

Système de Fichiers Distribué HDFS & SGBD NoSQL avancé

Compte Rendu de Exam: **Examen Architecture Distribuée et Middleware**

II BDCC – Ingénierie Informatique, Big Data et Cloud Computing

**Encadrer par :  
 -** Pr. Abdelmajid BOUSSELHAM

**Compte Rendu réalisé par :  
 -** Taoudi Abdelbasset

**Filière : II-BDCC 2025-2026**

**Cycle d’ingénieurs**

1. Sommaire

**Contents**

[I. Sommaire 2](#__RefHeading___Toc411_2104189841)

[II. S\_\_ 3](#__RefHeading___Toc413_2104189841)

1. Conception :

# Établir une architecture technique du projet:

## 1. Couche Frontend (Angular)

* **Technologie** : Framework Angular
* **Rôle** : Fournit l'interface utilisateur et les fonctionnalités côté client
* **Fonctionnalités** :
  + Envoie des requêtes HTTP à l'API REST backend
  + Affiche les données reçues du backend
  + Gère la validation côté client et les interactions utilisateur
  + Gère l'état de l'application côté client
* **Communication** : Utilise le protocole HTTP avec format de données JSON pour communiquer avec le backend

## 2. Couche API REST (Contrôleurs)

* **Technologie** : Contrôleurs Spring MVC
* **Rôle** : Point d'entrée pour toutes les requêtes HTTP vers le backend
* **Fonctionnalités** :
  + Définit les endpoints pour diverses opérations (GET, POST, PUT, DELETE)
  + Valide les requêtes entrantes
  + Délègue le traitement métier à la couche Service
  + Renvoie des réponses HTTP appropriées (codes d'état et corps de réponse)
* **Classes principales** : Classes de contrôleur annotées avec @RestController et @RequestMapping

## 3. DTOs (Objets de Transfert de Données)

* **Rôle** : Définit la structure des données échangées entre frontend et backend
* **Fonctionnalités** :
  + Encapsule les données de requête et de réponse
  + Fournit un contrat clair pour la communication API
  + Découple l'interface API des structures de données internes
* **Classes principales** : Classes POJO dans le package dtos avec champs, getters, setters et annotations de validation

## 4. Couche Sécurité

* **Technologie** : Spring Security avec JWT (JSON Web Tokens)
* **Rôle** : Protège les ressources backend et gère l'authentification/autorisation
* **Fonctionnalités** :
  + Authentifie les utilisateurs basé sur leurs identifiants
  + Émet des tokens JWT lors d'une authentification réussie
  + Valide les tokens pour l'accès aux endpoints protégés
  + Applique le contrôle d'accès basé sur les rôles
* **Composants principaux** :
  + Filtres de sécurité
  + Fournisseur de tokens JWT
  + Gestionnaire d'authentification
  + Service de détails utilisateur

## 5. Couche Service

* **Rôle** : Contient la logique métier et orchestre les opérations
* **Fonctionnalités** :
  + Implémente les règles métier fondamentales de l'application
  + Coordonne plusieurs repositories lorsque nécessaire
  + Gère les transactions
  + Effectue la validation et la transformation des données
* **Classes principales** : Classes de service annotées avec @Service

## 6. Mappers

* **Rôle** : Convertit entre les objets DTO et les objets Entity
* **Fonctionnalités** :
  + Transforme les données entre la représentation API (DTOs) et le modèle de domaine (Entities)
  + Gère les mappages complexes entre différentes structures d'objets
  + Isole la logique de conversion dans des classes dédiées
* **Options d'implémentation** :
  + Méthodes de mapping manuel
  + Bibliothèque MapStruct
  + Bibliothèque ModelMapper

## 7. Entities (Entités)

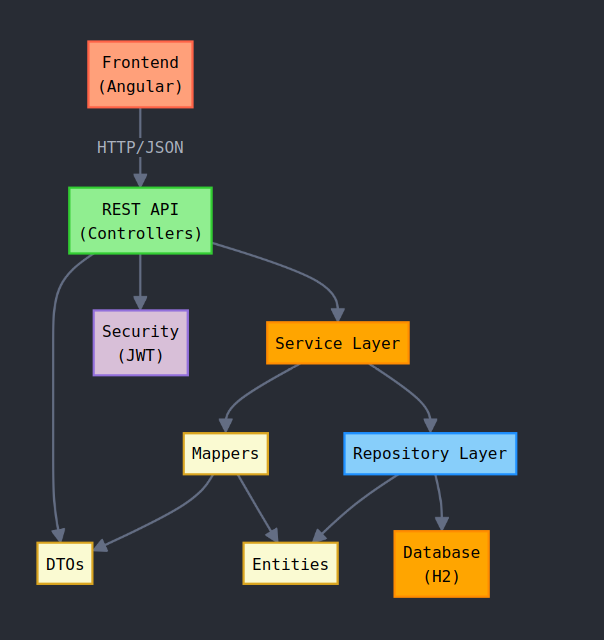
* **Rôle** : Représente le modèle de domaine et la structure de la base de données
* **Fonctionnalités** :
  + Correspond aux tables de la base de données via les annotations JPA
  + Définit les relations entre les objets du domaine
  + Encapsule la logique de domaine liée à l'entité
* **Caractéristiques principales** : Annotations JPA comme @Entity, @Table, @Column, etc.

## 8. Couche Repository

* **Technologie** : Spring Data JPA
* **Rôle** : Gère les opérations d'accès aux données
* **Fonctionnalités** :
  + Fournit des opérations CRUD pour les entités
  + Traduit entre les objets Java et les enregistrements de base de données
  + Gère les requêtes et les transactions de base de données
* **Classes principales** : Interfaces Repository étendant JpaRepository ou d'autres interfaces Spring Data

## 9. Base de données (H2)

* **Technologie** : Base de données en mémoire H2
* **Rôle** : Persiste les données de l'application
* **Fonctionnalités** :
  + Stocke et récupère les données
  + Applique les contraintes d'intégrité des données
  + Fournit des capacités transactionnelles
* **Configuration** : Définie dans les propriétés de l'application avec les détails de connexion, options de génération de schéma



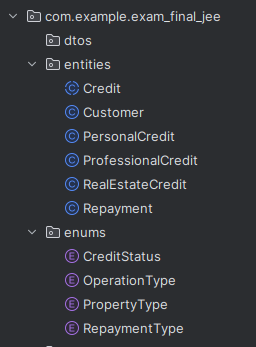
# Établir un diagramme de classes qui montre les entités. On ne représentera que les attributs:

# 

1. Implémentation:

# Couche DAO:

### Créer les entités JPA



- Credit

package com.example.exam\_final\_jee.entities;  
  
  
import com.example.exam\_final\_jee.enums.CreditStatus;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.util.Date;  
import java.util.List;  
  
  
@Entity  
@Inheritance(strategy = InheritanceType.*SINGLE\_TABLE*)  
@DiscriminatorColumn(name = "CREDIT\_TYPE")  
public abstract class Credit {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private Date applicationDate;  
 private Double amount;  
 private Integer repaymentDuration; // in months  
 private Double interestRate;  
  
 @Enumerated(EnumType.*STRING*)  
 private CreditStatus status; // IN\_PROGRESS, ACCEPTED, REJECTED  
  
 private Date acceptanceDate;  
  
 @ManyToOne  
 private Customer customer;  
  
 @OneToMany(mappedBy = "credit", cascade = CascadeType.*ALL*)  
 private List<Repayment> repayments;  
}

- Customer

package com.example.exam\_final\_jee.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.util.List;  
  
@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<Credit> credits;  
}

- Repayment

package com.example.exam\_final\_jee.entities;  
  
import com.example.exam\_final\_jee.enums.RepaymentType;  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import java.util.Date;  
  
