")  
 public CustomerResponseDTO getCustomer(@PathVariable String id){  
 return customerService.getCustomer(id);  
 }  
}

## Main App:

@SpringBootApplication  
public class CustomerServiceApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(CustomerServiceApplication.class, args);  
 }  
  
 @Bean  
 CommandLineRunner start(CustomerService customerService){  
 return args -> {  
 customerService.save(new CustomerRequestDTO("A10", "Soukaina1","soukaina@gmail.com"));  
 customerService.save(new CustomerRequestDTO("A20", "Soukaina2","soukaina2@gmail.com"));  
 };  
 }  
}

## Résultats:

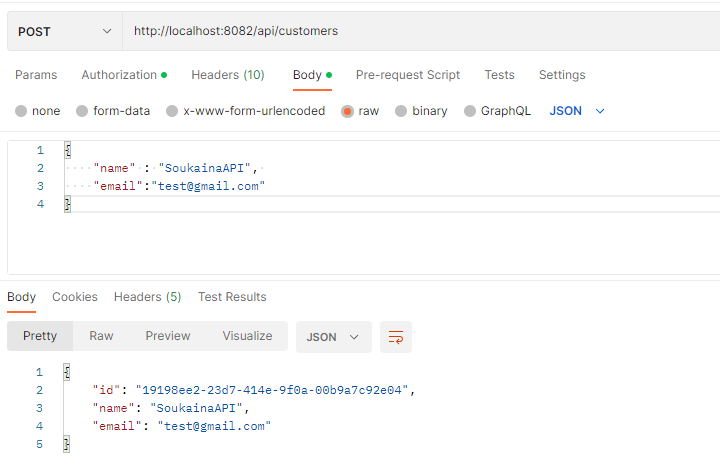
* H2 Database:



* Local Host:



* Sur Postman pour tester le POST :



On voit bien que le nouveau client est ajouté :



* On peut consulter la documentation du API comme suivant :



# Project Billing Server:

## Entities:

* Customer:

@Data @NoArgsConstructor @AllArgsConstructor  
public class Customer {  
 private String id;  
 private String name;  
 private String email;  
}

* Invoice:

@Entity  
@Data @AllArgsConstructor @NoArgsConstructor  
public class Invoice {  
 @Id  
 private String id;  
 private Date date;  
 private BigDecimal amount;  
 private String customerID;  
 @Transient  
 private Customer customer;  
}

## DTO:

* Invoice Request DTO:

@Data @NoArgsConstructor @AllArgsConstructor  
public class InvoiceRequestDTO {  
 private BigDecimal amount;  
 private String customerID;  
}

* Invoice Respond DTO:

@Data @AllArgsConstructor @NoArgsConstructor  
public class InvoiceResponseDTO {  
 private String id;  
 private Date date;  
 private BigDecimal amount;  
 private Customer customer;  
}

## Repository:

public interface InvoiceRepository extends JpaRepository<Invoice,String> {  
 List<Invoice> findByCustomerID(String customerId);  
}

## Services:

* Interface:

public interface InvoiceService {  
 InvoiceResponseDTO save(InvoiceRequestDTO invoiceRequestDTO) throws CustomerNotFoundException;  
 InvoiceResponseDTO getInvoice(String invoiceId);  
 List<InvoiceResponseDTO> invoiceByCustomerId(String customerId);  
 List<InvoiceResponseDTO> allInvoices();  
}

* Implementation:

@Service  
@Transactional  
public class InvoiceServiceImpl implements InvoiceService {  
  
 private InvoiceRepository invoiceRepository;  
 private InvoiceMapper invoiceMapper;  
 private CustomerRestClient customerRestClient;  
  
 public InvoiceServiceImpl(InvoiceRepository invoiceRepository, InvoiceMapper invoiceMapper, CustomerRestClient customerRestClient) {  
 this.invoiceRepository = invoiceRepository;  
 this.invoiceMapper = invoiceMapper;  
 this.customerRestClient = customerRestClient;  
 }  
  
 @Override  
 public InvoiceResponseDTO save(InvoiceRequestDTO invoiceRequestDTO){  
 Customer customer=null;  
 try {  
 customer = customerRestClient.getCustomer(invoiceRequestDTO.getCustomerID());  
 } catch (Exception e) {  
 throw new CustomerNotFoundException("Customer Not Found");  
 }  
 Invoice invoice = invoiceMapper.fromInvoiceRequestDTO(invoiceRequestDTO);  
 invoice.setId(UUID.*randomUUID*().toString());  
 invoice.setDate(new Date());  
 Invoice saveInvoice = invoiceRepository.save(invoice);  
 saveInvoice.setCustomer(customer);  
 return invoiceMapper.fromInvoice(saveInvoice);  
 }  
  
 @Override  
 public InvoiceResponseDTO getInvoice(String invoiceId) {  
 Invoice invoice=invoiceRepository.findById(invoiceId).get();  
 Customer customer=customerRestClient.getCustomer(invoice.getCustomerID());  
 invoice.setCustomer(customer);  
 return invoiceMapper.fromInvoice(invoice);  
 }  
  
 @Override  
 public List<InvoiceResponseDTO> invoiceByCustomerId(String customerId) {  
 List<Invoice> invoices=invoiceRepository.findByCustomerID(customerId);  
 for (Invoice invoice: invoices){  
 Customer customer=customerRestClient.getCustomer(invoice.getCustomerID());  
 invoice.setCustomer(customer);  
 }  
 return invoices.stream()  
 .map(invoice -> invoiceMapper.fromInvoice(invoice))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<InvoiceResponseDTO> allInvoices() {  
 List<Invoice> invoices = invoiceRepository.findAll();  
 for (Invoice invoice: invoices){  
 Customer customer=customerRestClient.getCustomer(invoice.getCustomerID());  
 invoice.setCustomer(customer);  
 }  
 return invoices.stream().map(invoice -> invoiceMapper.fromInvoice(invoice)).collect(Collectors.*toList*());  
 }  
}

## Web Service:

@RestController  
@RequestMapping(path = "/api")  
public class InvoiceRestController {  
 private InvoiceService invoiceService;  
  
 public InvoiceRestController(InvoiceService invoiceService) {  
 this.invoiceService = invoiceService;  
 }  
  
 @GetMapping(path = "/invoices/{id}")  
 public InvoiceResponseDTO getInvoice(@PathVariable(name = "id") String invoiceId){  
 return invoiceService.getInvoice(invoiceId);  
 }  
  
 @GetMapping(path = "/invoicesByCustomer/{customerId}")  
 public List<InvoiceResponseDTO> getInvoicesByCustomer(@PathVariable String customerId){  
 return invoiceService.invoiceByCustomerId(customerId);  
 }  
  
 @PostMapping(path = "/invoices")  
 private InvoiceResponseDTO save(@RequestBody InvoiceRequestDTO invoiceRequestDTO){  
 return invoiceService.save(invoiceRequestDTO);  
 }  
  
 @GetMapping(path = "/invoices")  
 public List<InvoiceResponseDTO> allInvoices(){  
 return invoiceService.allInvoices();  
 }  
  
 @ExceptionHandler(Exception.class)  
 public ResponseEntity<String> exceptionHandler(Exception e){  
 return new ResponseEntity<>(e.getMessage(), HttpStatus.*INTERNAL\_SERVER\_ERROR*);  
 }  
}

## Mapper:

@Mapper(componentModel = "spring")  
public interface InvoiceMapper {  
 Invoice fromInvoiceRequestDTO(InvoiceRequestDTO invoiceRequestDTO);  
 InvoiceResponseDTO fromInvoice(Invoice invoice);  
}

## Open Feign:

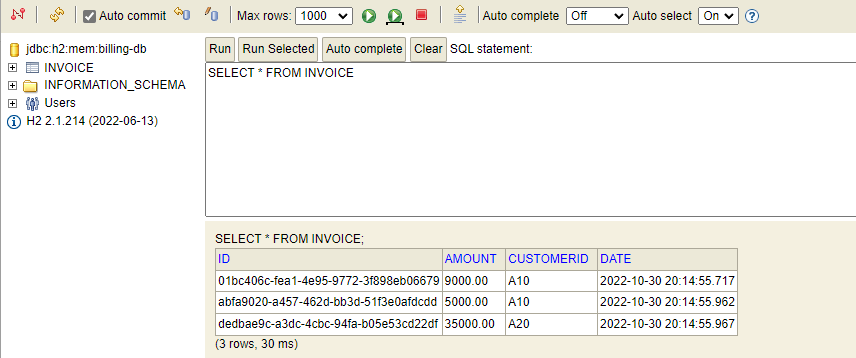
@FeignClient(name = "CUSTOMER-SERVICE")  
public interface CustomerRestClient {  
 @GetMapping(path = "/api/customers/{id}")  
 Customer getCustomer(@PathVariable(name = "id") String id);  
  
 @GetMapping(path = "/api/customers")  
 List<Customer> allCustomers();  
}

## Main App:

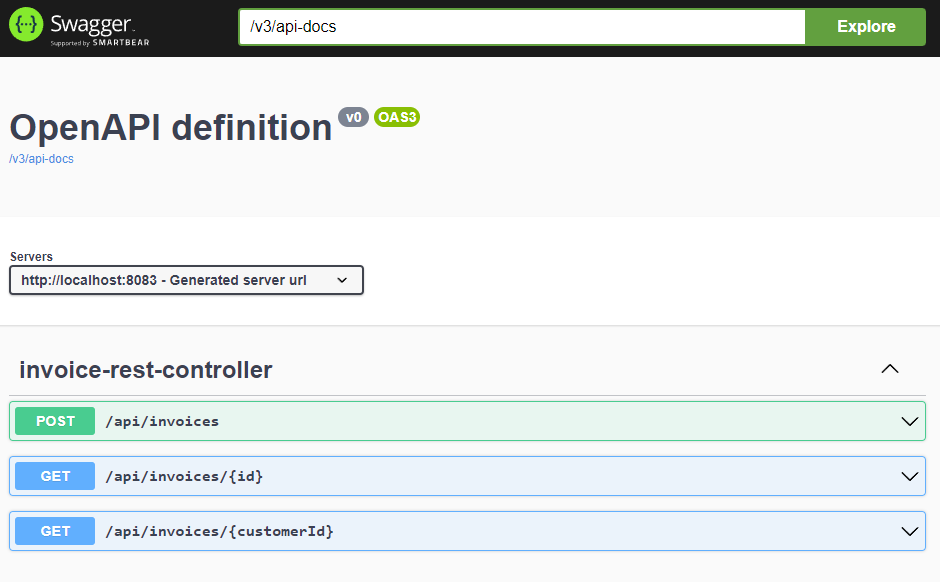
@SpringBootApplication  
@EnableFeignClients  
public class BilingServiceApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(BilingServiceApplication.class, args);  
 }  
  
 @Bean  
 CommandLineRunner commandLineRunner(InvoiceService invoiceService){  
 return args -> {  
 invoiceService.save(new InvoiceRequestDTO(BigDecimal.*valueOf*(9000), "A10"));  
 invoiceService.save(new InvoiceRequestDTO(BigDecimal.*valueOf*(5000), "A10"));  
 invoiceService.save(new InvoiceRequestDTO(BigDecimal.*valueOf*(35000), "A20"));  
 };  
 }  
}

## Résultats:

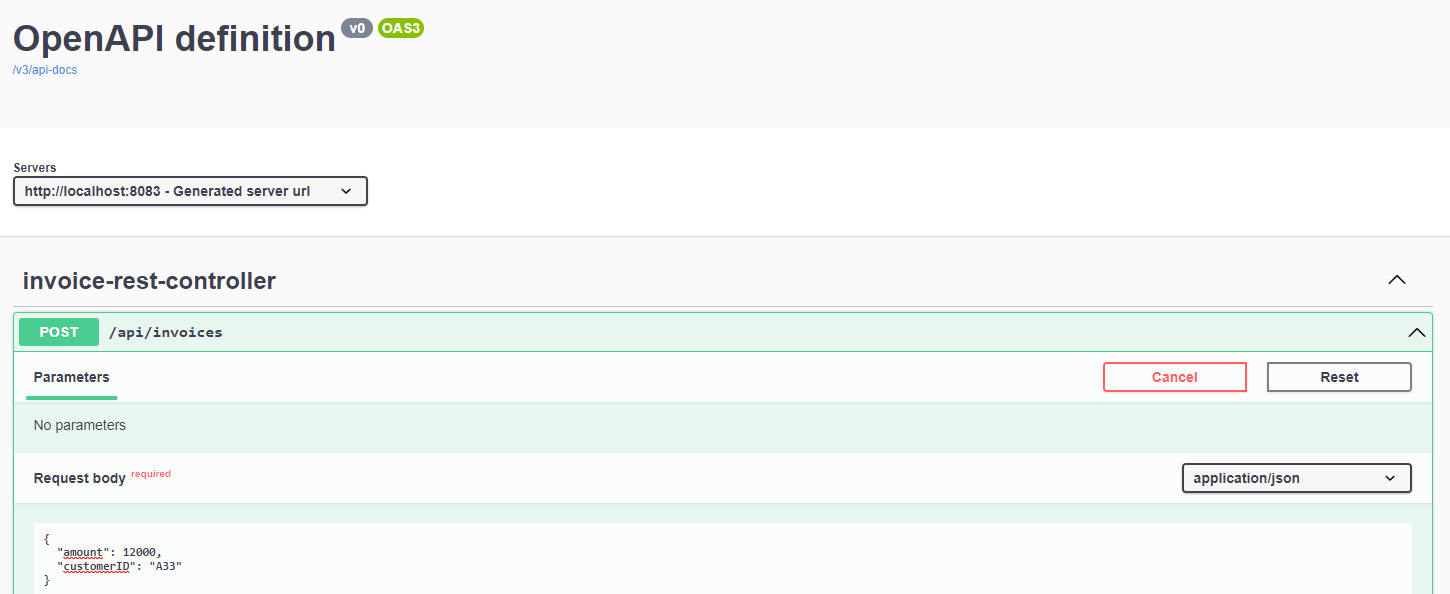
* H2 Database:

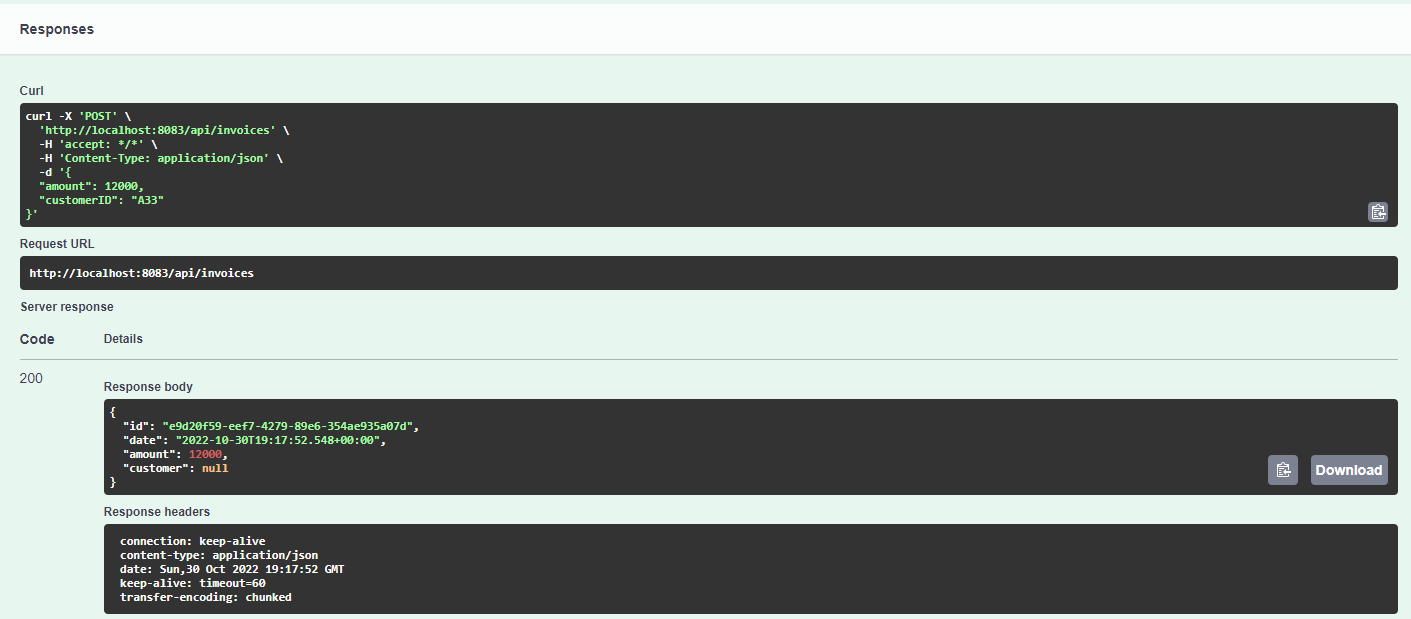


* Swagger API documentation:



* Testing the POST:





* En consultant le local host :





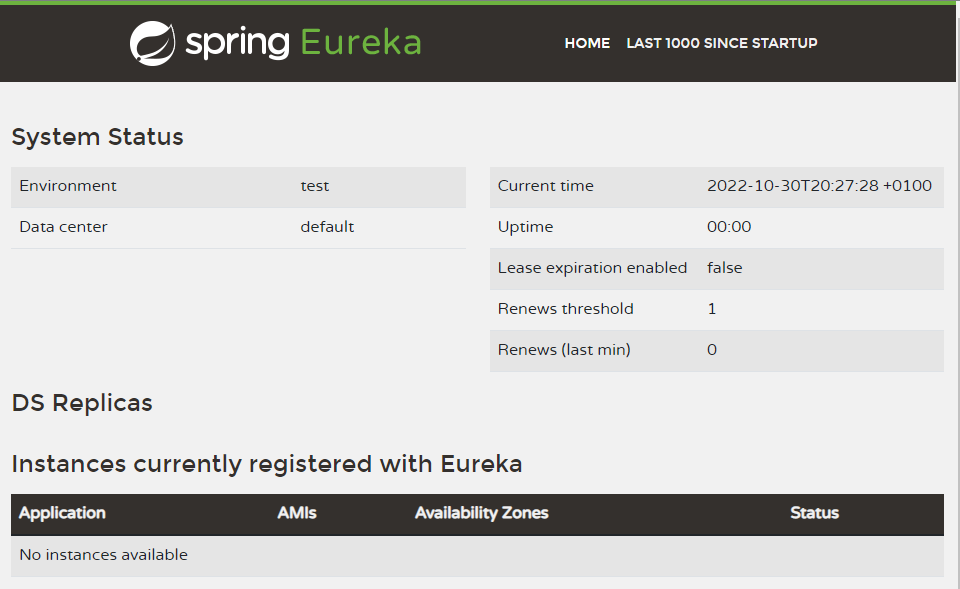
# Project Eureka:

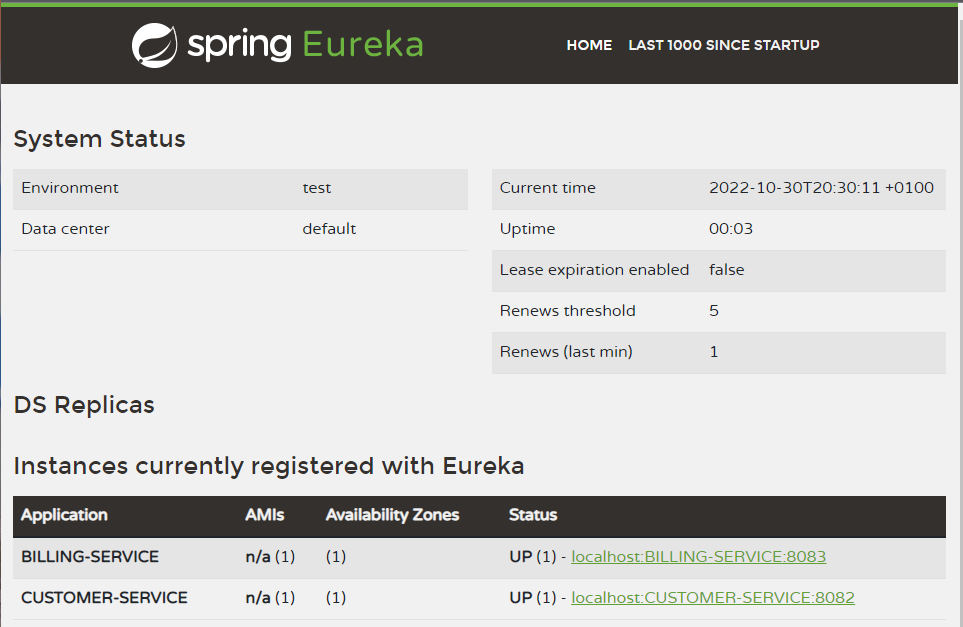
## Configuration:

@SpringBootApplication  
@EnableEurekaServer  
public class EurekaServiceApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(EurekaServiceApplication.class, args);  
 }  
  
}

## Résultat:

Grâce à Eureka, nous pouvons voir chaque service en marche ainsi que savoir son adresse, pour l’utiliser directement.



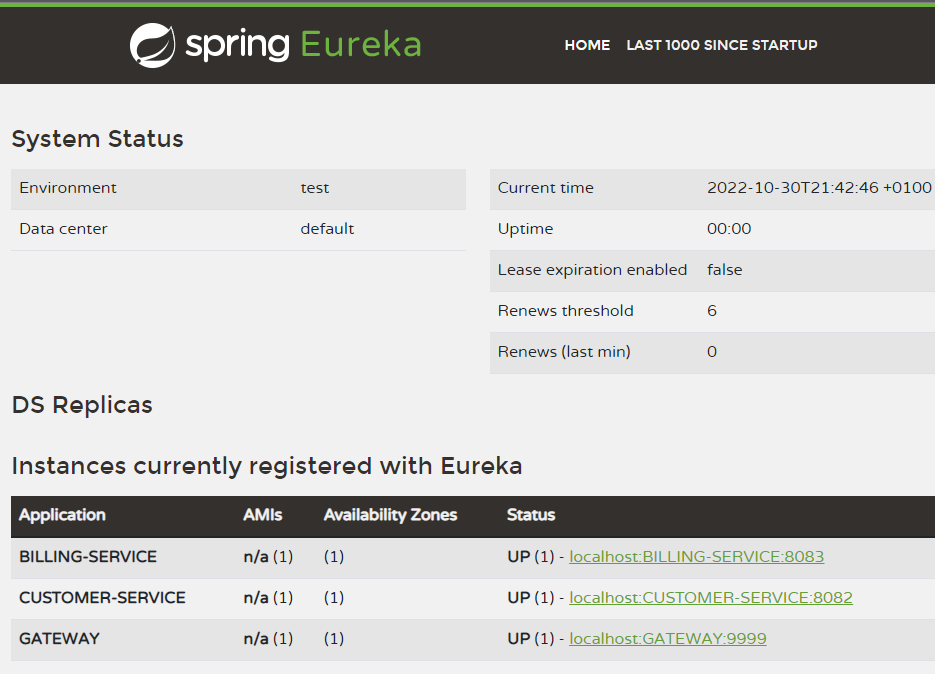


# La Gateway:

## La configuration:

@SpringBootApplication  
public class GatewayApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(GatewayApplication.class, args);  
 }  
  
 @Bean  
 DiscoveryClientRouteDefinitionLocator discoveryClientRouteDefinitionLocator(  
 ReactiveDiscoveryClient reactiveDiscoveryClient, DiscoveryLocatorProperties discoveryLocatorProperties  
 ){  
 return new DiscoveryClientRouteDefinitionLocator(reactiveDiscoveryClient,discoveryLocatorProperties);  
 }  
}

## Résultat:



On peut accéder à tous les services disponible à partir de la Gateway :





GitHub Link: <https://github.com/SoukainaElkm/Syst-Distrib-et-Big-Data>