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

<u>Filière :</u> II-BDCC 2025-2026

Cycle d'ingénieurs

I. Sommaire

Contents

II. Conception :

1. 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)

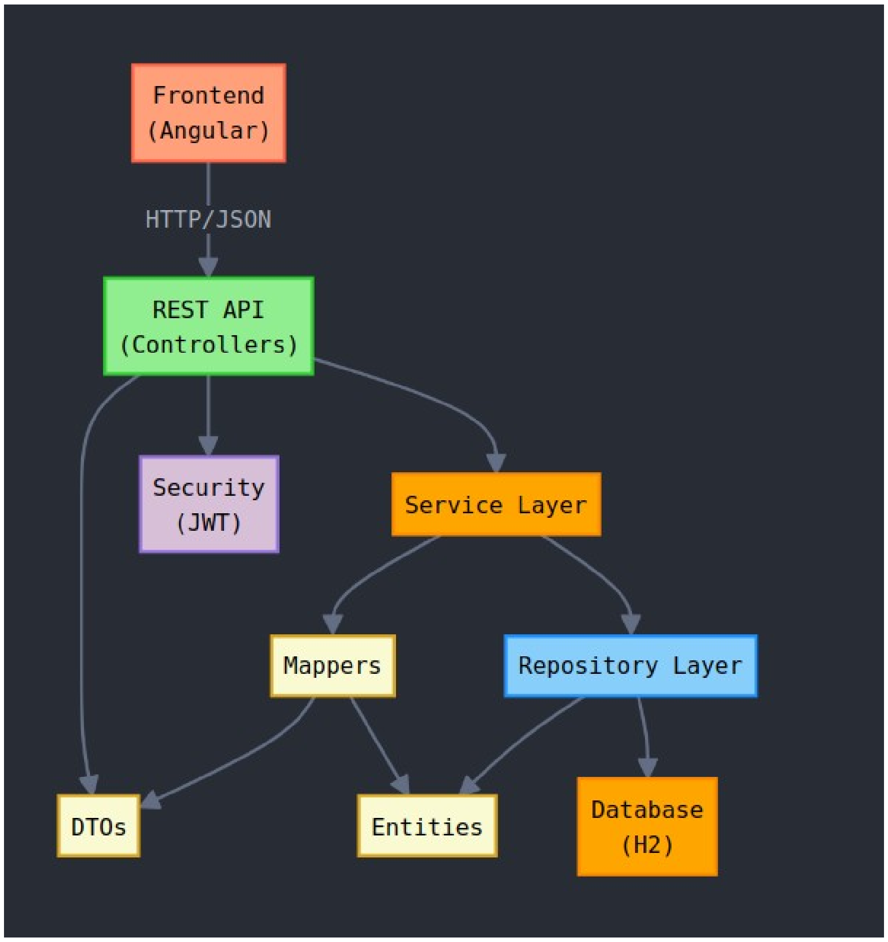
- **Rôle** : Représente le modèle de domaine et la structure de la base de données
- **Fonctionnalités** :
    - Correspond aux tables de la base de données via les annotations JPA
    - Définit les relations entre les objets du domaine
    - Encapsule la logique de domaine liée à l'entité
- **Caractéristiques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

8. Couche Repository

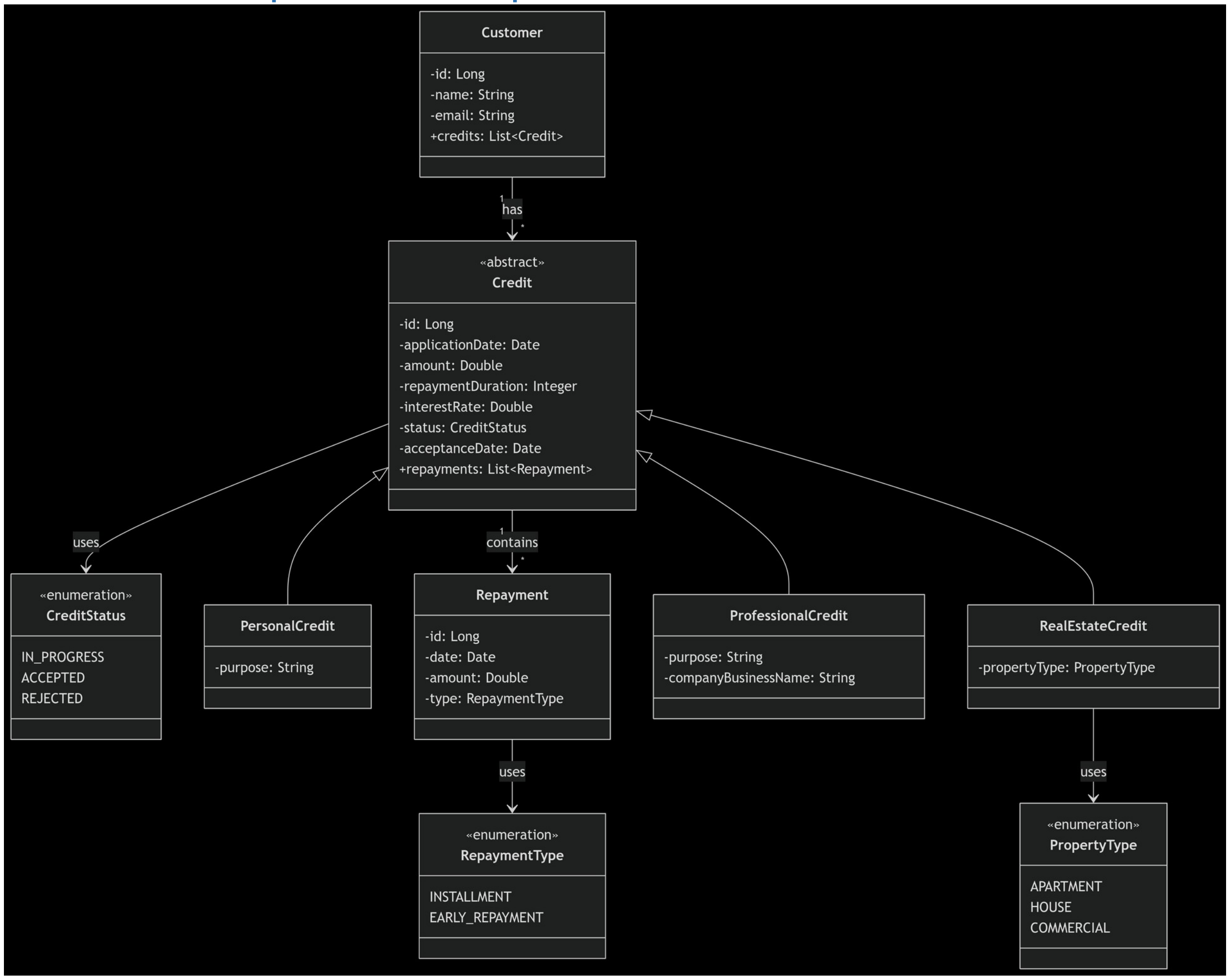
- **Technologie** : Spring Data JPA
- **Rôle** : Gère les opérations d'accès aux données
- **Fonctionnalités** :
    - Fournit des opérations CRUD pour les entités
    - Traduit entre les objets Java et les enregistrements de base de données
    - Gère les requêtes et les transactions de base de données
- **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



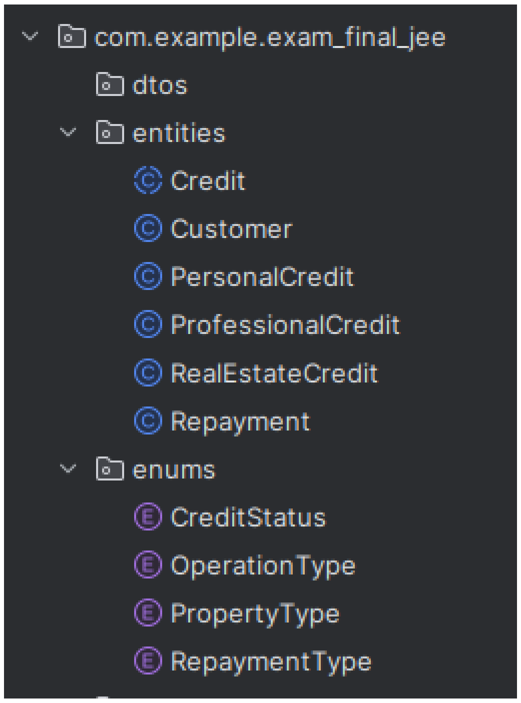
2. Établir un diagramme de classes qui montre les entités. On ne représentera que les attributs:



II. Implémentation:

1. Couche DAO:

a. Créer les entités JPA



```
com.example.exam_final_jee
    dtos
    entities
        Credit
        Customer
        PersonalCredit
        ProfessionalCredit
        RealEstateCredit
        Repayment
    enums
        CreditStatus
        OperationType
        PropertyType
        RepaymentType
```

- Credit

```java
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 :

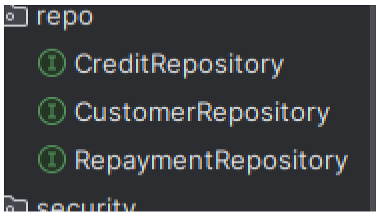
```
package com.example.exam_final_jee.entities;

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
```

```java
@DiscriminatorValue("REAL_ESTATE")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class RealEstateCredit extends Credit {
    @Enumerated(EnumType.STRING)
    private PropertyType propertyType; // APARTMENT, HOUSE, COMMERCIAL
}
```

```java
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;
}
```

```java
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.
}
```

b. 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

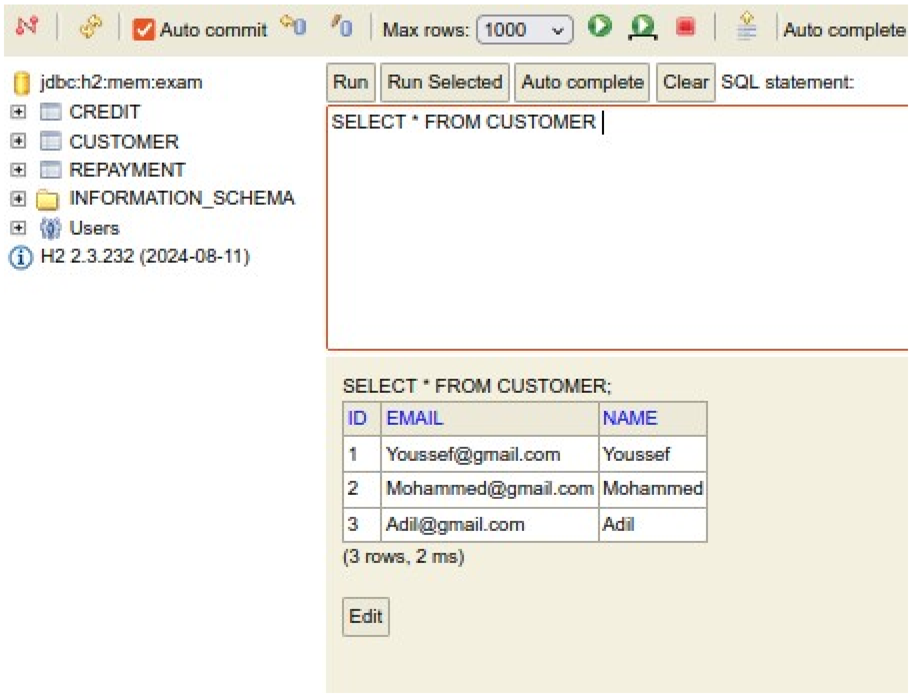
```
package com.example.exam_final_jee.repo;

import com.example.exam_final_jee.entities.Customer;
```

```java
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

```java
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);
}
```

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



```
SELECT * FROM CUSTOMER
```

```
SELECT * FROM CUSTOMER;
```

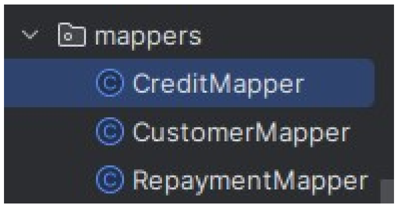
ID	EMAIL	NAME
1	Youssef@gmail.com	Youssef
2	Mohammed@gmail.com	Mohammed
3	Adil@gmail.com	Adil

(3 rows, 2 ms)

```java
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);
			});
		};
	}
}
```

2. Créer une couche service:

d. 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;
```

```java
// RealEstateCreditDTO.java
@Data
public class RealEstateCreditDTO extends CreditDTO {
    private PropertyType propertyType;
}
```

```java
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;
}
```

```java
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

```java
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

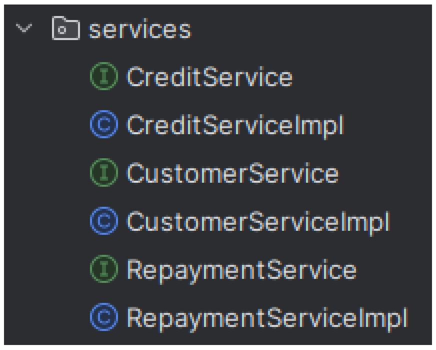
```java
package com.example.exam_final_jee.dtos;

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

```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 {
```

```java
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;
    }
}
```

e. Couch Service



```
services
    CreditService
    CreditServiceImpl
    CustomerService
    CustomerServiceImpl
    RepaymentService
    RepaymentServiceImpl
```

- 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;
```

```java
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 {
```

```java
        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);
    }
}
```

```java
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;
```

```java
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);
    }
}
```

3. Créer les Web services:

- RepaymentController

```
package com.example.exam_final_jee.web;

import com.example.exam_final_jee.dtos.RepaymentDTO;
import com.example.exam_final_jee.enums.RepaymentType;
```

```java
import com.example.exam_final_jee.exceptions.CreditNotFoundException;
import com.example.exam_final_jee.exceptions.RepaymentNotFoundException;
import com.example.exam_final_jee.services.RepaymentService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.List;

@RestController
@RequestMapping("/api/repayments")
@AllArgsConstructor
@CrossOrigin("*")
public class RepaymentController {
    private RepaymentService repaymentService;

    @PostMapping
    public RepaymentDTO saveRepayment(@RequestBody RepaymentDTO repaymentDTO) throws
CreditNotFoundException {
        return repaymentService.saveRepayment(repaymentDTO);
    }

    @GetMapping("/credit/{creditId}")
    public List<RepaymentDTO> getCreditRepayments(@PathVariable Long creditId) {
        return repaymentService.getRepaymentsByCredit(creditId);
    }

    @GetMapping("/type/{type}")
    public List<RepaymentDTO> getRepaymentsByType(@PathVariable RepaymentType type) {
        return repaymentService.getRepaymentsByType(type);
    }

    @GetMapping("/between-dates")
    public List<RepaymentDTO> getRepaymentsBetweenDates(@RequestParam Date startDate,
@RequestParam Date endDate) {
        return repaymentService.getRepaymentsBetweenDates(startDate, endDate);
    }

    @GetMapping("/total-repaid/{creditId}")
    public Double getTotalRepaidAmount(@PathVariable Long creditId) {
        return repaymentService.getTotalRepaidAmount(creditId);
    }

    @GetMapping("/{id}")
    public RepaymentDTO getRepayment(@PathVariable Long id) throws
RepaymentNotFoundException {
        return repaymentService.getRepayment(id);
    }

    @DeleteMapping("/{id}")
    public void deleteRepayment(@PathVariable Long id) {
        repaymentService.deleteRepayment(id);
    }
}
```

- CustomerControlelr

```
package com.example.exam_final_jee.web;

import com.example.exam_final_jee.dtos.CustomerDTO;
import com.example.exam_final_jee.exceptions.CustomerNotFoundException;
import com.example.exam_final_jee.services.CustomerService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/customers")
@AllArgsConstructor
@CrossOrigin("*")
public class CustomerController {
    private CustomerService customerService;

    @GetMapping
    public List<CustomerDTO> customers() {
        return customerService.listCustomers();
    }

    @GetMapping("/{id}")
    public CustomerDTO getCustomer(@PathVariable Long id) throws CustomerNotFoundException {
        return customerService.getCustomer(id);
    }

    @PostMapping
    public CustomerDTO saveCustomer(@RequestBody CustomerDTO customerDTO) {
        return customerService.saveCustomer(customerDTO);
```

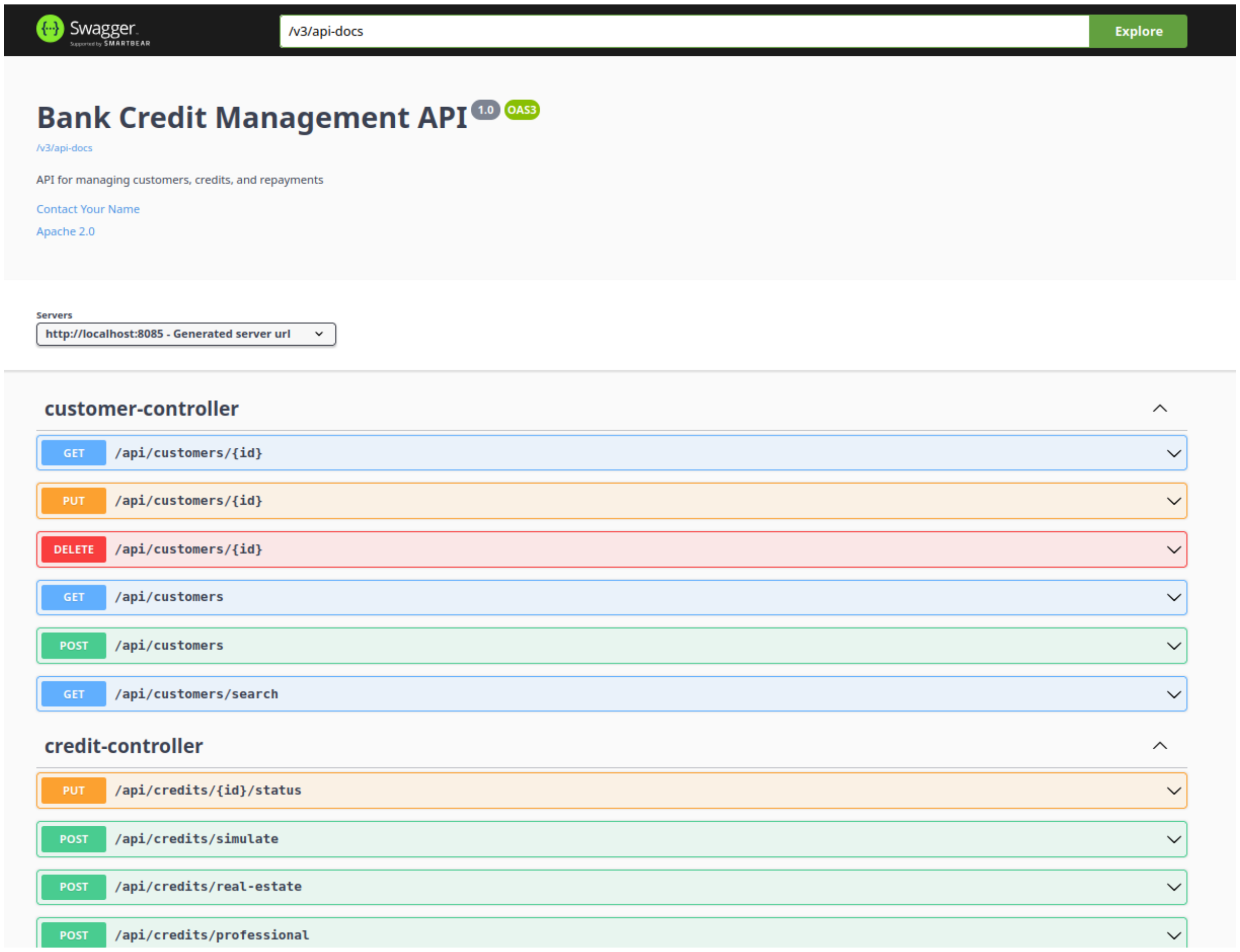
```
  }

  @PutMapping("/{id}")
  public CustomerDTO updateCustomer(@PathVariable Long id, @RequestBody CustomerDTO
customerDTO) {
      customerDTO.setId(id);
      return customerService.updateCustomer(customerDTO);
  }

  @DeleteMapping("/{id}")
  public void deleteCustomer(@PathVariable Long id) {
      customerService.deleteCustomer(id);
  }

  @GetMapping("/search")
  public List<CustomerDTO> searchCustomers(@RequestParam(name = "keyword", defaultValue =
"") String keyword) {
      return customerService.searchCustomers(keyword);
  }
}
```

- CreditController

```
package com.example.exam_final_jee.web;

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 com.example.exam_final_jee.services.CreditService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.List;

@RestController
@RequestMapping("/api/credits")
@AllArgsConstructor
@CrossOrigin("*")
public class CreditController {
  private CreditService creditService;
  @GetMapping
  public List<CreditDTO> credits() {
     return creditService.listCredits();
  }

  @GetMapping("/{id}")
  public CreditDTO getCredit(@PathVariable Long id) throws CreditNotFoundException {
     return creditService.getCredit(id);
  }

  @DeleteMapping("/{id}")
  public void deleteCredit(@PathVariable Long id) {
     creditService.deleteCredit(id);
  }

  @GetMapping("/customer/{customerId}")
  public List<CreditDTO> getCustomerCredits(@PathVariable Long customerId) throws
CustomerNotFoundException {
     return creditService.getCreditsByCustomer(customerId);
  }

  @GetMapping("/status/{status}")
  public List<CreditDTO> getCreditsByStatus(@PathVariable CreditStatus status) {
     return creditService.getCreditsByStatus(status);
  }

  @GetMapping("/between-dates")
  public List<CreditDTO> getCreditsBetweenDates(@RequestParam Date startDate,
@RequestParam Date endDate) {
     return creditService.getCreditsBetweenDates(startDate, endDate);
  }

  @GetMapping("/above-amount")
  public List<CreditDTO> getCreditsAboveAmount(@RequestParam Double amount) {
     return creditService.getCreditsAboveAmount(amount);
  }

  @PutMapping("/{id}/status")
  public CreditDTO updateCreditStatus(@PathVariable Long id, @RequestParam CreditStatus
status) throws CreditNotFoundException {
     return creditService.updateCreditStatus(id, status);
  }
```

```java
    @PostMapping("/personal")
    public PersonalCreditDTO savePersonalCredit(@RequestBody PersonalCreditDTO creditDTO)
throws CustomerNotFoundException {
        return creditService.savePersonalCredit(creditDTO);
    }

    @PostMapping("/professional")
    public ProfessionalCreditDTO saveProfessionalCredit(@RequestBody ProfessionalCreditDTO
creditDTO) throws CustomerNotFoundException {
        return creditService.saveProfessionalCredit(creditDTO);
    }

    @PostMapping("/real-estate")
    public RealEstateCreditDTO saveRealEstateCredit(@RequestBody RealEstateCreditDTO
creditDTO) throws CustomerNotFoundException {
        return creditService.saveRealEstateCredit(creditDTO);
    }

    @PostMapping("/simulate")
    public CreditSimulationResponseDTO simulateCredit(@RequestBody
CreditSimulationRequestDTO simulationRequest) {
        return creditService.simulateCredit(simulationRequest);
    }
}
```

f. Test using Open API Doc


Swagger

/v3/api-docs

Explore

Bank Credit Management API 1.0 OAS3

/v3/api-docs

API for managing customers, credits, and repayments

Contact Your Name

Apache 2.0

Servers

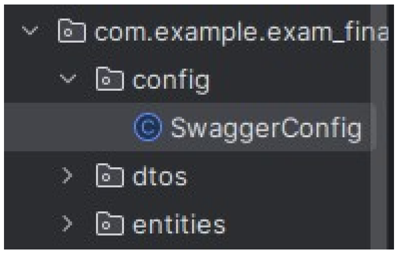
http://localhost:8085 - Generated server url

customer-controller

```
GET     /api/customers/{id}
PUT     /api/customers/{id}
DELETE  /api/customers/{id}
GET     /api/customers
POST    /api/customers
GET     /api/customers/search
```

credit-controller

```
PUT     /api/credits/{id}/status
POST    /api/credits/simulate
POST    /api/credits/real-estate
POST    /api/credits/professional
```



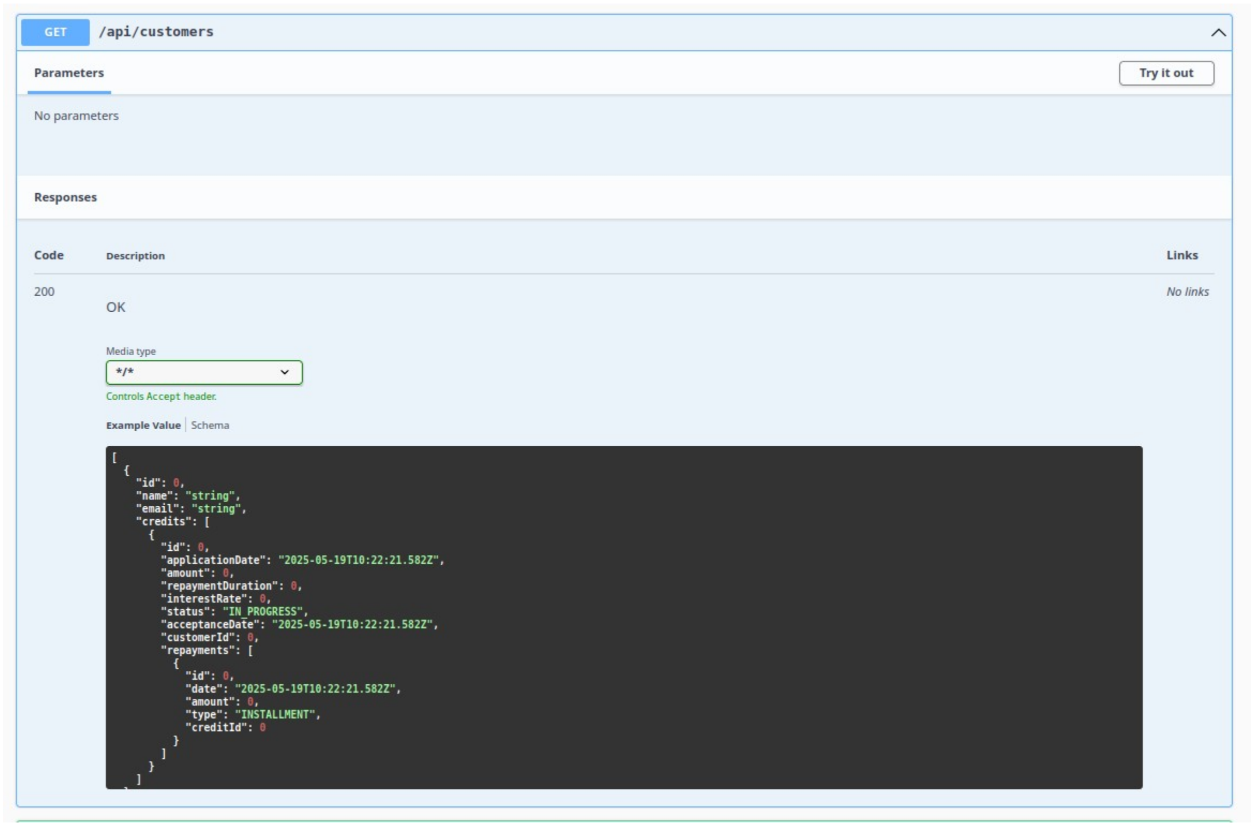
```
package com.example.exam_final_jee;

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;
```

```java
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);
			});
		};
	}
}
```

add dependcy to **pom.xml**

```xml
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-ui</artifactId>
	<version>1.8.0</version>
</dependency>
```

test


GET /api/customers

Parameters

No parameters

Responses

Code	Description	Links
200	OK	No links

Media type: */*

Controls Accept header.

Example Value | Schema

```
[
  {
    "id": 0,
    "name": "string",
    "email": "string",
    "credits": [
      {
        "id": 0,
        "applicationDate": "2025-05-19T10:22:21.582Z",
        "amount": 0,
        "repaymentDuration": 0,
        "interestRate": 0,
        "status": "IN_PROGRESS",
        "acceptanceDate": "2025-05-19T10:22:21.582Z",
        "customerId": 0,
        "repayments": [
          {
            "id": 0,
            "date": "2025-05-19T10:22:21.582Z",
            "amount": 0,
            "type": "INSTALLMENT",
            "creditId": 0
          }
        ]
      }
    ]
```

Request 200 OK

4. Application frontend en utilisant Angular:

Npm new e-bank-fontend

npm install bootstrap --save

npm install bootstrap-icons --save

5. Sécuriser:

```
package q.jv.digital_banking_app.web;

import org.springframework.web.bind.annotation.*;
import q.jv.digital_banking_app.dtos.*;
import q.jv.digital_banking_app.exceptions.BalanceNotSufficientException;
import q.jv.digital_banking_app.exceptions.BankAccountNotFoundException;
import q.jv.digital_banking_app.services.BankAccountService;

import java.util.List;
```

```
@RestController
@CrossOrigin("*")
public class BankAccountRestController {
   private BankAccountService bankAccountService;

   public BankAccountRestController(BankAccountService bankAccountService) {
      this.bankAccountService = bankAccountService;
   }

   @GetMapping("/accounts/{accountId}")
   public BankAccountDTO getBankAccount(@PathVariable String accountId) throws
BankAccountNotFoundException {
      return bankAccountService.getBankAccount(accountId);
   }
   @GetMapping("/accounts")
   public List<BankAccountDTO> listAccounts(){
      return bankAccountService.bankAccountList();
   }

   @GetMapping("/accounts/{accountId}/operations")
   public List<AccountOperationDTO> getHistory(@PathVariable String accountId){
      return bankAccountService.accountHistory(accountId);
   }

   @GetMapping("/accounts/{accountId}/pageOperations")
   public AccountHistoryDTO getAccountHistory(
         @PathVariable String accountId,
         @RequestParam(name="page",defaultValue = "0") int page,
         @RequestParam(name="size",defaultValue = "5")int size) throws
BankAccountNotFoundException {
      return bankAccountService.getAccountHistory(accountId,page,size);
   }

   @PostMapping("/accounts/debit")
   public DebitDTO debit(@RequestBody DebitDTO debitDTO) throws
BankAccountNotFoundException, BalanceNotSufficientException {

this.bankAccountService.debit(debitDTO.getAccountId(),debitDTO.getAmount(),debitDTO.getDescri
ption());
      return debitDTO;
   }
   @PostMapping("/accounts/credit")
   public CreditDTO credit(@RequestBody CreditDTO creditDTO) throws
BankAccountNotFoundException {
```

```
this.bankAccountService.credit(creditDTO.getAccountId(),creditDTO.getAmount(),creditDTO.getDes
cription());
        return creditDTO;
    }
    @PostMapping("/accounts/transfer")
    public void transfer(@RequestBody TransferRequestDTO transferRequestDTO) throws
BankAccountNotFoundException, BalanceNotSufficientException {
        this.bankAccountService.transfer(
                transferRequestDTO.getAccountSource(),
                transferRequestDTO.getAccountDestination(),
                transferRequestDTO.getAmount());
    }
}
```

```
package q.jv.digital_banking_app.web;


import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import q.jv.digital_banking_app.dtos.CustomerDTO;
import q.jv.digital_banking_app.exceptions.CustomerNotFoundException;
import q.jv.digital_banking_app.services.BankAccountService;

import java.util.List;

@RestController
@CrossOrigin("*")
public class CustomerRestController {
    private BankAccountService bankAccountService;

    public CustomerRestController(BankAccountService bankAccountService) {
        this.bankAccountService = bankAccountService;
    }

    @GetMapping("/customers")
    @PreAuthorize("hasAuthority('SCOPE_USER')")
    public List<CustomerDTO> customers(){
        return bankAccountService.listCustomers();
    }
    @GetMapping("/customers/{id}")
    @PreAuthorize("hasAuthority('SCOPE_USER')")
    public CustomerDTO getCustomer(@PathVariable(name = "id") Long customerId) throws
CustomerNotFoundException {
        return bankAccountService.getCustomer(customerId);
    }
    @PostMapping("/customers")
    @PreAuthorize("hasAuthority('SCOPE_ADMIN')")
    public CustomerDTO saveCustomer(@RequestBody CustomerDTO customerDTO){
        return bankAccountService.saveCustomer(customerDTO);
    }
    @PutMapping("/customers/{customerId}")
    @PreAuthorize("hasAuthority('SCOPE_ADMIN')")
    public CustomerDTO updateCustomer(@PathVariable Long customerId, @RequestBody
CustomerDTO customerDTO){
        customerDTO.setId(customerId);
        return bankAccountService.updateCustomer(customerDTO);
    }
    @DeleteMapping("/customers/{id}")
    @PreAuthorize("hasAuthority('SCOPE_ADMIN')")
    public void deleteCustomer(@PathVariable Long id){
        bankAccountService.deleteCustomer(id);
    }

    @GetMapping("/customers/search")
    @PreAuthorize("hasAuthority('SCOPE_USER')")
    public List<CustomerDTO> searchCustomers(@RequestParam(name = "keyword",defaultValue =
"") String keyword){
        return bankAccountService.searchCustomers("%"+keyword+"%");
    }
}
```

6. Apporter des améliorations additionnelles à votre projet:

- **Authentification forte** :
  Ajouter la prise en charge de l'authentification multi-facteurs (MFA) ou des passkeys (clés d'accès sans mot de passe) pour renforcer la sécurité.

- **Chiffrement des données sensibles** :
  Mettre en œuvre le chiffrement au repos (data-at-rest) et en transit (data-in-transit) pour les informations sensibles.
- **Journalisation et audit** :
  Ajouter un système de journalisation des connexions, actions sensibles, et erreurs afin de pouvoir auditer les activités des utilisateurs.
- **Détection des comportements suspects** :
  Intégrer un système de détection d'intrusions ou d'activités inhabituelles (login à des heures improbables, changements soudains de rôle, etc.).
- **Utilisation de la Blockchain (si pertinent)** :
  Pour renforcer la traçabilité et l'intégrité, notamment pour les actions critiques (ex. : modifications de tickets), utiliser une chaîne de blocs pour enregistrer certains événements.
- **Contrôle d'accès basé sur les rôles avancé (RBAC/ABAC)** :
  Mettre en place un contrôle d'accès plus granulaire, basé sur les attributs ou les contextes d'utilisation.
- **Tests de sécurité automatisés** :
  Intégrer des tests de vulnérabilité ou de sécurité automatisés (ex. : Snyk, OWASP ZAP) dans votre pipeline CI/CD.
- **Protection contre les attaques courantes** :
  Implémenter une protection contre le CSRF, XSS, et les injections SQL ou JWT malveillants.
- **Monitoring en temps réel** :
  Ajouter un tableau de bord de supervision en temps réel des accès, erreurs critiques, ou pics d'utilisation.
- **Limiter les tentatives de connexion** :
  Implémenter un système de blocage temporaire après plusieurs tentatives de connexion échouées pour éviter les attaques par force brute.
- **Utiliser des standards de sécurité modernes** :
  Respecter les recommandations de l'OWASP et suivre les standards comme OAuth 2.1, OpenID Connect, etc.
- **Versionnage et gestion des secrets** :
  Utiliser des outils comme HashiCorp Vault ou AWS Secrets Manager pour stocker et gérer les clés secrètes au lieu de les coder en dur.