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Repayment {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private Date date; // Date when repayment was made  
 private Double amount; // Amount repaid  
  
 @Enumerated(EnumType.*STRING*)  
 private RepaymentType type; // INSTALLMENT or EARLY\_REPAYMENT  
  
 @ManyToOne  
 @JoinColumn(name = "credit\_id")  
 private Credit credit; // Link to the associated credit  
}

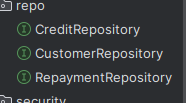
Type Credit :

package com.example.exam\_final\_jee.entities;  
  
import com.example.exam\_final\_jee.enums.PropertyType;  
import jakarta.persistence.DiscriminatorValue;  
import jakarta.persistence.Entity;  
import jakarta.persistence.EnumType;  
import jakarta.persistence.Enumerated;  
  
@Entity  
@DiscriminatorValue("REAL\_ESTATE")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class RealEstateCredit extends Credit {  
 @Enumerated(EnumType.*STRING*)  
 private PropertyType propertyType; // APARTMENT, HOUSE, COMMERCIAL  
}

package com.example.exam\_final\_jee.entities;  
  
import jakarta.persistence.DiscriminatorValue;  
import jakarta.persistence.Entity;  
  
@Entity  
@DiscriminatorValue("PROFESSIONAL")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class ProfessionalCredit extends Credit {  
 private String purpose;  
 private String companyBusinessName;  
}

package com.example.exam\_final\_jee.entities;  
  
import jakarta.persistence.\*;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Entity  
@DiscriminatorValue("PERSONAL")  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class PersonalCredit extends Credit {  
 private String purpose; // car purchase, studies, renovations, etc.  
}

### Créer les interfaces JPA Repository basées sur Spring Data



- CreaditRepo

package com.example.exam\_final\_jee.repo;  
  
import com.example.exam\_final\_jee.entities.Credit;  
import com.example.exam\_final\_jee.enums.CreditStatus;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import java.util.Date;  
import java.util.List;  
  
public interface CreditRepository extends JpaRepository<Credit, Long> {  
 // Find credits by customer ID  
 List<Credit> findByCustomerId(Long customerId);  
  
 // Find credits by status  
 List<Credit> findByStatus(CreditStatus status);  
  
 // Search credits within a date range  
 @Query("SELECT c FROM Credit c WHERE c.applicationDate BETWEEN :start AND :end")  
 List<Credit> findCreditsBetweenDates(@Param("start") Date startDate, @Param("end") Date endDate);  
  
 // Find credits with amount greater than  
 List<Credit> findByAmountGreaterThan(Double amount);  
}

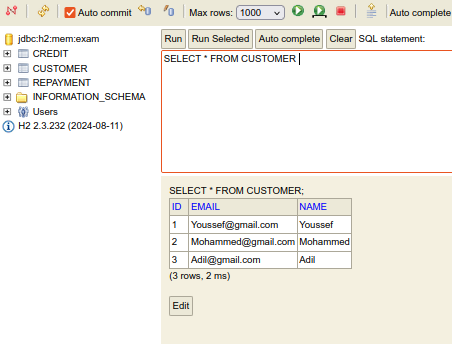
- CustomerRepo

package com.example.exam\_final\_jee.repo;  
  
import com.example.exam\_final\_jee.entities.Customer;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
  
import java.util.List;  
  
public interface CustomerRepository extends JpaRepository<Customer, Long> {  
 @Query("select c from Customer c where c.name like :kw")  
 List<Customer> searchCustomer(@Param("kw") String keyword);  
}

- RepaymentRepo

package com.example.exam\_final\_jee.repo;  
  
import com.example.exam\_final\_jee.entities.Repayment;  
import com.example.exam\_final\_jee.enums.RepaymentType;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.data.jpa.repository.Query;  
import org.springframework.data.repository.query.Param;  
import java.util.Date;  
import java.util.List;  
  
public interface RepaymentRepository extends JpaRepository<Repayment, Long> {  
 List<Repayment> findByCreditId(Long creditId);  
  
 // Find repayments by type (INSTALLMENT/EARLY\_REPAYMENT)  
 List<Repayment> findByType(RepaymentType type);  
  
 // Find repayments within a date range  
 @Query("SELECT r FROM Repayment r WHERE r.date BETWEEN :start AND :end")  
 List<Repayment> findRepaymentsBetweenDates(@Param("start") Date startDate, @Param("end") Date endDate);  
  
 // Calculate total repaid amount for a credit  
 @Query("SELECT COALESCE(SUM(r.amount), 0) FROM Repayment r WHERE r.credit.id = :creditId")  
 Double getTotalRepaidAmountByCreditId(@Param("creditId") Long creditId);  
}

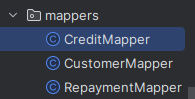
### Tester la couche DAO avec une application qui alimente la base de données avec quelques enregistrements de test.



package com.example.exam\_final\_jee;  
  
import com.example.exam\_final\_jee.entities.Customer;  
import com.example.exam\_final\_jee.repo.CreditRepository;  
import com.example.exam\_final\_jee.repo.CustomerRepository;  
import com.example.exam\_final\_jee.repo.RepaymentRepository;  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
  
import java.util.stream.Stream;  
  
@SpringBootApplication  
public class ExamFinalJeeApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(ExamFinalJeeApplication.class, args);  
 }  
  
 @Bean  
 CommandLineRunner start(CustomerRepository customerRepository, CreditRepository creditRepository, RepaymentRepository repaymentRepository){  
 return args -> {  
 Stream.*of*("Youssef","Mohammed","Adil").forEach(name -> {  
 Customer customer = new Customer();  
 customer.setName(name);  
 customer.setEmail(name+"@gmail.com");  
 customerRepository.save(customer);  
 });  
 };  
 }  
}

# Créer une couche service:

### Creation DTO est Mapper



- CreditFTO

package com.example.exam\_final\_jee.dtos;  
  
import com.example.exam\_final\_jee.enums.CreditStatus;  
import lombok.Data;  
import java.util.Date;  
import java.util.List;  
  
@Data  
public class CreditDTO {  
 private Long id;  
 private Date applicationDate;  
 private Double amount;  
 private Integer repaymentDuration;  
 private Double interestRate;  
 private CreditStatus status;  
 private Date acceptanceDate;  
 private Long customerId;  
 private List<RepaymentDTO> repayments;  
}

package com.example.exam\_final\_jee.dtos;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
// PersonalCreditDTO.java  
@Data  
public class PersonalCreditDTO extends CreditDTO {  
 private String purpose;  
}

package com.example.exam\_final\_jee.dtos;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
public class ProfessionalCreditDTO extends CreditDTO {  
 private String purpose;  
 private String companyBusinessName;  
}

package com.example.exam\_final\_jee.dtos;  
  
import com.example.exam\_final\_jee.enums.PropertyType;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
// RealEstateCreditDTO.java  
@Data  
public class RealEstateCreditDTO extends CreditDTO {  
 private PropertyType propertyType;  
}

- CustomerDTO

package com.example.exam\_final\_jee.dtos;  
  
import lombok.Data;  
import java.util.List;  
  
@Data  
public class CustomerDTO {  
 private Long id;  
 private String name;  
 private String email;  
 private List<CreditDTO> credits;  
}

- RepayemntDTO

package com.example.exam\_final\_jee.dtos;  
  
import com.example.exam\_final\_jee.enums.RepaymentType;  
import lombok.Data;  
import java.util.Date;  
  
@Data  
public class RepaymentDTO {  
 private Long id;  
 private Date date;  
 private Double amount;  
 private RepaymentType type;  
 private Long creditId;  
}

# Créer les Web services:

# Application frontend en utilisant Angular:

# Sécuriser:

# Apporter des améliorations additionnelles à votre projet: