

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)

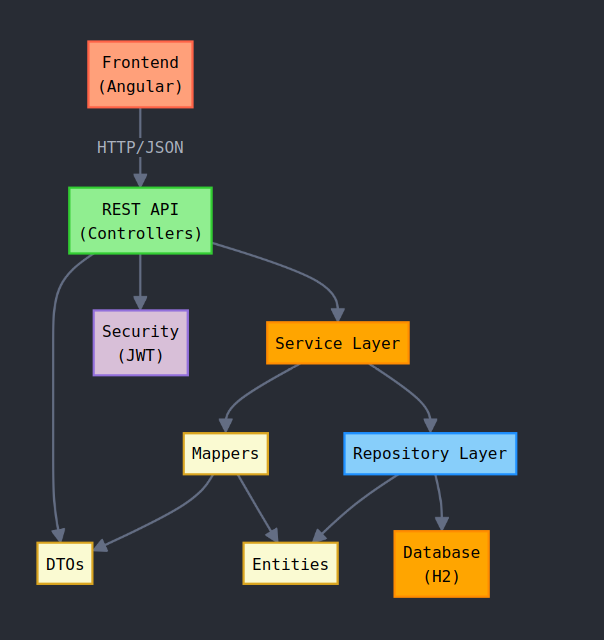
* **Rôle** : Représente le modèle de domaine et la structure de la base de données
* **Fonctionnalités** :
  + Correspond aux tables de la base de données via les annotations JPA
  + Définit les relations entre les objets du domaine
  + Encapsule la logique de domaine liée à l'entité
* **Caractéristiques 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



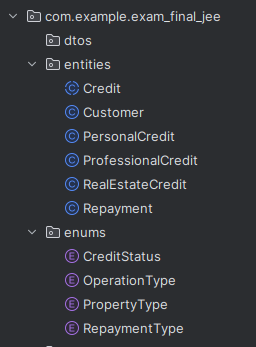
# Établir un diagramme de classes qui montre les entités. On ne représentera que les attributs:

# 

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  
}

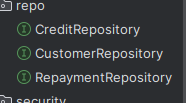
Type Credit :

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);  
}

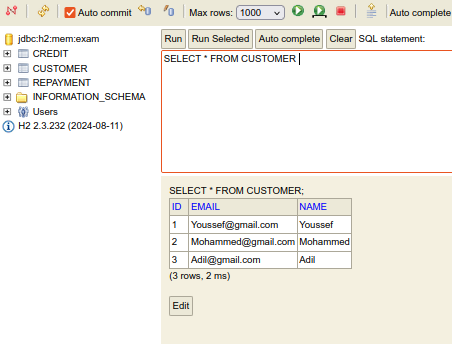
- CustomerRepo

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);  
}

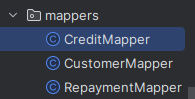
### Tester la couche DAO avec une application qui alimente la base de données avec quelques enregistrements de test.



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;  
}

package com.example.exam\_final\_jee.dtos;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class CreditSimulationResponseDTO {  
 private Double monthlyPayment;  
 private Double totalInterest;  
 private Double totalPayment;  
}

package com.example.exam\_final\_jee.dtos;  
  
import lombok.Data;  
  
@Data  
public class CreditSimulationRequestDTO {  
 private Double amount;  
 private Double interestRate; // annual rate  
 private Integer durationMonths;  
}

-----

- 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;  
}

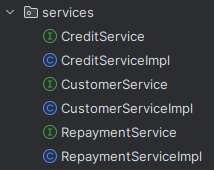
- RepayemntDTO

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;  
}

- BankMapper.java

package com.example.exam\_final\_jee.mappers;  
  
import com.example.exam\_final\_jee.dtos.\*;  
import com.example.exam\_final\_jee.entities.\*;  
import org.springframework.beans.BeanUtils;  
import org.springframework.stereotype.Service;  
  
