Examen Architecture JEE

Durée : 1h30 | Documents non autorisés | Annexes Fournies avec l'épreuve | Session juillet 2023

Professeurs : Pr. F.Z. BADRI TIJANE, Pr. K. AHAIDOUS Pr. M. EL BAKKALI, Pr. M. YOUSSFI

On souhaite concevoir et développer une application Web JEE basée sur le Framework Spring. L'application permet de gérer les réservations des salles de conférences. Cette épreuve couvre une partie de l'application dont les règles de gestion sont les suivantes :

- Un employé peut effectuer plusieurs réservations. Chaque réservation concerne un seul employé
- Chaque réservation concerne une seule salle de conférence. Une salle de conférence peut concerner plusieurs réservations
- Chaque réservation concerne également plusieurs équipements. Chaque équipement peut concerner plusieurs réservations

Dans cette conception nous allons prendre en considération les attributs suivants :

- Un employé est défini par son id, son nom et son email
- Une salle est définie par son id, son numéro, son nom, le nombre de places et son type (REUNION, COURS, TP)
- Chaque équipement est défini par son id, son nom, son poids et son type (VIDEO_PROJECTEUR, CAMERA, TABLEAU_INTERACTIF, KIT_VISIO_CONFERENCE)
- Chaque réservation est définie par son id, sa date, durée, a description et son statut (EN_ATTENTE, VELIDEE, ANNULEE, REFUSEE)

Les données des micro-services sont stockées dans des bases de données de type H2 ou MySQL. L'application se compose de trois couches :

- La couche DAO est basée sur Spring Data, JPA, Hibernate et JDBC
- La couche service (métier) définie par une interface et une implémentation
- La couche Web basée sur Spring MVC avec un moteur de rendu HTML coté serveur basé sur Thymeleaf.

La sécurité de l'application est basée sur Spring Security.

Cette épreuve se compose de trois parties :

A. **Partie QCM** qui concerne des questions de concepts qui n'est pas liée à ce problème
B. **Partie Conception** qui est liée à ce problème
C. **Partie Implémentation** qui est liée à ce problème

Des annexes contenant des extraits de codes sont fournies. 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. **Dans est le cas qui correspond couplage faible ?**
   - **A.** Une classe A est liée à une autre classe B
   - **B.** Une classe A est liée à l'interface IB implémentée par la classe B
   - **C.** Une classe A n'est liée à aucune classe ni interface
   - **D.** Une classe A hérite d'une classe abstraite

2. **Quelle est la relation qui existe entre JPA et Hibernate ?**
   - **A.** JPA est une API et Hibernate et une implémentation de JPA
   - **B.** Hibernate est une API et JPA est un Framework qui implémente JPA
   - **C.** JPA est un Framework alors que Hibernate non
   - **D.** Aucune relation entre JPA et Hibernate

3. **Quelles sont les deux annotations JPA qui sont obligatoires pour créer une entité JPA ?**
   - **A.** @Entitiy
   - **B.** @Id
   - **C.** @DataJPA
   - **D.** @PersistenceContext

4. **Dans Spring, quelle est la façon la plus recommandée pour faire l'injection des dépendances ?**
   - **A.** En utilisant @Autowired
   - **B.** En utilisant un constructeur C.
   - **C.** En utilisant les Setters
   - **D.** En utilisant @Inject

5. **Quelle annotation à utiliser sur une méthode d'un contrôleur spring mvc pour que celle-ci soit exécuté quand une requête http est envoyée par la méthode GET avec le path « /reservations » ?**

   A. @GetMapping(path="/reservations")
   B. @GetHttpMapping(path="/reservations")
   C. @HttpGet(path="/reservations")
   D. @PostMapping(path="/reservations")

6. **On suppose que les classes A, B et C héritent de la classe abstraite D. Dans JPA, en utilisant la stratégie SINGLE_TABLE pour mapper l'héritage, Hibernate va générer les tables suivantes dans la base de données :**
   - **A.** Trois tables représentant les classes A, B et C
   - **B.** Quartes tables représentant les classes A, B, C et D.
   - **C.** Une seule table qui contient tous les attributs des 4 classes A, B, C et D
   - **D.** Une seule table qui contient tous les attributs des 3 classes A, B, C

7. **Dans un ORM (Object Relationnel Mapping), on veut réaliser le mapping d'une relation 1..* entre deux classes, par exemple Service et Salarié (un service contient plusieurs salariés). On peut réaliser ce mapping en utilisant structure relationnelle suivante :**
   - **A.** Sans table de jointure, en mettant la clef primaire de la table SALARIE dans la table SERVICE.
   - **B.** Sans table de jointure, en mettant la clef primaire de la table SERVICE dans la table SALARIE.
   - **C.** Avec une table de jointure, en mettant les clefs primaires des tables SERVICE et SALARIE dans la table de jointure.

Pour les questions QCM suivantes, on considère le code suivant (**ANNEXE_QCM**) d'une application basée sur Spring Boot permettant de gérer des payements en utilisant les dépendances Spring Data JPA, H2, Spring Web et Lombok. 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. Dans cette annexe chaque bloc de code est numéroté.

```
/*****************************************/
/************ ANNEXE_QCM *****************/
/*****************************************/
```

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
@Builder
public class Payment {
    @Id
    private String paymentId;
    private double amount;
    @Temporal(TemporalType.TIMESTAMP)
    private Date paymentDate;
}
```

1

```
public interface PaymentRepository
extends JpaRepository<Payment,String> {
}
```

2

```
public interface PaymentService {
    Payment findPaymentById(String id)
throws RuntimeException ;
    List<Payment> findAllPayments();
}
```

3

```
@Service @Transactional
@Slf4j
public class PaymentServiceImpl implements PaymentService {
    @Autowired private PaymentRepository paymentRepository;
    @Override
    public Payment findPaymentById(String id) throws RuntimeException {
      // To Do
    }
    @Override
    public List<Payment> findAllPayments() {
        // To do
    }
}
```

4

```
@RestController
public class PaymentRestController {
    @Autowired private PaymentService paymentService;
    @GetMapping("/payments")
    public List<Payment> payments(){
        return paymentService.findAllPayments();
    }
    @GetMapping("/payments/{id}")
    public Payment findPaymentById(String id){
        return paymentService.findPaymentById(id);
    }
}
```

5

Questions relatives à l'annexe de la page précédente (**ANNEXES_QCM : Page 3**) :

8. **Dans le code de (ANNEXE_QCM), quels sont les deux blocs de code qui représente la couche métier ?**
   A. 1 et 2
   B. 3 et 4
   C. 1 et 5
   D. 4 et 5
9. **Le code de la classe PaymentServiceImpl n'est pas complet, quel est le code qui est le plus juste pour l'implémentation de la méthode `findAllPayments()` ?**
   A. `paymentRepository.findAll();`
   B. `return paymentRepository.findAll();`
   C. `return paymentRepository.findAllPayments();`
   D. `return paymentService.findAll();`
10. **Le code de la classe PaymentServiceImpl n'est pas complet, quel est le code qui est le plus juste pour l'implémentation de la méthode `findPaymentById` () ?**
   A.
```
Payment payment=paymentRepository.findById(id)
    .orElseThrow(()->new RuntimeException("Payment not found"));
   return payment;
```
   B.
```
Payment payment=paymentRepository.findPaymentBy()
    .orElseThrow(()->new RuntimeException("Payment not found"));
   return payment;
```
   C.
```
Payment payment=paymentService.findById(id)
    .orElseThrow(()->new RuntimeException("Payment not found"));
   return payment;
```
   D. `return paymentService.findById (id);`
11. **Dans le code de l'ANNEXE_QCM, Quelle est l'annotation Lombock qui qui permet générer les getters et setters pour l'entité Payment ?**
   A. @Entity
   B. @Data
   C. @Id
   D. @GenratedValue
12. **Dans le code de l'ANNEXE1, Quelle est la classe ou l'interface qui permet d'intégrer Spring Data pour faire le mapping objet relationnel ?**
   A. Payment
   B. PaymentRepository
   C. PaymentService
   D. PaymentRestController
13. **Dans le code de l'ANNEXE_QCM, il manque une annotation à la méthode findPaymentById () de la classe PaymentRestController. Quelle est cette annotation manquante ?**
   A. @Parameter
   B. @QueryParam
   C. @PathParam
   D. @PathVariable

Les questions suivantes concernent le problème « Réservations des salles » 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 composants principaux de l'application. Cette architecture est un dessin qui montre les différents composants de l'application : Base de données, Backend, frontend, différentes couches de l'application (Dao, Métier, Web) et tous les composants que vous voyez importants dans l'architecture technique de cette application.
2. Etablir un diagramme de classes qui montre les entités métier de l'application. Dans ce digramme de classe, on ne représentera que les attributs des classes et les relations entre les classes.

C. Partie Implémentation :

1. **Couche DAO**
   a. Écrire le code des entités JPA Reservation, Salle et Equipement
   b. Écrire le code des interfaces JPA Repository basées sur Spring data : ReservationRepositoty, SalleRepository et EquipementRepository
2. **Couche service**
   a. Écrire le code de l'interface IReservationService répondant aux spécifications fonctionnelles suivantes :
      - Enregistrer une réservation
      - Consulter les réservations d'un employé donnée
   b. Écrire le code d'une implémentation de l'interface ReservationServiceImpl
3. **Couche Web**
   a. Écrire le code d'un contrôleur Spring MVC qui permet de :
      - Enregistrer une réservation
      - Consulter les réservations d'un employé dont son id est sera saisi
   b. Écrire la partie d'un code d'une vue basée sur Thymeleaf qui permet d'afficher la liste des réservations
4. **Sécurité :** On souhaite sécuriser l'application avec un système d'authentification basé sur Spring Security en utilisant deux rôles « ROLE_EMPLOYE » et « ROLE_ADMIN », Écrire l'essentiel du code de la configuration de base de Spring Security permettant de sécuriser l'application en utilisant la stratégie InMemoryAuthentication.

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 en cas de besoin.

```
public interface BankAccountRepository extends
JpaRepository<BankAccount,String> {
}

public interface OperationRepository extends
JpaRepository<AccountOperation,Long> {
  List<AccountOperation> findByBankAccountId(String accountId);
  Page<AccountOperation> findByBankAccountIdOrderByOperationDateDesc(String
accountId, Pageable pageable);
}
```

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @OneToMany(mappedBy = "customer")
    private List<BankAccount> bankAccounts;
}
```

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "TYPE",length = 4)
@Data @NoArgsConstructor @AllArgsConstructor
public class BankAccount {
    @Id
    private String id;
    private double balance;
    private Date createdAt;
    @Enumerated(EnumType.STRING)
    private AccountStatus status;
    @ManyToOne
    private Customer customer;
    @OneToMany(mappedBy = "bankAccount",fetch = FetchType.LAZY)
    private List<AccountOperation> accountOperations;
}
```

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

```
@Service
public class BankAccountMapperImpl {
    public CustomerDTO fromCustomer(Customer customer){
        CustomerDTO customerDTO=new CustomerDTO();
        BeanUtils.copyProperties(customer,customerDTO);
        return  customerDTO;
    }
}
```

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AccountOperation {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date operationDate;
    private double amount;
    @Enumerated(EnumType.STRING)
    private OperationType type;
    @ManyToOne
    private BankAccount bankAccount;
    private String description;
}
```

```
@Entity
@DiscriminatorValue("CA")
@Data @NoArgsConstructor @AllArgsConstructor
public class CurrentAccount extends BankAccount {
    private double overDraft;
}
```

```
public enum OperationType {
    DEBIT, CREDIT
}
```

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class AppUser {
    @Id
    private String userId;
    private String username;
    private String password;
    private String email;
    @ManyToMany(fetch = FetchType.EAGER)
    private List<AppRole> appRoles=new ArrayList<>();
}

@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class AppRole {
    @Id
    private String role;
}
```

```
@Controller
public class ProduitController {
	@Autowired
	private ProduitRepository produitRepository;

	@GetMapping(path="/index")
	public String index(Model model,
			@RequestParam(name="page",defaultValue="0")int p,
			@RequestParam(name="size",defaultValue="5")int s,
			@RequestParam(name="motCle",defaultValue="")String mc){
		Page<Produit> pageProduits=
	produitRepository.chercher("%"+mc+"%",new PageRequest(p, s));

		model.addAttribute("listProduits",pageProduits.getContent());
		int[] pages=new int[pageProduits.getTotalPages()];
		model.addAttribute("pages",pages);model.addAttribute("size",s);
		model.addAttribute("pageCourante",p);
		model.addAttribute("motCle",mc);
		return "produits";
	}

	@RequestMapping(value="/admin/save",method=RequestMethod.POST)
	public String save(Model model,@Valid Produit p,BindingResult
	bindingResult){
	 if(bindingResult.hasErrors()){ return "FormProduit"; }
	 produitRepository.save(p);
	 return "Confirmation";
	}
```

```
@RestController
public class ProduitRestService {
	@Autowired
  private ProduitRepository  produitRepository;

	@GetMapping(path="/produits")
	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);
	}

}
```

```
@Service
@Transactional
public class BanqueMetierImpl  implements IBanqueMetier {
  @Autowired
  private CompteRepsoitory compteRepository ;
```

```
        @Override
        public Compte save(Compte cp){
          return compteRepsoitory.save(cp) ;
        }
      }
```

```
var app=angular.module("MyApp",['ui.router']);
app.controller("ProduitController",function($scope,$http){
  $scope.pageProduits=null;  $scope.motCle="";
  $scope.pageCourante=0;  $scope.size=5; $scope.pages=[];
  $scope.chercher=function(){
        $http.get("chercher?mc="+$scope.motCle
                    +"&page="+$scope.pageCourante+"&size="+$scope.size)
          .success(function(data){
            $scope.pageProduits=data;
            $scope.pages=new Array(data.totalPages);
          })
          .error(function(err){
            console.log(err);
          })
  }
});
```

```
      <!DOCTYPE html>
      <html xmlns:th="http://www.thymeleaf.org">
      <head>
      <meta charset="utf-8"/>
      <title>Insert title here</title>
       <link rel="stylesheet" type="text/css"
      href="../static/css/bootstrap.min.css"
       th:href="@{css/bootstrap.min.css}"/>
       <link rel="stylesheet" type="text/css" href="../static/css/myStyle.css"
       th:href="@{css/myStyle.css}"/>
      </head>
      <body>
       <div class="container spacer">
         <form action="chercher?page=0" method="GET">
           <label>Mot Clé:</label>
           <input type="text" name="mc" th:value="${mc}"/>
           <input type="submit" value="Chercher"/>
         </form>
       </div>
      <div class="container">

        <table class="table table-striped">
          <thead>
            <tr>
              <th>ID</th><th>DES</th><th>PRIX</th>
            </tr>
          </thead>
          <tbody>
            <tr th:each="p:${pageProduit.content}">
              <td th:text="${p.idProduit}"></td>
              <td th:text="${p.designation}"></td>
              <td th:text="${p.prix}"></td>
```

```
                <td><a th:href="@{/detail(id=${p.id})}"> Detail</a></td>
              </tr>
             </tbody>
           </table>
          </div>
         </body>
         </html>
```

```
<div class="container mt-2">
 <ng-container *ngIf="customers | async as listCustomers; else failureOrLading">
  <div class="card">
   <div class="card-header">Customers</div>
   <div class="card-body">
    <div *ngIf="searchFormGroup">
     <form [formGroup]="searchFormGroup" (ngSubmit)="handleSearchCustomers()">
      <div class="input-group">
       <label class="input-group-text">Keyword :</label>
       <input type="text" formControlName="keyword" class="form-control">
       <button class="btn btn-info">
        <i class="bi bi-search"></i>
       </button>
      </div>
     </form>
    </div>
    <table class="table">
     <thead>
     <tr>
      <th>ID</th><th>Name</th><th>Email</th>
     </tr>
     </thead>
     <tbody>
     <tr *ngFor="let c of customers | async">
      <td>{{c.id}}</td>
      <td>{{c.name}}</td>
      <td>{{c.email}}</td>
      <td>
       <button (click)="handleDeleteCustomer(c)" class="btn btn-danger">
        <i class="bi bi-trash"></i>
       </button>
      </td>
      <td>
       <button (click)="handleCustomerAccounts(c)" class="btn btn-success">
        Accounts
       </button>
      </td>
     </tr>
     </tbody>
    </table>
   </div>
  </div>
 </ng-container>
 <ng-template #failureOrLading>
  <ng-container *ngIf="errorMessage; else loading">
   <div class="text-danger">
    {{errorMessage}}
   </div>
```

```
  </ng-container>
  <ng-template #loading>
   Loading .....
  </ng-template>
 </ng-template>
</div>
```

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
public class SecurityConfig {

  @Autowired
  private PasswordEncoder passwordEncoder;

  @Bean
  public JdbcUserDetailsManager jdbcUserDetailsManager(DataSource dataSource){
    JdbcUserDetailsManager jdbcUserDetailsManager=new JdbcUserDetailsManager(dataSource);
    return jdbcUserDetailsManager;
  }
  //@Bean
  public InMemoryUserDetailsManager inMemoryUserDetailsManager(){
    String pwd=passwordEncoder.encode("12345");
    System.out.println(pwd);
    InMemoryUserDetailsManager inMemoryUserDetailsManager=new InMemoryUserDetailsManager(
        User.withUsername("user1").password(pwd).roles("USER").build(),
        User.withUsername("user2").password(passwordEncoder.encode("3333")).roles("USER").build(),

User.withUsername("admin").password(passwordEncoder.encode("4444")).roles("USER","ADMIN").build()
    );
    return inMemoryUserDetailsManager;
  }
  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {
    httpSecurity.formLogin();
    httpSecurity.authorizeHttpRequests().requestMatchers("/admin/**").hasRole("ADMIN");
    httpSecurity.authorizeHttpRequests().requestMatchers("/user/**").hasRole("USER");
    httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
    httpSecurity.authorizeHttpRequests(ar->ar.anyRequest().authenticated());

    httpSecurity.exceptionHandling().accessDeniedPage("/notAuthorized");
   return httpSecurity.build();
  }
}
```

```
@Configuration @EnableWebSecurity
public class Config extends WebConfigurerAdapter {
      @Autowired
      private DataSource dataSource;
   @Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {

         auth.inMemoryAuthentication()
        .withUser("admin").password("123").roles("USER","ADMIN");
```

```
       auth.inMemoryAuthentication()
         .withUser("user").password("123").roles("USER");

          auth.jdbcAuthentication()
            .dataSource(dataSource)
            .usersByUsernameQuery("select username as principal,password as
credentials,active from users where username=?")
            .authoritiesByUsernameQuery("select username as principal, role as role
from users_roles where username=?")
            .rolePrefix("ROLE_")
            .passwordEncoder(new Md5PasswordEncoder());
}

   @Override
       protected void configure(HttpSecurity http) throws Exception {
              auth.inMemoryAuthentication()
              .withUser("admin").password("123").roles("USER","ADMIN");
              auth.inMemoryAuthentication()
              .withUser("user").password("123").roles("USER");

              http.formLogin().loginPage("/login");
              http.csrf().disable();
              http.authorizeRequests().antMatchers("/admin/*").hasRole("ADMIN");
              http.authorizeRequests().antMatchers("/user/*").hasRole("USER");
              http.exceptionHandling().accessDeniedPage("/403");
       }
}
```

```
package web;
import java.io.IOException; import javax.servlet.*; import javax.servlet.http.*;
public class ControleurServlet extends HttpServlet{
  @Override
  public void init() throws ServletException {
    // Initialisation
      }
  @Override
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Traitement effectué si une requête Http est envoyée avec GET
  }
  @Override
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // Traitement effectué si une requête Http est envoyée avec POST
  }
  @Override
  public void destroy() {
  }
}
```