

Examen Architecture Distribuée JEE

Durée : 1h30 | Documents non autorisés | Annexes Fournies avec l'épreuve

Professeurs : Pr. F.Z. BADRI TIJANE, Pr. K. AHAI DOUS Pr. M. EL BAKKALI, Pr. M. YOUSSEFI

On souhaite développer un système distribué de certification électronique de chèques. L'objectif est de permettre à un commerçant de saisir des informations via un système de reconnaissance optique. Les informations du chèque d'un client (numéro, RIB et le montant) sont envoyées à un service central déployé dans la banque centrale du Maroc. Ce service se charge d'envoyer une requête vers la banque du client. La banque devrait vérifier le solde du compte et certifier ce chèque en réservant le montant de la requête pour ce chèque. Dans le cas favorable, le commerçant reçoit un message comme quoi le chèque est certifié. Ce qui lui permet d'accepter le chèque qui devient certifié numériquement.

Le système distribué se compose de 3 micro-services :

- **Commerçant-Service** : Ce service gère des informations sur des chèques. Un chèque est défini par son id, le numéro de chèque, le code de la banque, le numéro de compte, le nom du client, le montant et un attribut qui indique si le chèque est certifié ou non
- **Banque-Centrale-Service** : Ce micro-service permet de gérer des informations sur les agences bancaires. Chaque Agence Bancaire est définie par son id, son nom, sa ville, l'adresse URL du web service de la banque.
- **Agence-Bancaire-Service** : Ce micro-service gère des comptes. Chaque compte peut subir plusieurs opérations (DEBIT, CREDIT, CERTIFICATION_CHEQUE). Un compte est défini par son numéro, le solde, la référence du client, son type (COURANT, EPARGNE). Une opération est définie par son id, sa date, son montant, le numéro de chèque, son type et une description.

Les données des micro-services sont stockées dans des bases de données de type H2. La messagerie asynchrone entre les micro-services se fait en utilisant le Broker KAFKA. La communication synchrone entre les micro-services fonctionnels se fait en utilisant Open Feign. La partie back end de l'application est basée sur Spring Cloud et la partie Frontend Web est basée sur Angular ou React.

Cette épreuve se compose de trois parties :

- Partie QCM** qui concerne des questions de concepts qui n'est pas liée à ce problème
- Partie Conception qui est liée à ce problème
- Partie Implémentation qui est liée à ce problème

Des annexes contenant des extraits de codes est fournie à partir de la page 7 jusqu'à la page 11. Ces annexes contiennent des exemples de codes sources qui n'ont aucun lien avec le problème traité dans cet examen. Toutefois, quelques annexes peuvent vous servir pour des consultations syntaxiques

Travail demandé :

A. Partie QCM :

- Barème : 1 pour une bonne réponse et 0 pour une mauvaise réponse
- Mode de réponse : Les réponses doivent être portées directement sur la feuille de réponse de l'examen en respectant le format suivant par exemple :
 - Question 1 : A
 - Question 8 : B , C

Ce qui signifie que vous avez choisi la réponse A pour la question 1 et les réponses B et C pour la question 8.

1. Quelles sont les différentes solutions qui permettent de faire communiquer des systèmes Distribués (Plusieurs réponses sont possibles) ?

- | | |
|-----------------|----------|
| A. Web services | D. CORBA |
| B. RMI | E. JMS |
| C. JPA | F. GRPC |

2. Quelle est la spécification JEE qui permet d'implémenter les web services basés sur le protocole SOAP ?

- | | |
|----------|-----------|
| A. JAXRS | C. JMS |
| B. JAXWS | D. SoapWS |

3. Quelle est la spécification JEE qui permet d'implémenter les web services basés sur RESTFul ?

- | | |
|----------|-----------|
| E. JAXRS | G. JMS |
| F. JAXWS | H. SoapWS |

4. Dans les systèmes distribués, quelques les solutions qui permettent aux applications distribuées de communiquer avec des messages de manière asynchrone (Plusieurs réponses sont possibles) ?

- | | |
|---------|-------------|
| A. SOAP | D. RabbitMQ |
| B. RMI | E. KAFKA |
| C. JMS | |

5. Quelle est la définition qui est correcte pour le WSDL ?

- A. Le WSDL est un document XML, qui permet qui permet de sécuriser un Web service
- B. Le WSDL est un document XML utilisant les schémas XML, qui permet de faire la description de l'interface d'un web services
- C. Le WSDL est un document JSON, qui permet de faire la description de l'interface d'un web services
- D. Le WSDL est un document PDF, qui permet de faire la description de l'interface d'un web services

6. Pour implémenter un web service basé sur JAXWS, quelle est l'annotation à utiliser sur la classe du web service ?

- | | |
|----------------|-----------------|
| A. @Service | C. @SoapService |
| B. @WebService | D. @Path |

7. Dans Spring cloud, quel est le service qui permet d'implémenter le Discovery Service ?

- | | |
|---------------------------|--------------------|
| A. Spring Cloud Discovery | C. Netflix Hystrix |
| B. Eureka Discovery | D. Spring UDDI |

- 8. Quelles sont les trois parties qui composent un JWT (Json Web Token) ?**
- A. Header, Payload et Signature
 - B. Header, Contract, Signature
 - C. SOAP, WSDL et UDDI
 - D. Contrat, Payload et Signature
- 9. Comment est calculée la signature d'un JWT ?**
- A. La signature est calculée en fonction du Header, Payload et la clé publique
 - B. La signature est calculée en fonction du Header, Payload et la clé privée
 - C. La signature est calculée en fonction du Header, Payload et les deux clés : publique et privée
 - D. La signature est calculée en fonction Payload et la clé privée
- 10. Quand un utilisateur est authentifié via Keycloak, l'application frontend reçoit un Token qui contient :**
- A. Un JWT Access Token
 - B. Un JWT Refresh Token
 - C. Un JWT Access Token et un JWT Refresh Token
 - D. Un simple Token
- 11. Quelle est l'annotation de Spring MVC qui permet de déployer un web service RESTFUL sous forme d'un contrôleur ?**
- A. @RestService
 - B. @RestController
 - C. @RepositoryRestResource
 - D. @RestResource

Pour les questions QCM suivantes, on considère le code suivant (**ANNEXE_QCM**) d'un simple micro-service basé sur Spring Boot permettant de gérer des comptes en utilisant les dépendances Spring Data JPA, H2, Spring Web, Rest Repositories, Lombok et Dev Tools. On suppose que tous les éléments de l'application sont dans le même package, que le fichier de configuration application.properties est vide et que le port 8080 est libre

```

/*****
/***** ANNEXE_QCM *****/
/*****/

@Entity @Data @NoArgsConstructor @AllArgsConstructor @ToString
public class Compte{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private double solde; private Date dateCreation;
    private long contractID;    private TypeCompte type;
}

@RepositoryRestResource
public interface CompteRepository extends JpaRepository<Compte,Long> {
    List<Compte> findByType(TypeCompte typeCompte);
    List<Compte> findByContractID(long contratID);
}

@SpringBootApplication
public class ServiceCompteApplication extends SpringBootServletInitializer {
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(ServiceCompteApplication.class);
    }
    public static void main(String[] args) {
        SpringApplication.run(ServiceCompteApplication.class, args);
    }

    @Bean
    CommandLineRunner start(CompteRepository compteRepository){
        return args -> {
            compteRepository.save(new Compte(null,Math.random(),new Date(),1,TypeCompte.COURANT)) ;
            compteRepository.save(new Compte(null,Math.random(),new Date(),2,TypeCompte.COURANT)) ;
            compteRepository.save(new Compte(null,Math.random(),new Date(),1,TypeCompte.EPARGNE)) ;
        };
    }
}

```

12. Est-ce que le code de cette application (ANNEXE_QCM) est suffisant de générer au démarrage la table comptes dans la base de données H2

- 5

Les questions suivantes concernent le projet « INFRACTION » qui a été décrit au début de l'énoncé de l'épreuve.

B. Partie Conception :

1. Etablir une architecture technique du projet qui montre l'ensemble des micro-services de l'application
2. Etablir un diagramme de classes qui montre les entités métier de l'application.
3. Établir un diagramme de séquence qui montre les interactions entre les micro-services pour une opération de certification de chèque.

C. Partie Implémentation :

1. En utilisant Spring Data, Spring Data Rest, et Lombok, et Open Feign, écrire le code du micro-service **AGENCE-BANCARE-SERVICE** en implémentant les éléments suivants :
 - a. Code source des entités JPA Compte et Operation
 - b. Interfaces JPA Repository basées sur Spring Data
 - c. Un service qui permet de certifier un chèque sachant le numéro de compte, le numéro de chèque et le montant du chèque
 - d. Un Web service Restful basée sur RestController qui permet de certifier un chèque
2. En utilisant JPA, Spring Data, Spring Data Rest, et Lombok, écrire le code du micro-service **BNANQUE-CENTRALE-SERVICE** en implémentant les éléments suivants :
 - a. Code de l'entité JPA **AgenceBancaire**
 - b. Interface JPA Repository basée sur Spring Data
 - c. Interface OpenFeign permettant de consulter un véhicule sachant son id.
 - d. Web service Restful qui permet de certifier un chèque sachant le code de la banque, le numéro de chèque, le montant du chèque et le numéro de compte

ANNEXES

Ces annexes contiennent des exemples de codes sources qui n'ont aucun lien avec le problème traité dans cet examen. Toutefois, quelques annexes peuvent vous servir pour des consultations syntaxiques.

```
-----
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @ToString
class Product{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
}
-----
<div class="container">
    <button *ngIf="authService.isAdmin()" class="btn btn-danger"
(click)="onNewTask()">New Task</button>
    <table class="table table-striped">
        <thead>
            <tr>
                <th>ID</th><th>Task Name</th>
            </tr>
        </thead>
        <tbody>
            <tr *ngFor="let t of tasks">
                <td>{{t.id}}</td>
                <td>{{t.taskName}}</td>
            </tr>
        </tbody>
    </table>
</div>
-----
```

```
@RepositoryRestResource
interface ProductRepository extends JpaRepository<Product,Long>{
}
-----
```

```
@Document
@Data @AllArgsConstructor @NoArgsConstructor
public class Category {
    @Id
    private String id;
    private String name;
    @DBRef
    private Collection<Product> products=new ArrayList<>();
}
-----
```

```
import {JwtHelper} from 'angular2-jwt';
@Injectable()
export class AuthenticationService{
    private host:string="http://localhost:8080";
    private jwtToken:string;
    private roles:Array<any>=[];
    constructor(private http:HttpClient){}

    getTasks(){
        if(this.jwtToken==null) this.loadToken();
    }
}
```

```

        return this.http.get(this.host+"/tasks",{headers:new
HttpHeaders({'authorization':this.jwtToken)}});
    }
    logout(){
        localStorage.removeItem('token');
    }
    isAdmin(){
        for(let r of this.roles){
            if(r.authority=='ADMIN') return true;
        }
        return false;
    }
}

```

```

-----
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
class Bill{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date billingDate;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Long customerID;
    @Transient private Customer customer;
    @OneToMany(mappedBy = "bill")
    private Collection<ProductItem> productItems;
}

```

```

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

}

```

```

-----
@Projection(name="fullBill",types = Bill.class)
interface BillProjection{
    public Long getId();
    public Date getBillingDate();
    public Long getCustomerID();
    public Collection<ProductItem> getProductItems();
}

```

```

-----
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
class ProductItem{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Long productID;
    @Transient private Product product;
    private double price;
    private double quantity;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Bill bill;
}

```

```

-----
@Component
public class Configuration {
    @Autowired
    private EtudiantRepositoy etudiantRepositoy;
}

```



```

    @JmsListener(destination="scolarite.queue")

    public void receive(byte[] message) throws Exception {

        String strMessage=new String(message,"UTF-8");
        System.out.println(strMessage);
        ObjectMapper objectMapper=new ObjectMapper();
        Etudiant et=objectMapper.readValue(message, Etudiant.class);
        System.out.println("Nom:"+et.getNom());
        System.out.println("Prénom:"+et.getPrenom());
        etudiantRepositoy.save(et);
    }
}
-----
@RepositoryRestResource
interface ProductItemRepository extends JpaRepository<ProductItem,Long>{

}
-----
@Data
class Customer{
    private Long id; private String name; private String email;
}
-----
@FeignClient(name = "CUSTOMER-SERVICE")
interface CustomerService{
    @GetMapping("/customers/{id}")
    public Customer findCustomerById(@PathVariable(name="id") Long id);
}
-----
@Data
class Product{
    private Long id;
    private String name;
    private double price;
}
-----
@FeignClient(name="INVENTORY-SERVICE")
interface InventoryService{
    @GetMapping("/products/{id}")
    public Product findProductById(@PathVariable(name="id") Long id);
    @GetMapping("/products")
    public PagedModel<Product> findAllProducts();
}
-----
@RestController
public class Catalogue {
    @Autowired
    private ProduitRepository produitRepository;

    @RequestMapping(value="/produits",method=RequestMethod.GET)
    public List<Produit> listProduits(){
        return produitRepository.findAll();
    }
    @RequestMapping(value="/produits/{id}",method=RequestMethod.GET)
    public Produit getProduit(@PathVariable(name="id")Long id){
        return produitRepository.findOne(id);
    }
}

```

```

    }
    @RequestMapping(value="/produits",method=RequestMethod.POST)
    public Produit save(@RequestBody Produit p){
        return produitRepository.save(p);
    }

}

-----
@RestController
class BillRestController{
    @Autowired
    private BillRepository billRepository;
    @Autowired
    private ProductItemRepository productItemRepository;
    @Autowired
    private CustomerService customerService;
    @Autowired
    private InventoryService inventoryService;

    @GetMapping("/fullBill/{id}")

    public Bill getBill(@PathVariable(name="id") Long id){
        Bill bill=billRepository.findById(id).get();
        bill.setCustomer(customerService.findCustomerById(bill.getCustomerID()));
        bill.getProductItems().forEach(pi->{
            pi.setProduct(inventoryService.findProductById(pi.getProductID()));
        });
        return bill;
    }
}

-----
@SpringBootApplication
@EnableEurekaServer
public class EurekaServiceApplication {

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

}


-----
@RepositoryRestResource
public interface ProductRepository extends MongoRepository<Product,String> {
    public List<Product> findByNameContains(String motCle);
}

-----
@Bean
DiscoveryClientRouteDefinitionLocator dynamicRoutes(ReactiveDiscoveryClient rdc,
                                                    DiscoveryLocatorProperties dlp){
    return new DiscoveryClientRouteDefinitionLocator(rdc,dlp);
}

}

-----
@Component

```



```
@WebService
public class Sclarite {
    @Autowired
    private EtudiantRepository etudiantRepository;
    @Autowired
    private FormationRepository formationRepository;
    @WebMethod(operationName="listEtudiants")
    public List<Etudiant> list(){
        return etudiantRepository.findAll();
    }
    @WebMethod
    public Etudiant getOne(@WebParam(name="id") Long id) {
        return etudiantRepository.findOne(id);
    }
    @WebMethod
    public Formation save(@WebParam(name="formation")Formation f) {
        return formationRepository.save(f);
    }
}
```