@Service  
public class BankMapper {  
 public CustomerDTO fromCustomer(Customer customer) {  
 CustomerDTO customerDTO = new CustomerDTO();  
 BeanUtils.*copyProperties*(customer, customerDTO);  
 return customerDTO;  
 }  
  
 public Customer fromCustomerDTO(CustomerDTO customerDTO) {  
 Customer customer = new Customer();  
 BeanUtils.*copyProperties*(customerDTO, customer);  
 return customer;  
 }  
  
 public CreditDTO fromCredit(Credit credit) {  
 CreditDTO creditDTO = new CreditDTO();  
 BeanUtils.*copyProperties*(credit, creditDTO);  
 creditDTO.setCustomerId(credit.getCustomer().getId());  
 return creditDTO;  
 }  
  
 public PersonalCreditDTO fromPersonalCredit(PersonalCredit credit) {  
 PersonalCreditDTO creditDTO = new PersonalCreditDTO();  
 BeanUtils.*copyProperties*(credit, creditDTO);  
 creditDTO.setCustomerId(credit.getCustomer().getId());  
 return creditDTO;  
 }  
  
 public ProfessionalCreditDTO fromProfessionalCredit(ProfessionalCredit credit) {  
 ProfessionalCreditDTO creditDTO = new ProfessionalCreditDTO();  
 BeanUtils.*copyProperties*(credit, creditDTO);  
 creditDTO.setCustomerId(credit.getCustomer().getId());  
 return creditDTO;  
 }  
  
 public RealEstateCreditDTO fromRealEstateCredit(RealEstateCredit credit) {  
 RealEstateCreditDTO creditDTO = new RealEstateCreditDTO();  
 BeanUtils.*copyProperties*(credit, creditDTO);  
 creditDTO.setCustomerId(credit.getCustomer().getId());  
 return creditDTO;  
 }  
  
 public PersonalCredit fromPersonalCreditDTO(PersonalCreditDTO creditDTO) {  
 PersonalCredit credit = new PersonalCredit();  
 BeanUtils.*copyProperties*(creditDTO, credit);  
 return credit;  
 }  
  
 public ProfessionalCredit fromProfessionalCreditDTO(ProfessionalCreditDTO creditDTO) {  
 ProfessionalCredit credit = new ProfessionalCredit();  
 BeanUtils.*copyProperties*(creditDTO, credit);  
 return credit;  
 }  
  
 public RealEstateCredit fromRealEstateCreditDTO(RealEstateCreditDTO creditDTO) {  
 RealEstateCredit credit = new RealEstateCredit();  
 BeanUtils.*copyProperties*(creditDTO, credit);  
 return credit;  
 }  
  
 public RepaymentDTO fromRepayment(Repayment repayment) {  
 RepaymentDTO repaymentDTO = new RepaymentDTO();  
 BeanUtils.*copyProperties*(repayment, repaymentDTO);  
 repaymentDTO.setCreditId(repayment.getCredit().getId());  
 return repaymentDTO;  
 }  
  
 public Repayment fromRepaymentDTO(RepaymentDTO repaymentDTO) {  
 Repayment repayment = new Repayment();  
 BeanUtils.*copyProperties*(repaymentDTO, repayment);  
 return repayment;  
 }  
}

### Couch Service



**- Interfaces :**

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.\*;  
import com.example.exam\_final\_jee.enums.CreditStatus;  
import com.example.exam\_final\_jee.exceptions.CreditNotFoundException;  
import com.example.exam\_final\_jee.exceptions.CustomerNotFoundException;  
  
import java.util.Date;  
import java.util.List;  
  
public interface CreditService {  
 // Common credit operations  
 CreditDTO getCredit(Long creditId) throws CreditNotFoundException;  
 List<CreditDTO> listCredits();  
 void deleteCredit(Long creditId);  
 List<CreditDTO> getCreditsByCustomer(Long customerId) throws CustomerNotFoundException;  
 List<CreditDTO> getCreditsByStatus(CreditStatus status);  
 List<CreditDTO> getCreditsBetweenDates(Date startDate, Date endDate);  
 List<CreditDTO> getCreditsAboveAmount(Double amount);  
 CreditDTO updateCreditStatus(Long creditId, CreditStatus status) throws CreditNotFoundException;  
  
 // Specific credit type operations  
 PersonalCreditDTO savePersonalCredit(PersonalCreditDTO creditDTO) throws CustomerNotFoundException;  
 ProfessionalCreditDTO saveProfessionalCredit(ProfessionalCreditDTO creditDTO) throws CustomerNotFoundException;  
 RealEstateCreditDTO saveRealEstateCredit(RealEstateCreditDTO creditDTO) throws CustomerNotFoundException;  
  
 // Credit simulation  
 CreditSimulationResponseDTO simulateCredit(CreditSimulationRequestDTO simulationRequest);  
}

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.CustomerDTO;  
import com.example.exam\_final\_jee.exceptions.CustomerNotFoundException;  
  
import java.util.List;  
  
public interface CustomerService {  
 CustomerDTO saveCustomer(CustomerDTO customerDTO);  
 List<CustomerDTO> listCustomers();  
 CustomerDTO getCustomer(Long customerId) throws CustomerNotFoundException;  
 CustomerDTO updateCustomer(CustomerDTO customerDTO);  
 void deleteCustomer(Long customerId);  
 List<CustomerDTO> searchCustomers(String keyword);  
}

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.RepaymentDTO;  
import com.example.exam\_final\_jee.enums.RepaymentType;  
import com.example.exam\_final\_jee.exceptions.CreditNotFoundException;  
import com.example.exam\_final\_jee.exceptions.RepaymentNotFoundException;  
  
import java.util.Date;  
import java.util.List;  
  
public interface RepaymentService {  
 RepaymentDTO saveRepayment(RepaymentDTO repaymentDTO) throws CreditNotFoundException;  
 List<RepaymentDTO> getRepaymentsByCredit(Long creditId);  
 List<RepaymentDTO> getRepaymentsByType(RepaymentType type);  
 List<RepaymentDTO> getRepaymentsBetweenDates(Date startDate, Date endDate);  
 Double getTotalRepaidAmount(Long creditId);  
 RepaymentDTO getRepayment(Long repaymentId) throws RepaymentNotFoundException;  
 void deleteRepayment(Long repaymentId);  
}

**- Implementation :**

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.\*;  
import com.example.exam\_final\_jee.entities.\*;  
import com.example.exam\_final\_jee.enums.CreditStatus;  
import com.example.exam\_final\_jee.exceptions.CreditNotFoundException;  
import com.example.exam\_final\_jee.exceptions.CustomerNotFoundException;  
import com.example.exam\_final\_jee.mappers.BankMapper;  
import com.example.exam\_final\_jee.repo.CreditRepository;  
import com.example.exam\_final\_jee.repo.CustomerRepository;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Date;  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@Transactional  
@AllArgsConstructor  
public class CreditServiceImpl implements CreditService {  
 private CreditRepository creditRepository;  
 private CustomerRepository customerRepository;  
 private BankMapper bankMapper;  
  
 @Override  
 public CreditDTO getCredit(Long creditId) throws CreditNotFoundException {  
 Credit credit = creditRepository.findById(creditId)  
 .orElseThrow(() -> new CreditNotFoundException("Credit Not Found"));  
 return mapToSpecificCreditDTO(credit);  
 }  
  
 @Override  
 public List<CreditDTO> listCredits() {  
 List<Credit> credits = creditRepository.findAll();  
 return credits.stream()  
 .map(this::mapToSpecificCreditDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public void deleteCredit(Long creditId) {  
 creditRepository.deleteById(creditId);  
 }  
  
 @Override  
 public List<CreditDTO> getCreditsByCustomer(Long customerId) throws CustomerNotFoundException {  
 Customer customer = customerRepository.findById(customerId)  
 .orElseThrow(() -> new CustomerNotFoundException("Customer Not Found"));  
 List<Credit> credits = creditRepository.findByCustomerId(customerId);  
 return credits.stream()  
 .map(this::mapToSpecificCreditDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<CreditDTO> getCreditsByStatus(CreditStatus status) {  
 List<Credit> credits = creditRepository.findByStatus(status);  
 return credits.stream()  
 .map(this::mapToSpecificCreditDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<CreditDTO> getCreditsBetweenDates(Date startDate, Date endDate) {  
 List<Credit> credits = creditRepository.findCreditsBetweenDates(startDate, endDate);  
 return credits.stream()  
 .map(this::mapToSpecificCreditDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<CreditDTO> getCreditsAboveAmount(Double amount) {  
 List<Credit> credits = creditRepository.findByAmountGreaterThan(amount);  
 return credits.stream()  
 .map(this::mapToSpecificCreditDTO)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public CreditDTO updateCreditStatus(Long creditId, CreditStatus status) throws CreditNotFoundException {  
 Credit credit = creditRepository.findById(creditId)  
 .orElseThrow(() -> new CreditNotFoundException("Credit Not Found"));  
 credit.setStatus(status);  
 if(status == CreditStatus.*ACCEPTED*) {  
 credit.setAcceptanceDate(new Date());  
 }  
 Credit updatedCredit = creditRepository.save(credit);  
 return mapToSpecificCreditDTO(updatedCredit);  
 }  
  
 @Override  
 public PersonalCreditDTO savePersonalCredit(PersonalCreditDTO creditDTO) throws CustomerNotFoundException {  
 PersonalCredit credit = bankMapper.fromPersonalCreditDTO(creditDTO);  
 return saveCredit(credit, creditDTO.getCustomerId());  
 }  
  
 @Override  
 public ProfessionalCreditDTO saveProfessionalCredit(ProfessionalCreditDTO creditDTO) throws CustomerNotFoundException {  
 ProfessionalCredit credit = bankMapper.fromProfessionalCreditDTO(creditDTO);  
 return saveCredit(credit, creditDTO.getCustomerId());  
 }  
  
 @Override  
 public RealEstateCreditDTO saveRealEstateCredit(RealEstateCreditDTO creditDTO) throws CustomerNotFoundException {  
 RealEstateCredit credit = bankMapper.fromRealEstateCreditDTO(creditDTO);  
 return saveCredit(credit, creditDTO.getCustomerId());  
 }  
  
 @Override  
 public CreditSimulationResponseDTO simulateCredit(CreditSimulationRequestDTO simulationRequest) {  
 // Calculate monthly payment and total interest  
 double monthlyInterestRate = simulationRequest.getInterestRate() / 100 / 12;  
 int numberOfPayments = simulationRequest.getDurationMonths();  
  
 double monthlyPayment = (simulationRequest.getAmount() \* monthlyInterestRate) /  
 (1 - Math.*pow*(1 + monthlyInterestRate, -numberOfPayments));  
  
 double totalInterest = (monthlyPayment \* numberOfPayments) - simulationRequest.getAmount();  
  
 return new CreditSimulationResponseDTO(  
 monthlyPayment,  
 totalInterest,  
 monthlyPayment \* numberOfPayments  
 );  
 }  
  
 private <T extends CreditDTO> T saveCredit(Credit credit, Long customerId) throws CustomerNotFoundException {  
 Customer customer = customerRepository.findById(customerId)  
 .orElseThrow(() -> new CustomerNotFoundException("Customer Not Found"));  
 credit.setCustomer(customer);  
 credit.setApplicationDate(new Date());  
 credit.setStatus(CreditStatus.*IN\_PROGRESS*);  
  
 Credit savedCredit = creditRepository.save(credit);  
 return (T) mapToSpecificCreditDTO(savedCredit);  
 }  
  
 private CreditDTO mapToSpecificCreditDTO(Credit credit) {  
 if (credit instanceof PersonalCredit) {  
 return bankMapper.fromPersonalCredit((PersonalCredit) credit);  
 } else if (credit instanceof ProfessionalCredit) {  
 return bankMapper.fromProfessionalCredit((ProfessionalCredit) credit);  
 } else if (credit instanceof RealEstateCredit) {  
 return bankMapper.fromRealEstateCredit((RealEstateCredit) credit);  
 }  
 return bankMapper.fromCredit(credit);  
 }  
}

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.CustomerDTO;  
import com.example.exam\_final\_jee.entities.Customer;  
import com.example.exam\_final\_jee.exceptions.CustomerNotFoundException;  
import com.example.exam\_final\_jee.mappers.BankMapper;  
import com.example.exam\_final\_jee.repo.CustomerRepository;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@Transactional  
@AllArgsConstructor  
public class CustomerServiceImpl implements CustomerService {  
 private CustomerRepository customerRepository;  
 private BankMapper bankMapper;  
  
 @Override  
 public CustomerDTO saveCustomer(CustomerDTO customerDTO) {  
 Customer customer = bankMapper.fromCustomerDTO(customerDTO);  
 Customer savedCustomer = customerRepository.save(customer);  
 return bankMapper.fromCustomer(savedCustomer);  
 }  
  
 @Override  
 public List<CustomerDTO> listCustomers() {  
 List<Customer> customers = customerRepository.findAll();  
 return customers.stream()  
 .map(customer -> bankMapper.fromCustomer(customer))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public CustomerDTO getCustomer(Long customerId) throws CustomerNotFoundException {  
 Customer customer = customerRepository.findById(customerId)  
 .orElseThrow(() -> new CustomerNotFoundException("Customer Not Found"));  
 return bankMapper.fromCustomer(customer);  
 }  
  
 @Override  
 public CustomerDTO updateCustomer(CustomerDTO customerDTO) {  
 Customer customer = bankMapper.fromCustomerDTO(customerDTO);  
 Customer savedCustomer = customerRepository.save(customer);  
 return bankMapper.fromCustomer(savedCustomer);  
 }  
  
 @Override  
 public void deleteCustomer(Long customerId) {  
 customerRepository.deleteById(customerId);  
 }  
  
 @Override  
 public List<CustomerDTO> searchCustomers(String keyword) {  
 List<Customer> customers = customerRepository.searchCustomer("%"+keyword+"%");  
 return customers.stream()  
 .map(customer -> bankMapper.fromCustomer(customer))  
 .collect(Collectors.*toList*());  
 }  
}

package com.example.exam\_final\_jee.services;  
  
import com.example.exam\_final\_jee.dtos.RepaymentDTO;  
import com.example.exam\_final\_jee.entities.Credit;  
import com.example.exam\_final\_jee.entities.Repayment;  
import com.example.exam\_final\_jee.enums.RepaymentType;  
import com.example.exam\_final\_jee.exceptions.CreditNotFoundException;  
import com.example.exam\_final\_jee.exceptions.RepaymentNotFoundException;  
import com.example.exam\_final\_jee.mappers.BankMapper;  
import com.example.exam\_final\_jee.repo.CreditRepository;  
import com.example.exam\_final\_jee.repo.RepaymentRepository;  
import lombok.AllArgsConstructor;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.Date;  
import java.util.List;  
import java.util.stream.Collectors;  
  
@Service  
@Transactional  
@AllArgsConstructor  
public class RepaymentServiceImpl implements RepaymentService {  
 private RepaymentRepository repaymentRepository;  
 private CreditRepository creditRepository;  
 private BankMapper bankMapper;  
  
 @Override  
 public RepaymentDTO saveRepayment(RepaymentDTO repaymentDTO) throws CreditNotFoundException {  
 Credit credit = creditRepository.findById(repaymentDTO.getCreditId())  
 .orElseThrow(() -> new CreditNotFoundException("Credit Not Found"));  
  
 Repayment repayment = bankMapper.fromRepaymentDTO(repaymentDTO);  
 repayment.setCredit(credit);  
 repayment.setDate(new Date());  
  
 Repayment savedRepayment = repaymentRepository.save(repayment);  
 return bankMapper.fromRepayment(savedRepayment);  
 }  
  
 @Override  
 public List<RepaymentDTO> getRepaymentsByCredit(Long creditId) {  
 List<Repayment> repayments = repaymentRepository.findByCreditId(creditId);  
 return repayments.stream()  
 .map(repayment -> bankMapper.fromRepayment(repayment))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<RepaymentDTO> getRepaymentsByType(RepaymentType type) {  
 List<Repayment> repayments = repaymentRepository.findByType(type);  
 return repayments.stream()  
 .map(repayment -> bankMapper.fromRepayment(repayment))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<RepaymentDTO> getRepaymentsBetweenDates(Date startDate, Date endDate) {  
 List<Repayment> repayments = repaymentRepository.findRepaymentsBetweenDates(startDate, endDate);  
 return repayments.stream()  
 .map(repayment -> bankMapper.fromRepayment(repayment))  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public Double getTotalRepaidAmount(Long creditId) {  
 return repaymentRepository.getTotalRepaidAmountByCreditId(creditId);  
 }  
  
 @Override  
 public RepaymentDTO getRepayment(Long repaymentId) throws RepaymentNotFoundException {  
 Repayment repayment = repaymentRepository.findById(repaymentId)  
 .orElseThrow(() -> new RepaymentNotFoundException("Repayment Not Found"));  
 return bankMapper.fromRepayment(repayment);  
 }  
  
 @Override  
 public void deleteRepayment(Long repaymentId) {  
 repaymentRepository.deleteById(repaymentId);  
 }  
}

**- Execptions:**

package com.example.exam\_final\_jee.exceptions;  
  
public class CreditNotFoundException extends Exception {  
 public CreditNotFoundException(String message) {  
 super(message);  
 }  
}

package com.example.exam\_final\_jee.exceptions;  
  
public class CustomerNotFoundException extends Exception {  
 public CustomerNotFoundException(String message) {  
 super(message);  
 }  
}

package com.example.exam\_final\_jee.exceptions;  
  
public class RepaymentNotFoundException extends Exception {  
 public RepaymentNotFoundException(String message) {  
 super(message);  
 }  
}

# Créer les Web services:

# Application frontend en utilisant Angular:

# Sécuriser:

# Apporter des améliorations additionnelles à votre projet: