ECOLE NORMALE SUPÉRIURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

# Examen Architecture Distribuée et Middleware


**Réalisé par :**

**Bektachi Souhail**

**2GLSID**


Année Professionnelle : 2024/2025

# Introduction :

Ce rapport documente la conception et le développement d'une application Web JEE pour la gestion des crédits bancaires, réalisée dans le cadre d'une évaluation. L'application, bâtie sur les frameworks Spring et Angular, vise à gérer les clients, les différents types de crédits (Personnel, Immobilier, Professionnel) ainsi que leurs remboursements. Le document expose l'architecture technique, la modélisation des données, l'implémentation des couches métier et d'accès aux données avec Spring Boot, la création des services web REST, et la sécurisation des accès via Spring Security et JWT.

# Conception :

## 1. Architecture Technique du Projet :

L'architecture du projet est une application Web multicouche. Un Frontend Angular interagit avec un Backend JEE (Spring Boot) via des API REST.

Le Backend comprend :

- Couche Sécurité (Spring Security, JWT) pour l'authentification et l'autorisation.
- Couche Web (Spring MVC REST Controllers) pour exposer les API.
- Couche Service pour la logique métier.
- Couche DAO (Spring Data, JPA, Hibernate) pour l'accès aux données.

Les données sont persistées dans une Base de Données MySQL (potentiellement conteneurisée avec Docker).

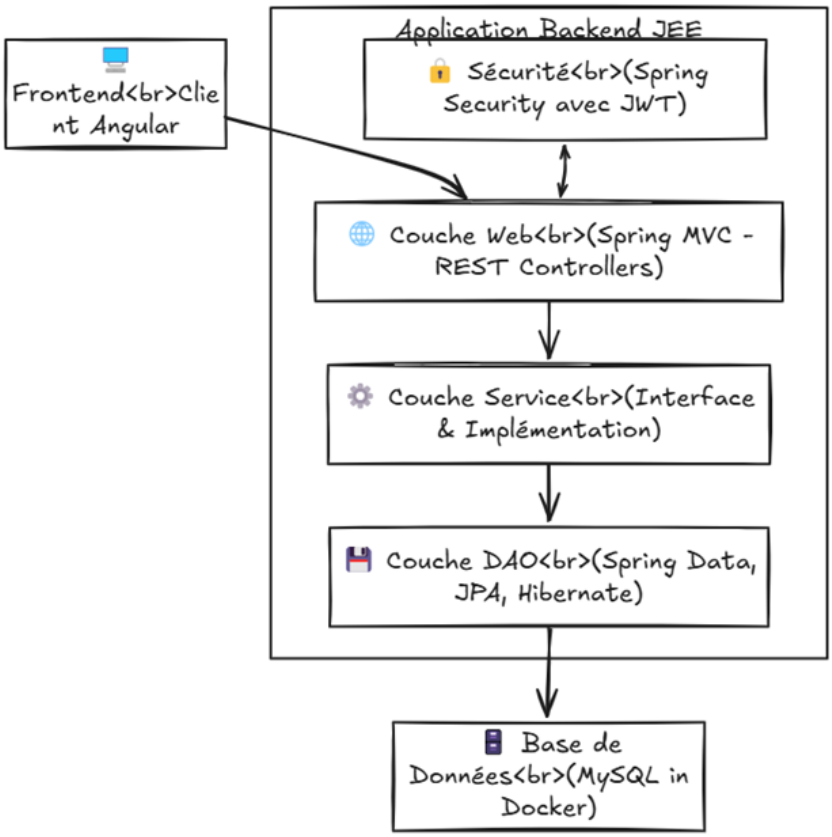



*Figure 1 Architecture Technique de l'Application Web JEE et Angular*

Compose.yml

```yaml
services:
  db:
    image: mysql:latest
    container_name: mysql_db
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: your_root_password
      MYSQL_DATABASE: your_database_name
      MYSQL_USER: your_user
      MYSQL_PASSWORD: your_user_password
    ports:
      - "3306:3306"
    volumes:
      - mysql_data:/var/lib/mysql

volumes:
  mysql_data:
```

## 2. Diagramme de Classes des Entités :

Le diagramme de classes ci-dessous (Figure 2) représente les entités JPA du domaine de l'application de gestion de crédits bancaires, en se concentrant sur leurs attributs. Il illustre les relations entre les entités principales : Client, Credit (et ses spécialisations CreditPersonnel, CreditImmobilier, CreditProfessionnel), et Remboursement, ainsi que les énumérations associées (StatutCredit, TypeBienImmobilier, TypeRemboursement). La classe Credit est définie comme abstraite et sert de base pour les différents types de crédits.

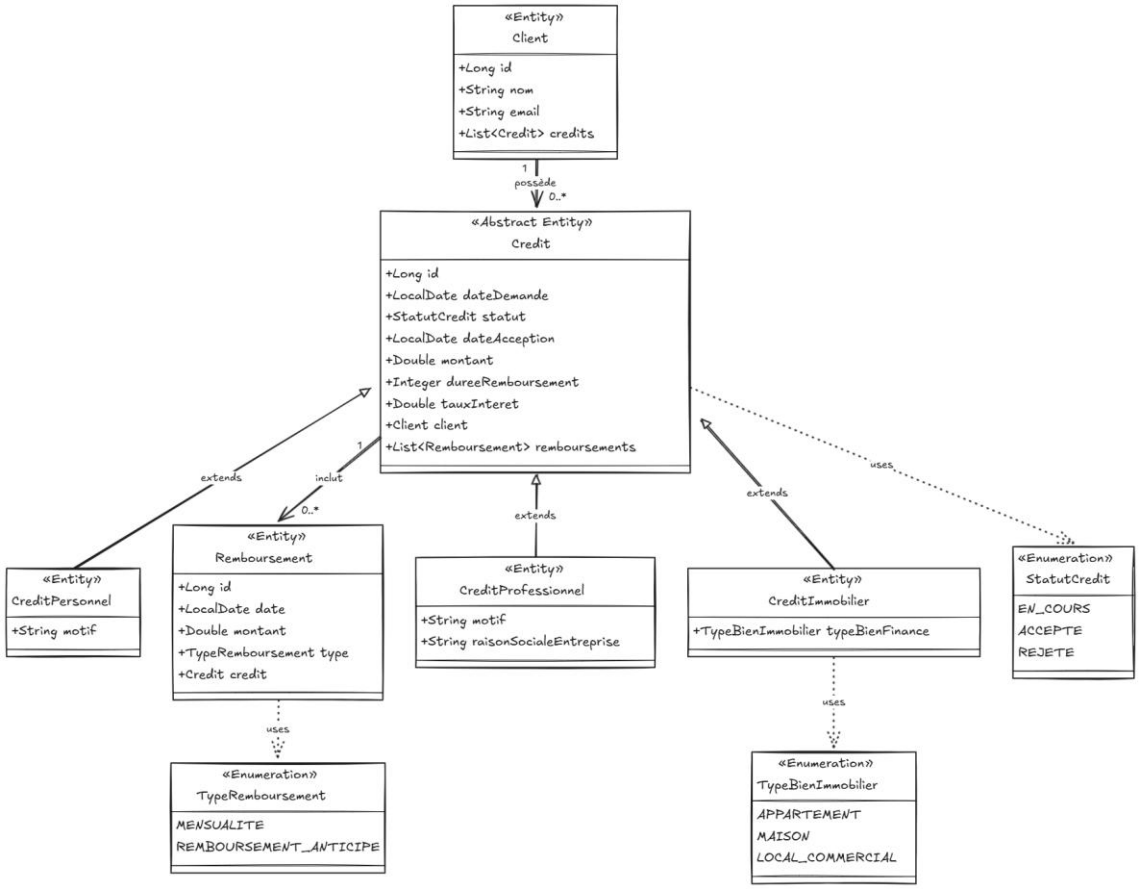



*Figure 2 Diagramme de Classes des Entités JPA*

# Implémentation :

## 1. Couche DAO :

Nous avons défini plusieurs entités JPA représentant les concepts métiers de l'application de gestion de crédits bancaires. Voici un aperçu des principales entités :

Client

```
package com.souhailbektachi.backend.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.ArrayList;
import java.util.List;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
```

```java
public class Client {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String nom;
    private String email;

    @OneToMany(mappedBy = "client", cascade = CascadeType.ALL)
    private List<Credit> credits = new ArrayList<>();
}
```

Credit (classe abstraite):

```java
package com.souhailbektachi.backend.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;
import java.util.ArrayList;
import java.util.List;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Credit {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate dateDemande;

    @Enumerated(EnumType.STRING)
    private StatutCredit statut;

    private LocalDate dateAcception;
    private Double montant;
    private Integer dureeRemboursement;
    private Double tauxInteret;

    @ManyToOne
    @JoinColumn(name = "client_id")
    private Client client;

    @OneToMany(mappedBy = "credit", cascade = CascadeType.ALL)
    private List<Remboursement> remboursements = new ArrayList<>();
}
```

CréditPersonnel:

```
package com.souhailbektachi.backend.entities;

import jakarta.persistence.Entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreditPersonnel extends Credit {
    private String motif;
}
```

CreditImmobilier

```
package com.souhailbektachi.backend.entities;

import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreditImmobilier extends Credit {
    @Enumerated(EnumType.STRING)
    private TypeBienImmobilier typeBienFinance;
}
```

CreditProfessionnel:

```
package com.souhailbektachi.backend.entities;

import jakarta.persistence.Entity;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class CreditProfessionnel extends Credit {
    private String motif;
    private String raisonSocialeEntreprise;
}
```

Remboursement:

```
package com.souhailbektachi.backend.entities;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.LocalDate;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Remboursement {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private LocalDate date;
    private Double montant;

    @Enumerated(EnumType.STRING)
    private TypeRemboursement type;

    @ManyToOne
    @JoinColumn(name = "credit_id")
    private Credit credit;
}
```

Enums :

```
public enum StatutCredit {
    EN_COURS,
    ACCEPTE,
    REJETE
}
public enum TypeBienImmobilier {
    APPARTEMENT,
    MAISON,
    LOCAL_COMMERCIAL
}
public enum TypeRemboursement {
    MENSUALITE,
    REMBOURSEMENT_ANTICIPE
}
```

## 2. Création des interfaces JPA Repository :

Les interfaces ci-dessous héritent de JpaRepository, ce qui permet de profiter des fonctionnalités de Spring Data JPA pour effectuer des opérations CRUD sans implémentation manuelle.

**Liste des interfaces créées :**

**ClientRepository :**

```
@Repository
public interface ClientRepository extends JpaRepository<Client, Long> {
    Optional<Client> findByEmail(String email);
    List<Client> findByNomContainingIgnoreCase(String keyword);
    List<Client> findByEmailContainingIgnoreCase(String email);
}
```

**CreditRepository**

```
@Repository
public interface CreditRepository extends JpaRepository<Credit, Long> {
    List<Credit> findByClientId(Long clientId);
    List<Credit> findByStatut(StatutCredit statut);

    @Query("SELECT c FROM CreditPersonnel c")
    List<Credit> findCreditPersonnel();

    @Query("SELECT c FROM CreditImmobilier c")
    List<Credit> findCreditImmobilier();

    @Query("SELECT c FROM CreditProfessionnel c")
    List<Credit> findCreditProfessionnel();

    List<Credit> findByMontantBetween(Double minAmount, Double maxAmount);
    List<Credit> findByMontantGreaterThanEqual(Double minAmount);
    List<Credit> findByMontantLessThanEqual(Double maxAmount);

    List<Credit> findByDateDemandeBetween(LocalDate startDate, LocalDate
endDate);
    List<Credit> findByDateDemandeGreaterThanEqual(LocalDate startDate);
    List<Credit> findByDateDemandeLessThanEqual(LocalDate endDate);
}
```

**CreditPersonnelRepository :**

```
@Repository
public interface CreditPersonnelRepository extends
JpaRepository<CreditPersonnel, Long> {
    // You can add custom query methods here
}
```

**CreditImmobilierRepository :**

```
@Repository
public interface CreditImmobilierRepository extends
JpaRepository<CreditImmobilier, Long> {
    List<CreditImmobilier> findByTypeBienFinance(TypeBienImmobilier
typeBienFinance);
}
```

**CreditProfessionnelRepository :**

```
@Repository
public interface CreditProfessionnelRepository extends
JpaRepository<CreditProfessionnel, Long> {
    List<CreditProfessionnel> findByRaisonSocialeEntrepriseContaining(String
keyword);
}
```

RemboursementRepository :

```
@Repository
public interface RemboursementRepository extends JpaRepository<Remboursement,
Long> {
    List<Remboursement> findByCreditId(Long creditId);
    List<Remboursement> findByType(TypeRemboursement type);

    List<Remboursement> findByDateBetween(LocalDate startDate, LocalDate
endDate);
    List<Remboursement> findByDateGreaterThanEqual(LocalDate startDate);
    List<Remboursement> findByDateLessThanEqual(LocalDate endDate);

    List<Remboursement> findByMontantBetween(Double minAmount, Double
maxAmount);
    List<Remboursement> findByMontantGreaterThanEqual(Double minAmount);
    List<Remboursement> findByMontantLessThanEqual(Double maxAmount);
}
```

## 3. Couche Service (DTOs, Mappers, Services) :

La couche service constitue la couche métier de l'application. Elle permet de séparer la logique métier des autres couches (DAO, Web) et facilite la maintenance, la réutilisation du code, ainsi que la sécurité et les tests unitaires.

Elle est composée de trois éléments clés :

- Les DTOs : pour transporter des données entre les couches.
- Les Mappers : pour convertir entre les entités JPA et les DTOs.
- Les Services : pour implémenter la logique métier via des interfaces et des classes concrètes.

### a. Création des DTOs :

Nous avons créé les DTOs suivants :

- ClientDTO, ClientRequestDTO, ClientSummaryDTO
- CreditDTO, CreditRequestDTO, CreditSummaryDTO
- CreditPersonnelDTO, CreditImmobilierDTO, CreditProfessionnelDTO
- CreditPersonnelRequestDTO, CreditImmobilierRequestDTO, CreditProfessionnelRequestDTO
- RemboursementDTO, RemboursementRequestDTO

### b. Exemple :

ClientDto :

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class ClientDTO {
    private Long id;
    private String nom;
    private String email;
    private List<CreditSummaryDTO> credits;
}
```

CreditPersonnelRequestDTO:

```
@Getter
@Setter
@NoArgsConstructor
public class CreditPersonnelRequestDTO extends CreditRequestDTO {
    private String motif;
}
```

Chaque DTO est conçu pour ne contenir que les données pertinentes à une opération (création, affichage, résumé…).

### c. Création des Mappers :

Les mappers permettent de transformer automatiquement les entités JPA en DTOs et vice versa. Les classes de mapping suivantes ont été créées :

- ClientMapper.java
- CreditMapper.java
- RemboursementMapper.java

### d. Exemple :

ClientMapper :

```
@Component
public class ClientMapper {

    private final CreditMapper creditMapper;

    @Autowired
    public ClientMapper(CreditMapper creditMapper) {
        this.creditMapper = creditMapper;
    }

    /**
     * Convert a Client entity to a ClientDTO
     */
    public ClientDTO toDto(Client client) {
        if (client == null) {
            return null;
        }

        ClientDTO dto = new ClientDTO();
        dto.setId(client.getId());
        dto.setNom(client.getNom());
        dto.setEmail(client.getEmail());

        // Map credits if they exist
        if (client.getCredits() != null && !client.getCredits().isEmpty()) {
            List<CreditSummaryDTO> creditSummaries =
client.getCredits().stream()
                    .map(creditMapper::toSummaryDto)
                    .collect(Collectors.toList());
            dto.setCredits(creditSummaries);
        } else {
            dto.setCredits(Collections.emptyList());
        }

        return dto;
    }

    /**
     * Convert a Client entity to a ClientSummaryDTO with credit count
     */
    public ClientSummaryDTO toSummaryDto(Client client) {
        if (client == null) {
            return null;
        }

        ClientSummaryDTO dto = new ClientSummaryDTO();
        dto.setId(client.getId());
        dto.setNom(client.getNom());
        dto.setEmail(client.getEmail());

        // Count the number of credits
        int creditCount = (client.getCredits() != null) ?
```

```
client.getCredits().size() : 0;
        dto.setNombreCredits(creditCount);

        return dto;
    }

    /**
     * Convert a ClientRequestDTO to a new Client entity
     */
    public Client toEntity(ClientRequestDTO requestDTO) {
        if (requestDTO == null) {
            return null;
        }

        Client client = new Client();
        client.setNom(requestDTO.getNom());
        client.setEmail(requestDTO.getEmail());
        // Credits will be empty for a new client
        client.setCredits(Collections.emptyList());

        return client;
    }

    /**
     * Update an existing Client entity from a ClientRequestDTO
     */
    public void updateClientFromDto(ClientRequestDTO requestDTO, Client
client) {
        if (requestDTO == null || client == null) {
            return;
        }

        if (requestDTO.getNom() != null) {
            client.setNom(requestDTO.getNom());
        }
        if (requestDTO.getEmail() != null) {
            client.setEmail(requestDTO.getEmail());
        }
        // We don't update credits through this method to maintain
relationship integrity
    }

    /**
     * Convert a list of Client entities to ClientDTOs
     */
    public List<ClientDTO> toDtoList(List<Client> clients) {
        if (clients == null) {
            return Collections.emptyList();
        }

        return clients.stream()
                .map(this::toDto)
                .collect(Collectors.toList());
    }

    /**
     * Convert a list of Client entities to ClientSummaryDTOs
```

```
     */
    public List<ClientSummaryDTO> toSummaryDtoList(List<Client> clients) {
        if (clients == null) {
            return Collections.emptyList();
        }

        return clients.stream()
                .map(this::toSummaryDto)
                .collect(Collectors.toList());
    }
}
```

CreditMapper :

```
@Component
public class CreditMapper {

    private final ClientRepository clientRepository;
    private final RemboursementMapper remboursementMapper;

    @Autowired
    public CreditMapper(ClientRepository clientRepository,
RemboursementMapper remboursementMapper) {
        this.clientRepository = clientRepository;
        this.remboursementMapper = remboursementMapper;
    }

    /**
     * Convert a Credit entity to the appropriate DTO based on its type
     */
    public CreditDTO toDto(Credit credit) {
        if (credit == null) {
            return null;
        }

        // Base credit information common to all types
        CreditDTO baseDto = new CreditDTO();
        baseDto.setId(credit.getId());
        baseDto.setDateDemande(credit.getDateDemande());
        baseDto.setStatut(credit.getStatut());
        baseDto.setDateAcception(credit.getDateAcception());
        baseDto.setMontant(credit.getMontant());
        baseDto.setDureeRemboursement(credit.getDureeRemboursement());
        baseDto.setTauxInteret(credit.getTauxInteret());

        // Set client summary information
        if (credit.getClient() != null) {
            ClientSummaryDTO clientSummary = new ClientSummaryDTO();
            clientSummary.setId(credit.getClient().getId());
            clientSummary.setNom(credit.getClient().getNom());
            clientSummary.setEmail(credit.getClient().getEmail());
            clientSummary.setNombreCredits(credit.getClient().getCredits() !=
null ?
                    credit.getClient().getCredits().size() : 0);
            baseDto.setClient(clientSummary);
        }

        // Set remboursements if they exist
```

```
        if (credit.getRemboursements() != null &&
!credit.getRemboursements().isEmpty()) {
            List<RemboursementDTO> remboursementDTOs =
credit.getRemboursements().stream()
                    .map(remboursementMapper::toDto)
                    .collect(Collectors.toList());
            baseDto.setRemboursements(remboursementDTOs);
        } else {
            baseDto.setRemboursements(Collections.emptyList());
        }

        // Return the appropriate DTO based on credit type
        if (credit instanceof CreditPersonnel) {
            CreditPersonnel personnelCredit = (CreditPersonnel) credit;
            CreditPersonnelDTO dto = new CreditPersonnelDTO(baseDto,
personnelCredit.getMotif());
            return dto;
        } else if (credit instanceof CreditImmobilier) {
            CreditImmobilier immobilierCredit = (CreditImmobilier) credit;
            CreditImmobilierDTO dto = new CreditImmobilierDTO(baseDto,
immobilierCredit.getTypeBienFinance());
            return dto;
        } else if (credit instanceof CreditProfessionnel) {
            CreditProfessionnel professionnelCredit = (CreditProfessionnel)
credit;
            CreditProfessionnelDTO dto = new CreditProfessionnelDTO(baseDto,
                    professionnelCredit.getMotif(),
professionnelCredit.getRaisonSocialeEntreprise());
            return dto;
        }

        // If not a specific type, return the base DTO
        return baseDto;
    }

    /**
     * Convert a Credit entity to a summary DTO with only essential
information
     */
    public CreditSummaryDTO toSummaryDto(Credit credit) {
        if (credit == null) {
            return null;
        }

        CreditSummaryDTO summary = new CreditSummaryDTO();
        summary.setId(credit.getId());
        summary.setDateDemande(credit.getDateDemande());
        summary.setStatut(credit.getStatut());
        summary.setMontant(credit.getMontant());
        summary.setDureeRemboursement(credit.getDureeRemboursement());

        // Set type-specific fields
        if (credit instanceof CreditPersonnel) {
            CreditPersonnel personnelCredit = (CreditPersonnel) credit;
            summary.setType("PERSONNEL");
            summary.setMotif(personnelCredit.getMotif());
        } else if (credit instanceof CreditImmobilier) {
```

```
            CreditImmobilier immobilierCredit = (CreditImmobilier) credit;
            summary.setType("IMMOBILIER");

summary.setTypeBienFinance(immobilierCredit.getTypeBienFinance().toString());
        } else if (credit instanceof CreditProfessionnel) {
            CreditProfessionnel professionnelCredit = (CreditProfessionnel)
credit;
            summary.setType("PROFESSIONNEL");
            summary.setMotif(professionnelCredit.getMotif());

summary.setRaisonSocialeEntreprise(professionnelCredit.getRaisonSocialeEntrep
rise());
        }

        return summary;
    }

    /**
     * Convert a CreditRequestDTO to the appropriate Credit entity based on
its type
     */
    public Credit toEntity(CreditRequestDTO requestDTO) {
        if (requestDTO == null) {
            return null;
        }

        Credit credit;

        // Create the appropriate credit type based on the request DTO class
        if (requestDTO instanceof CreditPersonnelRequestDTO) {
            CreditPersonnelRequestDTO personnelRequest =
(CreditPersonnelRequestDTO) requestDTO;
            CreditPersonnel personnelCredit = new CreditPersonnel();
            personnelCredit.setMotif(personnelRequest.getMotif());
            credit = personnelCredit;
        } else if (requestDTO instanceof CreditImmobilierRequestDTO) {
            CreditImmobilierRequestDTO immobilierRequest =
(CreditImmobilierRequestDTO) requestDTO;
            CreditImmobilier immobilierCredit = new CreditImmobilier();

immobilierCredit.setTypeBienFinance(immobilierRequest.getTypeBienFinance());
            credit = immobilierCredit;
        } else if (requestDTO instanceof CreditProfessionnelRequestDTO) {
            CreditProfessionnelRequestDTO professionnelRequest =
(CreditProfessionnelRequestDTO) requestDTO;
            CreditProfessionnel professionnelCredit = new
CreditProfessionnel();
            professionnelCredit.setMotif(professionnelRequest.getMotif());

professionnelCredit.setRaisonSocialeEntreprise(professionnelRequest.getRaison
SocialeEntreprise());
            credit = professionnelCredit;
        } else {
            // Default to a base credit (though this should not happen with
proper validation)
            return null;
        }
```

```
        // Set common fields
        credit.setMontant(requestDTO.getMontant());
        credit.setDureeRemboursement(requestDTO.getDureeRemboursement());
        credit.setTauxInteret(requestDTO.getTauxInteret());
        credit.setDateDemande(LocalDate.now());
        credit.setStatut(StatutCredit.EN_COURS);
        credit.setRemboursements(Collections.emptyList());

        // Set client if client ID is provided
        if (requestDTO.getClientId() != null) {
            clientRepository.findById(requestDTO.getClientId())
                    .ifPresent(credit::setClient);
        }

        return credit;
    }

    /**
     * Update an existing Credit entity from a CreditRequestDTO
     */
    public void updateCreditFromDto(CreditRequestDTO requestDTO, Credit
credit) {
        if (requestDTO == null || credit == null) {
            return;
        }

        // Update common fields if they're provided
        if (requestDTO.getMontant() != null) {
            credit.setMontant(requestDTO.getMontant());
        }
        if (requestDTO.getDureeRemboursement() != null) {
            credit.setDureeRemboursement(requestDTO.getDureeRemboursement());
        }
        if (requestDTO.getTauxInteret() != null) {
            credit.setTauxInteret(requestDTO.getTauxInteret());
        }

        // Update client if ID is provided and different from current
        if (requestDTO.getClientId() != null &&
                (credit.getClient() == null ||
!credit.getClient().getId().equals(requestDTO.getClientId()))) {
            clientRepository.findById(requestDTO.getClientId())
                    .ifPresent(credit::setClient);
        }

        // Update type-specific fields
        if (requestDTO instanceof CreditPersonnelRequestDTO && credit
instanceof CreditPersonnel) {
            CreditPersonnelRequestDTO personnelRequest =
(CreditPersonnelRequestDTO) requestDTO;
            CreditPersonnel personnelCredit = (CreditPersonnel) credit;

            if (personnelRequest.getMotif() != null) {
                personnelCredit.setMotif(personnelRequest.getMotif());
            }
        } else if (requestDTO instanceof CreditImmobilierRequestDTO && credit
```

```
instanceof CreditImmobilier) {
            CreditImmobilierRequestDTO immobilierRequest =
(CreditImmobilierRequestDTO) requestDTO;
            CreditImmobilier immobilierCredit = (CreditImmobilier) credit;

            if (immobilierRequest.getTypeBienFinance() != null) {

immobilierCredit.setTypeBienFinance(immobilierRequest.getTypeBienFinance());
            }
        } else if (requestDTO instanceof CreditProfessionnelRequestDTO &&
credit instanceof CreditProfessionnel) {
            CreditProfessionnelRequestDTO professionnelRequest =
(CreditProfessionnelRequestDTO) requestDTO;
            CreditProfessionnel professionnelCredit = (CreditProfessionnel)
credit;

            if (professionnelRequest.getMotif() != null) {

professionnelCredit.setMotif(professionnelRequest.getMotif());
            }
            if (professionnelRequest.getRaisonSocialeEntreprise() != null) {

professionnelCredit.setRaisonSocialeEntreprise(professionnelRequest.getRaison
SocialeEntreprise());
            }
        }
    }

    /**
     * Convert a list of Credit entities to DTOs
     */
    public List<CreditDTO> toDtoList(List<Credit> credits) {
        if (credits == null) {
            return Collections.emptyList();
        }

        return credits.stream()
                .map(this::toDto)
                .collect(Collectors.toList());
    }

    /**
     * Convert a list of Credit entities to summary DTOs
     */
    public List<CreditSummaryDTO> toSummaryDtoList(List<Credit> credits) {
        if (credits == null) {
            return Collections.emptyList();
        }

        return credits.stream()
                .map(this::toSummaryDto)
                .collect(Collectors.toList());
    }
}
```

### e. Création des Services:

**Services principaux :**

- ClientService & ClientServiceImpl
- CreditService & CreditServiceImpl
- RemboursementService & RemboursementServiceImpl
- ReportingService & ReportingServiceImpl (pour fonctionnalités avancées)

### f. Exemple :

ClientService :

```
public interface ClientService {

    List<ClientSummaryDTO> getAllClients();

    ClientDTO getClientById(Long id);

    ClientDTO createClient(ClientRequestDTO clientRequestDTO);

    ClientDTO updateClient(Long id, ClientRequestDTO clientRequestDTO);

    void deleteClient(Long id);

    List<CreditSummaryDTO> getClientCredits(Long clientId);

    List<ClientSummaryDTO> searchClientsByName(String keyword);

    List<ClientSummaryDTO> searchClientsByEmail(String email);
}
```

ClientServiceImpl :

```
@Service
@RequiredArgsConstructor
@Transactional
public class ClientServiceImpl implements ClientService {

    private final ClientRepository clientRepository;
    private final CreditRepository creditRepository;
    private final ClientMapper clientMapper;
    private final CreditMapper creditMapper;

    @Override
    public List<ClientSummaryDTO> getAllClients() {
        List<Client> clients = clientRepository.findAll();
        return clientMapper.toSummaryDtoList(clients);
    }

    @Override
    public ClientDTO getClientById(Long id) {
        Client client = findClientOrThrow(id);
        return clientMapper.toDto(client);
    }
```

```
    @Override
    public ClientDTO createClient(ClientRequestDTO clientRequestDTO) {
        validateClientRequest(clientRequestDTO);

        // Check if email is already in use
        if
(clientRepository.findByEmail(clientRequestDTO.getEmail()).isPresent()) {
            throw new BadRequestException("Email is already in use");
        }

        Client client = clientMapper.toEntity(clientRequestDTO);
        Client savedClient = clientRepository.save(client);

        return clientMapper.toDto(savedClient);
    }

    @Override
    public ClientDTO updateClient(Long id, ClientRequestDTO clientRequestDTO)
{
        validateClientRequest(clientRequestDTO);

        Client client = findClientOrThrow(id);

        // Check if email is already in use by another client
        if (clientRepository.findByEmail(clientRequestDTO.getEmail())
                .filter(c -> !c.getId().equals(id))
                .isPresent()) {
            throw new BadRequestException("Email is already in use by another
client");
        }

        clientMapper.updateClientFromDto(clientRequestDTO, client);
        Client updatedClient = clientRepository.save(client);

        return clientMapper.toDto(updatedClient);
    }

    @Override
    public void deleteClient(Long id) {
        Client client = findClientOrThrow(id);

        // Check if client has active credits
        boolean hasActiveCredits = client.getCredits().stream()
                .anyMatch(credit -> credit.getStatut() ==
StatutCredit.EN_COURS ||
                                    credit.getStatut() ==
StatutCredit.ACCEPTE);

        if (hasActiveCredits) {
            throw new BadRequestException("Cannot delete client with active
credits");
        }

        clientRepository.delete(client);
    }
```

```
    @Override
    public List<CreditSummaryDTO> getClientCredits(Long clientId) {
        Client client = findClientOrThrow(clientId);
        List<Credit> credits = creditRepository.findByClientId(clientId);

        return creditMapper.toSummaryDtoList(credits);
    }

    @Override
    public List<ClientSummaryDTO> searchClientsByName(String keyword) {
        if (StringUtils.isBlank(keyword)) {
            return getAllClients();
        }

        List<Client> clients =
clientRepository.findByNomContainingIgnoreCase(keyword);
        return clientMapper.toSummaryDtoList(clients);
    }

    @Override
    public List<ClientSummaryDTO> searchClientsByEmail(String email) {
        if (StringUtils.isBlank(email)) {
            return getAllClients();
        }

        List<Client> clients =
clientRepository.findByEmailContainingIgnoreCase(email);
        return clientMapper.toSummaryDtoList(clients);
    }

    // Helper methods

    private Client findClientOrThrow(Long id) {
        return clientRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Client",
"id", id));
    }

    private void validateClientRequest(ClientRequestDTO requestDTO) {
        if (requestDTO == null) {
            throw new BadRequestException("Client request cannot be null");
        }

        if (StringUtils.isBlank(requestDTO.getNom())) {
            throw new BadRequestException("Client name cannot be empty");
        }

        if (StringUtils.isBlank(requestDTO.getEmail())) {
            throw new BadRequestException("Client email cannot be empty");
        }

        // Basic email validation
        if (!requestDTO.getEmail().matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-
]{2,4}$")) {
            throw new BadRequestException("Invalid email format");
        }
```

```
    }
}
```

## 4. Couche Web : REST Controllers + Swagger (OpenAPI):

La couche Web est responsable de l'exposition des services via des API RESTful, permettant à des clients (comme Angular) d'interagir avec l'application Spring Boot. Elle est construite à l'aide de Spring MVC.

Les fonctionnalités exposées sont documentées grâce à Swagger (OpenAPI), ce qui facilite le test et l'intégration des API.

### a. Contrôleurs REST créés :

Les contrôleurs suivants ont été implémentés pour chaque entité principale :

| | |
|---|---|
| ClientController.java | Gérer les clients |
| CreditController.java | Gérer les crédits (tous types) |
| RemboursementController.java | Gérer les remboursements |
| HomeController.java | Page d'accueil ou test simple |

### b. Exemple :

ClientController :

```
RestController
@RequestMapping("/api/clients")
@RequiredArgsConstructor
@Tag(name = "Client", description = "Client management API")
@CrossOrigin(origins = "*")
public class ClientController {

    private final ClientService clientService;

    @Operation(summary = "Get all clients", description = "Returns a list of
all clients with summary information")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved the list of clients",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientSummaryDTO.class)))
    })
    @GetMapping
    public ResponseEntity<List<ClientSummaryDTO>> getAllClients() {
        return ResponseEntity.ok(clientService.getAllClients());
    }

    @Operation(summary = "Get client by ID", description = "Returns detailed
information for a specific client")
    @ApiResponses(value = {
```

```
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved the client",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientDTO.class))),
            @ApiResponse(responseCode = "404", description = "Client not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<ClientDTO> getClientById(
            @Parameter(description = "Client ID", required = true)
            @PathVariable Long id) {
        return ResponseEntity.ok(clientService.getClientById(id));
    }

    @Operation(summary = "Create new client", description = "Creates a new
client and returns the created client details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "Client
successfully created",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientDTO.class))),
            @ApiResponse(responseCode = "400", description = "Invalid input
data")
    })
    @PostMapping
    public ResponseEntity<ClientDTO> createClient(
            @Parameter(description = "Client data", required = true)
            @Valid @RequestBody ClientRequestDTO clientRequestDTO) {
        return new
ResponseEntity<>(clientService.createClient(clientRequestDTO),
HttpStatus.CREATED);
    }

    @Operation(summary = "Update client", description = "Updates an existing
client and returns the updated client details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Client
successfully updated",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientDTO.class))),
            @ApiResponse(responseCode = "400", description = "Invalid input
data"),
            @ApiResponse(responseCode = "404", description = "Client not
found")
    })
    @PutMapping("/{id}")
    public ResponseEntity<ClientDTO> updateClient(
            @Parameter(description = "Client ID", required = true)
            @PathVariable Long id,
            @Parameter(description = "Updated client data", required = true)
            @Valid @RequestBody ClientRequestDTO clientRequestDTO) {
        return ResponseEntity.ok(clientService.updateClient(id,
clientRequestDTO));
    }
```

```
    @Operation(summary = "Delete client", description = "Deletes a client by
ID")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "204", description = "Client
successfully deleted"),
            @ApiResponse(responseCode = "404", description = "Client not
found"),
            @ApiResponse(responseCode = "400", description = "Client cannot
be deleted")
    })
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteClient(
            @Parameter(description = "Client ID", required = true)
            @PathVariable Long id) {
        clientService.deleteClient(id);
        return ResponseEntity.noContent().build();
    }

    @Operation(summary = "Get client credits", description = "Returns all
credits for a specific client")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved client credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class))),
            @ApiResponse(responseCode = "404", description = "Client not
found")
    })
    @GetMapping("/{id}/credits")
    public ResponseEntity<List<CreditSummaryDTO>> getClientCredits(
            @Parameter(description = "Client ID", required = true)
            @PathVariable Long id) {
        return ResponseEntity.ok(clientService.getClientCredits(id));
    }

    @Operation(summary = "Search clients by name", description = "Returns
clients matching the specified name keyword")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved matching clients",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientSummaryDTO.class)))
    })
    @GetMapping("/search/name")
    public ResponseEntity<List<ClientSummaryDTO>> searchClientsByName(
            @Parameter(description = "Name keyword", required = true)
            @RequestParam String keyword) {
        return ResponseEntity.ok(clientService.searchClientsByName(keyword));
    }

    @Operation(summary = "Search clients by email", description = "Returns
clients matching the specified email")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
```

```
retrieved matching clients",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
ClientSummaryDTO.class)))
    })
    @GetMapping("/search/email")
    public ResponseEntity<List<ClientSummaryDTO>> searchClientsByEmail(
            @Parameter(description = "Email address", required = true)
            @RequestParam String email) {
        return ResponseEntity.ok(clientService.searchClientsByEmail(email));
    }
}
```

CreditController :

```
@RestController
@RequestMapping("/api/credits")
@RequiredArgsConstructor
@Tag(name = "Credit", description = "Credit management API")
@CrossOrigin(origins = "*")
public class CreditController {

    private final CreditService creditService;

    @Operation(summary = "Get all credits", description = "Returns a list of
all credits with summary information")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved the list of credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class)))
    })
    @GetMapping
    public ResponseEntity<List<CreditSummaryDTO>> getAllCredits() {
        return ResponseEntity.ok(creditService.getAllCredits());
    }

    @Operation(summary = "Get credit by ID", description = "Returns detailed
information for a specific credit")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved the credit",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditDTO.class))),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @GetMapping("/{id}")
    public ResponseEntity<CreditDTO> getCreditById(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id) {
        return ResponseEntity.ok(creditService.getCreditById(id));
    }
```

```
    @Operation(summary = "Create new credit", description = "Creates a new
credit application and returns the created credit details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "201", description = "Credit
successfully created",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditDTO.class))),
            @ApiResponse(responseCode = "400", description = "Invalid input
data")
    })
    @PostMapping
    public ResponseEntity<CreditDTO> createCredit(
            @Parameter(description = "Credit data", required = true)
            @Valid @RequestBody CreditRequestDTO creditRequestDTO) {
        return new
ResponseEntity<>(creditService.createCredit(creditRequestDTO),
HttpStatus.CREATED);
    }

    @Operation(summary = "Update credit", description = "Updates an existing
credit and returns the updated credit details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Credit
successfully updated",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditDTO.class))),
            @ApiResponse(responseCode = "400", description = "Invalid input
data or credit cannot be updated"),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @PutMapping("/{id}")
    public ResponseEntity<CreditDTO> updateCredit(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id,
            @Parameter(description = "Updated credit data", required = true)
            @Valid @RequestBody CreditRequestDTO creditRequestDTO) {
        return ResponseEntity.ok(creditService.updateCredit(id,
creditRequestDTO));
    }

    @Operation(summary = "Delete credit", description = "Deletes a credit by
ID")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "204", description = "Credit
successfully deleted"),
            @ApiResponse(responseCode = "404", description = "Credit not
found"),
            @ApiResponse(responseCode = "400", description = "Credit cannot
be deleted")
    })
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteCredit(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id) {
```

```
        creditService.deleteCredit(id);
        return ResponseEntity.noContent().build();
    }

    @Operation(summary = "Get credits by status", description = "Returns
credits with the specified status")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved matching credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class)))
    })
    @GetMapping("/status/{status}")
    public ResponseEntity<List<CreditSummaryDTO>> getCreditsByStatus(
            @Parameter(description = "Credit status", required = true)
            @PathVariable StatutCredit status) {
        return ResponseEntity.ok(creditService.getCreditsByStatus(status));
    }

    @Operation(summary = "Get credits by type", description = "Returns
credits of the specified type")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved matching credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class))),
            @ApiResponse(responseCode = "400", description = "Invalid credit
type")
    })
    @GetMapping("/type/{type}")
    public ResponseEntity<List<CreditSummaryDTO>> getCreditsByType(
            @Parameter(description = "Credit type (PERSONNEL, IMMOBILIER,
PROFESSIONNEL)", required = true)
            @PathVariable String type) {
        return ResponseEntity.ok(creditService.getCreditsByType(type));
    }

    @Operation(summary = "Approve credit", description = "Approves a credit
application and returns the updated credit details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Credit
successfully approved",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditDTO.class))),
            @ApiResponse(responseCode = "400", description = "Credit cannot
be approved"),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @PutMapping("/{id}/approve")
    public ResponseEntity<CreditDTO> approveCredit(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id,
            @Parameter(description = "Approval date (defaults to current date
```

```
if not provided)")
            @RequestParam(required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate approvalDate) {
        LocalDate date = approvalDate != null ? approvalDate :
LocalDate.now();
        return ResponseEntity.ok(creditService.approveCredit(id, date));
    }

    @Operation(summary = "Reject credit", description = "Rejects a credit
application and returns the updated credit details")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Credit
successfully rejected",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditDTO.class))),
            @ApiResponse(responseCode = "400", description = "Credit cannot
be rejected"),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @PutMapping("/{id}/reject")
    public ResponseEntity<CreditDTO> rejectCredit(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id,
            @Parameter(description = "Rejection reason")
            @RequestParam(required = false) String reason) {
        return ResponseEntity.ok(creditService.rejectCredit(id, reason));
    }

    @Operation(summary = "Calculate monthly payment", description =
"Calculates the monthly payment for a credit")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
calculated monthly payment",
                    content = @Content(mediaType = "application/json")),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @GetMapping("/{id}/monthly-payment")
    public ResponseEntity<Map<String, Object>> calculateMonthlyPayment(
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id) {
        return ResponseEntity.ok(creditService.calculateMonthlyPayment(id));
    }

    @Operation(summary = "Get payment schedule", description = "Returns the
detailed payment schedule for a credit")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved payment schedule",
                    content = @Content(mediaType = "application/json")),
            @ApiResponse(responseCode = "404", description = "Credit not
found")
    })
    @GetMapping("/{id}/payment-schedule")
    public ResponseEntity<List<Map<String, Object>>> getPaymentSchedule(
```

```
            @Parameter(description = "Credit ID", required = true)
            @PathVariable Long id) {
        return ResponseEntity.ok(creditService.getPaymentSchedule(id));
    }

    @Operation(summary = "Validate credit application", description =
"Validates a credit application before submission")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Validation
results",
                    content = @Content(mediaType = "application/json"))
    })
    @PostMapping("/validate")
    public ResponseEntity<Map<String, Object>> validateCreditApplication(
            @Parameter(description = "Credit data to validate", required =
true)
            @Valid @RequestBody CreditRequestDTO creditRequestDTO) {
        return
ResponseEntity.ok(creditService.validateCreditApplication(creditRequestDTO));
    }

    @Operation(summary = "Search credits by amount range", description =
"Returns credits within the specified amount range")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved matching credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class)))
    })
    @GetMapping("/search/amount")
    public ResponseEntity<List<CreditSummaryDTO>> searchCreditsByAmountRange(
            @Parameter(description = "Minimum amount")
            @RequestParam(required = false) Double minAmount,
            @Parameter(description = "Maximum amount")
            @RequestParam(required = false) Double maxAmount) {
        return
ResponseEntity.ok(creditService.searchCreditsByAmountRange(minAmount,
maxAmount));
    }

    @Operation(summary = "Search credits by date range", description =
"Returns credits within the specified date range")
    @ApiResponses(value = {
            @ApiResponse(responseCode = "200", description = "Successfully
retrieved matching credits",
                    content = @Content(mediaType = "application/json",
                            schema = @Schema(implementation =
CreditSummaryDTO.class)))
    })
    @GetMapping("/search/date")
    public ResponseEntity<List<CreditSummaryDTO>> searchCreditsByDateRange(
            @Parameter(description = "Start date")
            @RequestParam(required = false) @DateTimeFormat(iso =
DateTimeFormat.ISO.DATE) LocalDate startDate,
            @Parameter(description = "End date")
            @RequestParam(required = false) @DateTimeFormat(iso =
```

```
DateTimeFormat.ISO.DATE) LocalDate endDate) {
        return
ResponseEntity.ok(creditService.searchCreditsByDateRange(startDate,
endDate));
    }
}
```

### c. Documentation Swagger:

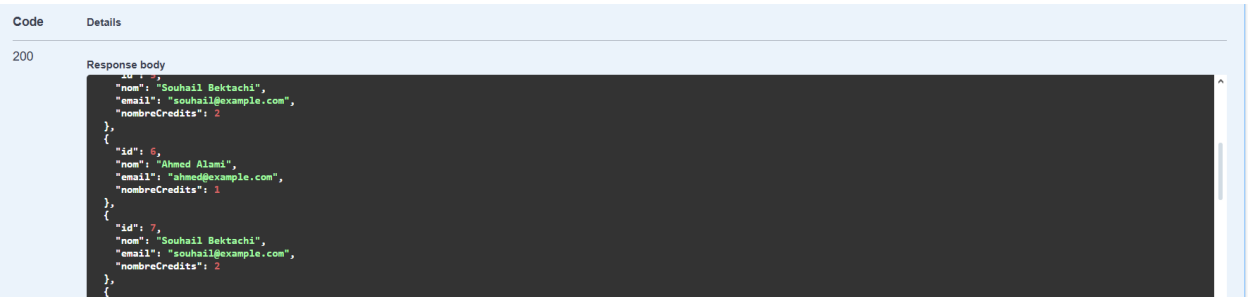
Pour documenter les API, le projet utilise **Springdoc OpenAPI** (Swagger v3).

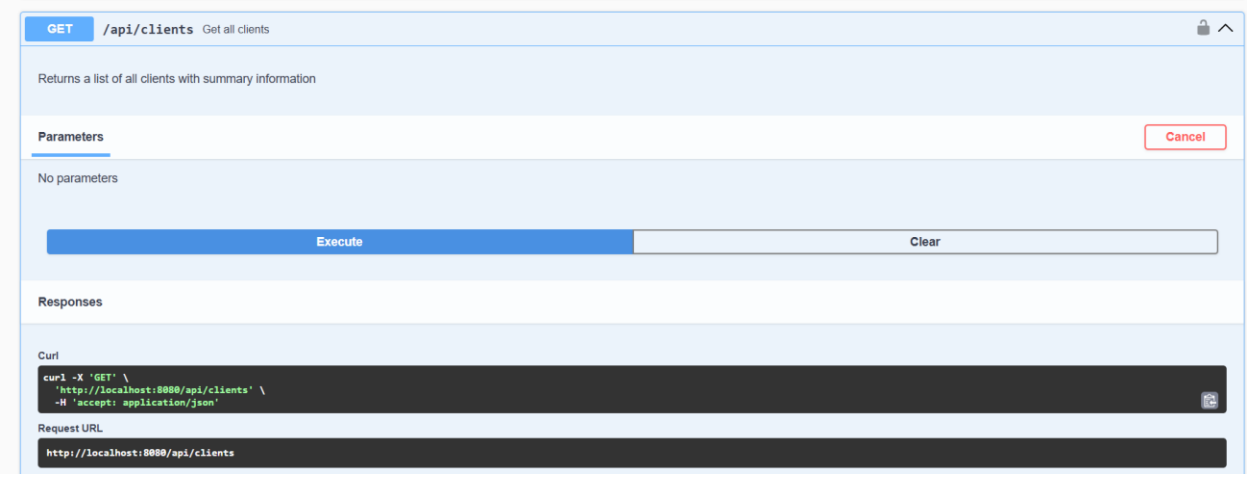
```
public class OpenApiConfig {

    @Bean
    @Primary  // Add this annotation to make this bean the primary one
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
                .info(new Info().title("Credit Management API")
                    .description("Spring Boot application for managing bank
credits")
                    .version("v1.0.0")
                    .contact(new Contact()
                        .name("Souhail Bektachi")
                        .email("souhail.bektachi@example.com"))
                    .license(new License().name("Apache
2.0").url("http://springdoc.org")))
                .externalDocs(new ExternalDocumentation()
                    .description("Credit Management Documentation")
                    .url("https://github.com/souhailbektachi"))
                .addSecurityItem(new
SecurityRequirement().addList("bearerAuth"))
                .components(new Components()
                    .addSecuritySchemes("bearerAuth",
                        new SecurityScheme()
                            .name("bearerAuth")
                            .type(SecurityScheme.Type.HTTP)
                            .scheme("bearer")
                            .bearerFormat("JWT")));
    }

    // Additional configuration for Swagger UI if needed
}
```


*Figure 3 Swagger ui*



**Remboursement** Repayment management API

| | | |
|---|---|---|
| GET | /api/remboursements/{id} | Get repayment by ID |
| PUT | /api/remboursements/{id} | Update repayment |
| DELETE | /api/remboursements/{id} | Delete repayment |
| GET | /api/remboursements | Get all repayments |
| POST | /api/remboursements | Create new repayment |
| POST | /api/remboursements/monthly-installment | Process monthly installment |
| POST | /api/remboursements/early-repayment | Process early repayment |
| GET | /api/remboursements/type/{type} | Get repayments by type |
| GET | /api/remboursements/search/date | Search repayments by date range |
| GET | /api/remboursements/search/amount | Search repayments by amount range |
| GET | /api/remboursements/remaining-balance/{creditId} | Calculate remaining balance |

**Credit** Credit management API

| | | |
|---|---|---|
| GET | /api/credits/{id} | Get credit by ID |
| PUT | /api/credits/{id} | Update credit |
| DELETE | /api/credits/{id} | Delete credit |
| PUT | /api/credits/{id}/reject | Reject credit |
| PUT | /api/credits/{id}/approve | Approve credit |
| GET | /api/credits | Get all credits |
| POST | /api/credits | Create new credit |
| POST | /api/credits/validate | Validate credit application |
| GET | /api/credits/{id}/payment-schedule | Get payment schedule |
| GET | /api/credits/{id}/monthly-payment | Calculate monthly payment |
| GET | /api/credits/type/{type} | Get credits by type |

## 5. Couche Sécurité (Spring Security + JWT):

La sécurité de l'application est assurée via Spring Security combinée avec JWT (JSON Web Token). Cela permet d'authentifier les utilisateurs et de gérer les autorisations d'accès aux différentes ressources exposées par les APIs.

### a. Gestion des utilisateurs et rôles :

entities/User.java

```
@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
```

```java
public class User implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(unique = true)
    private String username;

    private String password;

    @Column(unique = true)
    private String email;

    private String fullName;

    @Enumerated(EnumType.STRING)
    private Role role;

    private boolean enabled = true;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority(role.name()));
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

entities/Role.java :

```java
public enum Role {
    ROLE_CLIENT,
    ROLE_EMPLOYE,
    ROLE_ADMIN
}
```

## b. Authentification avec JWT:

**Login / Register**

- **Contrôleur :** AuthController.java
- **Endpoints :**
  - POST /api/auth/login : Authentification
  - POST /api/auth/register : Enregistrement utilisateur
- **Payloads :**
  - LoginRequestDTO : contient email et mot de passe
  - RegisterRequestDTO : contient nom, email, mot de passe, rôle(s)

```
•   @RestController
    @RequestMapping("/api/auth")
    @RequiredArgsConstructor
    @Tag(name = "Authentication", description = "Authentication API")
    public class AuthController {

        private final AuthenticationManager authenticationManager;
        private final UserRepository userRepository;
        private final PasswordEncoder passwordEncoder;
        private final JwtService jwtService;

        @PostMapping("/login")
        @Operation(summary = "Authenticate user", description =
    "Authenticates a user and returns a JWT token")
        public ResponseEntity<?> authenticateUser(@Valid @RequestBody
    LoginRequestDTO loginRequest) {
            try {
                Authentication authentication =
    authenticationManager.authenticate(
                        new UsernamePasswordAuthenticationToken(
                                loginRequest.getUsername(),
                                loginRequest.getPassword()
                        )
                );


    SecurityContextHolder.getContext().setAuthentication(authentication);
                User user = (User) authentication.getPrincipal();
                String jwt = jwtService.generateToken(user);

                return ResponseEntity.ok(new AuthResponseDTO(
                        jwt,
                        "Bearer",
                        user.getUsername(),
                        user.getRole().name()
                ));
            } catch (AuthenticationException e) {
                return
    ResponseEntity.status(HttpStatus.UNAUTHORIZED).body("Invalid username
```

```
or password");
        }
    }

    @PostMapping("/register")
    @Operation(summary = "Register user", description = "Registers a
new user")
    public ResponseEntity<?> registerUser(@Valid @RequestBody
RegisterRequestDTO registerRequest) {
        // Check if username is already taken
        if
(userRepository.existsByUsername(registerRequest.getUsername())) {
            return ResponseEntity.badRequest().body("Username is
already taken");
        }

        // Check if email is already in use
        if (userRepository.existsByEmail(registerRequest.getEmail())) {
            return ResponseEntity.badRequest().body("Email is already
in use");
        }

        // Create new user's account
        User user = new User();
        user.setUsername(registerRequest.getUsername());

user.setPassword(passwordEncoder.encode(registerRequest.getPassword()))
;
        user.setEmail(registerRequest.getEmail());
        user.setFullName(registerRequest.getFullName());
        user.setRole(registerRequest.getRole() != null ?
registerRequest.getRole() : Role.ROLE_CLIENT);
        user.setEnabled(true);

        userRepository.save(user);

        return ResponseEntity.ok("User registered successfully");
    }
}
```

## c. Configuration de sécurité:

**Classe principale : SecurityConfig.java**

- Configuration des endpoints publics/privés
- Intégration du filtre JWT (JwtAuthenticationFilter)
- Gestion des exceptions
- Activation du CORS, désactivation du CSRF pour REST

```
• @Configuration
  @EnableWebSecurity
  @EnableMethodSecurity
  public class SecurityConfig {

      private final JwtAuthenticationFilter jwtAuthFilter;
      private final CustomUserDetailsService customUserDetailsService;

      // Constructor with filter and dedicated user details service
      public SecurityConfig(JwtAuthenticationFilter jwtAuthFilter,
  CustomUserDetailsService customUserDetailsService) {
          this.jwtAuthFilter = jwtAuthFilter;
          this.customUserDetailsService = customUserDetailsService;
      }

      @Bean
      public SecurityFilterChain securityFilterChain(HttpSecurity http)
  throws Exception {
          http
              .csrf(AbstractHttpConfigurer::disable)
              .cors(cors ->
  cors.configurationSource(corsConfigurationSource()))
              .authorizeHttpRequests(auth -> auth
                  .requestMatchers("/api/auth/**", "/v3/api-docs/**",
  "/swagger-ui/**", "/swagger-ui.html").permitAll()
                  .requestMatchers("/api/admin/**").hasRole("ADMIN")
                  .requestMatchers("/api/employe/**").hasAnyRole("ADMIN",
  "EMPLOYE")
                  .requestMatchers("/api/client/**").hasAnyRole("ADMIN",
  "EMPLOYE", "CLIENT")
                  .anyRequest().authenticated()
              )
              .sessionManagement(session ->
  session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
              .authenticationProvider(authenticationProvider())
              .addFilterBefore(jwtAuthFilter,
  UsernamePasswordAuthenticationFilter.class);

          return http.build();
      }

      @Bean
      public CorsConfigurationSource corsConfigurationSource() {
          CorsConfiguration configuration = new CorsConfiguration();
```

```
    configuration.setAllowedOrigins(List.of("http://localhost:4200")); //
    Angular default port
            configuration.setAllowedMethods(Arrays.asList("GET", "POST",
    "PUT", "PATCH", "DELETE", "OPTIONS"));
            configuration.setAllowedHeaders(Arrays.asList("Authorization",
    "Content-Type"));
            configuration.setExposedHeaders(List.of("Authorization"));
            configuration.setAllowCredentials(true);

            UrlBasedCorsConfigurationSource source = new
    UrlBasedCorsConfigurationSource();
            source.registerCorsConfiguration("/**", configuration);
            return source;
        }

        @Bean
        public AuthenticationProvider authenticationProvider() {
            DaoAuthenticationProvider authProvider = new
    DaoAuthenticationProvider();
            authProvider.setUserDetailsService(customUserDetailsService);
            authProvider.setPasswordEncoder(passwordEncoder());
            return authProvider;
        }

        @Bean
        public PasswordEncoder passwordEncoder() {
            return new BCryptPasswordEncoder();
        }

        @Bean
        public AuthenticationManager
    authenticationManager(AuthenticationConfiguration config) throws
    Exception {
            return config.getAuthenticationManager();
        }
    }
```

### d. Services de sécurité

Les services de sécurité assurent la gestion des utilisateurs, la vérification des identifiants, la génération et la validation des tokens JWT. Voici les éléments principaux :

JwtService :

Service responsable de la génération, la validation et l'extraction des informations depuis les tokens JWT.

```
@Service
@RequiredArgsConstructor
public class JwtService {
```

```
    private final JwtConfig jwtConfig;

    public String generateToken(Authentication authentication) {
        UserDetails userDetails = (UserDetails)
authentication.getPrincipal();
        return JWT.create()
                .withSubject(userDetails.getUsername())
                .withClaim("role",
userDetails.getAuthorities().iterator().next().getAuthority())
                .withIssuedAt(new Date())
                .withExpiresAt(new Date(System.currentTimeMillis() +
jwtConfig.getTokenExpirationMs()))
                .withIssuer(jwtConfig.getIssuer())
                .sign(Algorithm.HMAC512(jwtConfig.getSecret()));
    }

    public String generateToken(User user) {
        return JWT.create()
                .withSubject(user.getUsername())
                .withClaim("role", user.getRole().name())
                .withIssuedAt(new Date())
                .withExpiresAt(new Date(System.currentTimeMillis() +
jwtConfig.getTokenExpirationMs()))
                .withIssuer(jwtConfig.getIssuer())
                .sign(Algorithm.HMAC512(jwtConfig.getSecret()));
    }

    public String getUsernameFromToken(String token) {
        DecodedJWT jwt = verifyToken(token);
        return jwt.getSubject();
    }

    public DecodedJWT verifyToken(String token) {
        try {
            return JWT.require(Algorithm.HMAC512(jwtConfig.getSecret()))
                    .withIssuer(jwtConfig.getIssuer())
                    .build()
                    .verify(token);
        } catch (JWTVerificationException exception) {
            throw new RuntimeException("Invalid JWT token", exception);
        }
    }

    public boolean validateToken(String token) {
        try {
            verifyToken(token);
            return true;
        } catch (Exception e) {
            return false;
        }
    }
}
```

JwtAuthenticationFilter:

Filtre qui intercepte chaque requête HTTP pour extraire le token JWT, le valider, puis configurer le contexte de sécurité Spring.

```
@Component
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final JwtConfig jwtConfig;
    private final CustomUserDetailsService userDetailsService; // Using our
own service to break circular dependency

    public JwtAuthenticationFilter(JwtService jwtService, JwtConfig
jwtConfig, CustomUserDetailsService userDetailsService) {
        this.jwtService = jwtService;
        this.jwtConfig = jwtConfig;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(@NonNull HttpServletRequest request,
                                    @NonNull HttpServletResponse response,
                                    @NonNull FilterChain filterChain)
            throws ServletException, IOException {
        try {
            String authHeader =
request.getHeader(jwtConfig.getHeaderString());

            if (authHeader == null ||
!authHeader.startsWith(jwtConfig.getTokenPrefix())) {
                filterChain.doFilter(request, response);
                return;
            }

            String jwt = authHeader.replace(jwtConfig.getTokenPrefix(),
"").trim();

            if (jwt.isEmpty() || !jwtService.validateToken(jwt)) {
                filterChain.doFilter(request, response);
                return;
            }

            String username = jwtService.getUsernameFromToken(jwt);

            if (username != null &&
SecurityContextHolder.getContext().getAuthentication() == null) {
                UserDetails userDetails =
userDetailsService.loadUserByUsername(username);
                UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(
                        userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
```

```
SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (Exception e) {
            log.error("Cannot set user authentication: {}", e.getMessage());
        }

        filterChain.doFilter(request, response);
    }
}
```

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/auth/login' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "username": "admin",
  "password": "admin123"
}'
```

Request URL

http://localhost:8080/api/auth/login

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
{
  "token": "eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbiIsInJvbGUiOiJST0xFX0FETU1OIiwiaWF0IjoxNzQ3NjQ4NjQyLCJleHAiOjE3NDc3MzUwNDIsImlzcyI6ImNyZWRpdC1tYW5hZ2VtZW50LWFwcCJ9.GtH5ISwaEP8xMUL_GgsrliXhH0Nw984qEOeDef7qUPyaIbto0JR0oJ1l_oMcQifyrI9Cnx61QB2RzKtcZCgi1A",
  "tokenType": "Bearer",
  "username": "admin",
  "role": "ROLE_ADMIN"
}
```


Download

Response headers

```
cache-control: no-cache,no-store,max-age=0,must-revalidate
connection: keep-alive
content-type: application/json
date: Mon,19 May 2025 09:57:22 GMT
expires: 0
keep-alive: timeout=60
pragma: no-cache
transfer-encoding: chunked
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
x-content-type-options: nosniff
x-frame-options: DENY
x-xss-protection: 0
```

*Figure 4 Login swagger impl*

```
curl -X 'GET' \
  'http://localhost:8080/api/credits' \
  -H 'accept: application/json' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbiIsInJvbGUiOiJST0xFX0FETU1OIiwiaWF0IjoxNzQ3NjQ4NjQyLCJleHAiOjE3NDc3MzUwNDIsImlzcyI6ImNyZWRpdC1tYW5hZ2VtZW50LWFwcCJ9.GtH5ISwaEP
```

Request URL

http://localhost:8080/api/credits

Server response

| Code | Details |
|---|---|
| 200 | Response body |


```
[
  {
    "id": 1,
    "dateDemande": "2025-02-19",
    "statut": "ACCEPTE",
    "montant": 50000,
    "dureeRemboursement": 36,
    "type": "PERSONNEL",
    "motif": "Achat de voiture",
    "typeBienFinance": null,
    "raisonSocialeEntreprise": null
  },
  {
    "id": 2,
    "dateDemande": "2024-11-19",
    "statut": "ACCEPTE",
    "montant": 800000,
    "dureeRemboursement": 240,
    "type": "IMMOBILIER",
    "motif": null,
    "typeBienFinance": "APPARTEMENT",
    "raisonSocialeEntreprise": null
  },
  {
    "id": 3,
    "dateDemande": "2025-01-19",
    "statut": "EN_COURS",
```


Download

*Figure 5 Request with token*

# Front end:

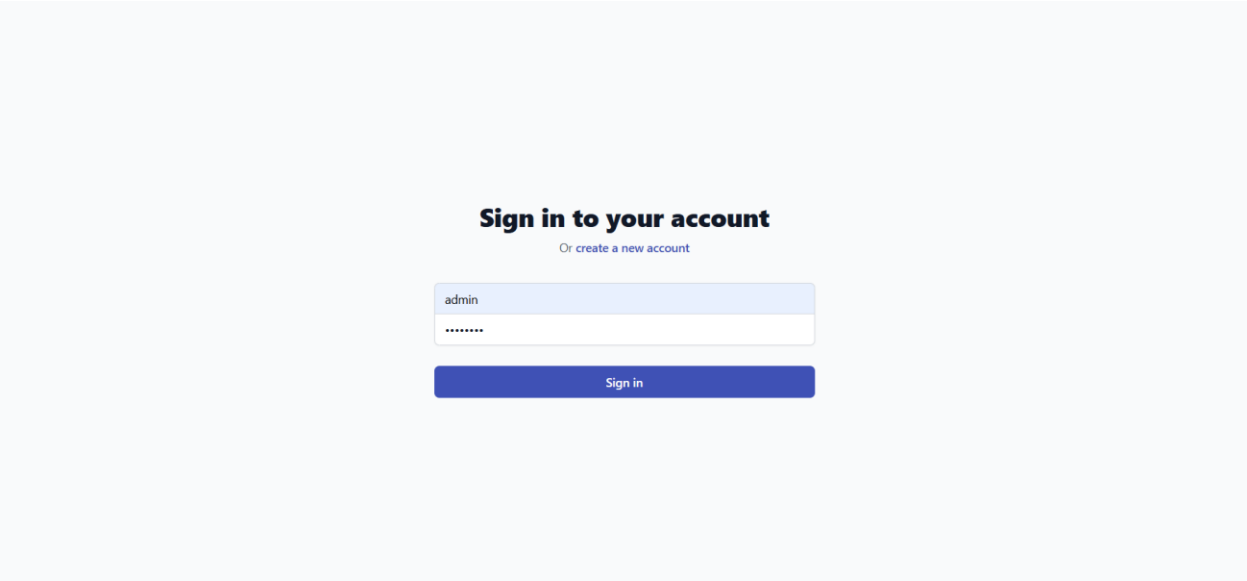



*Figure 6 login page*

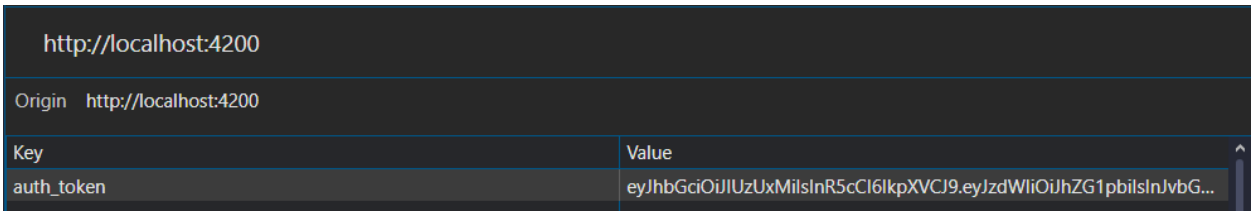
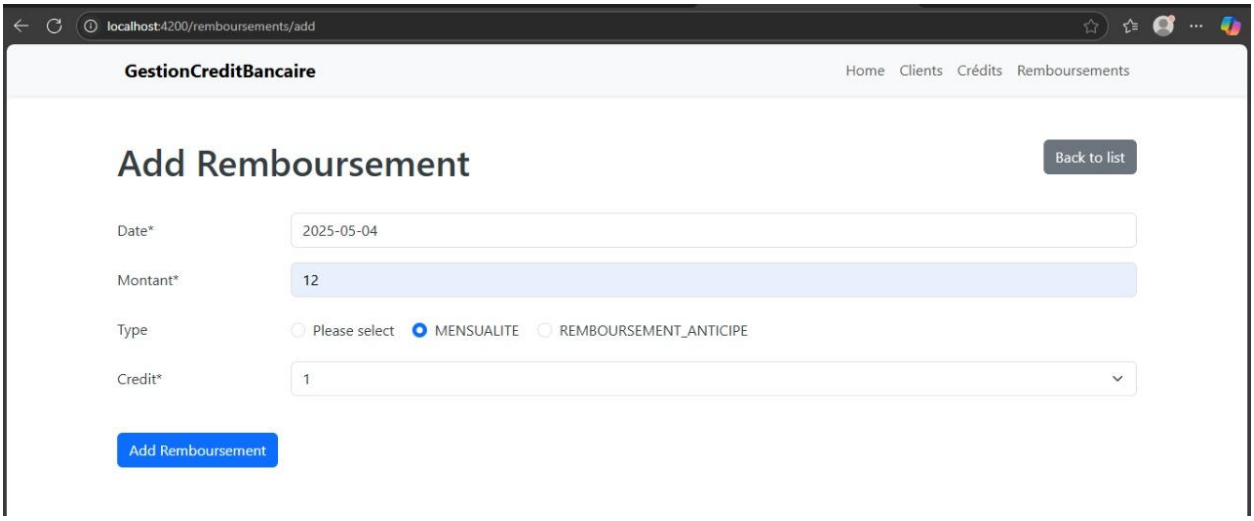
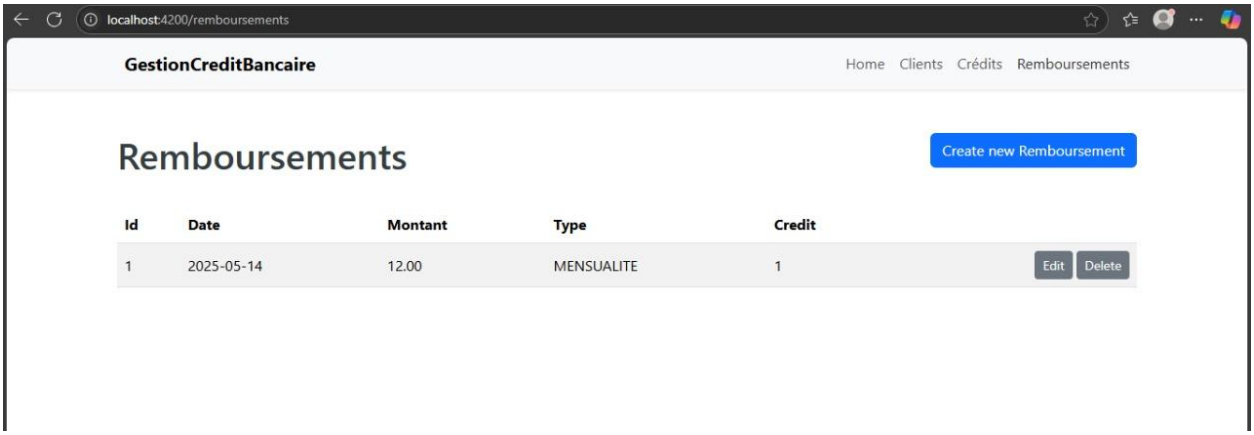
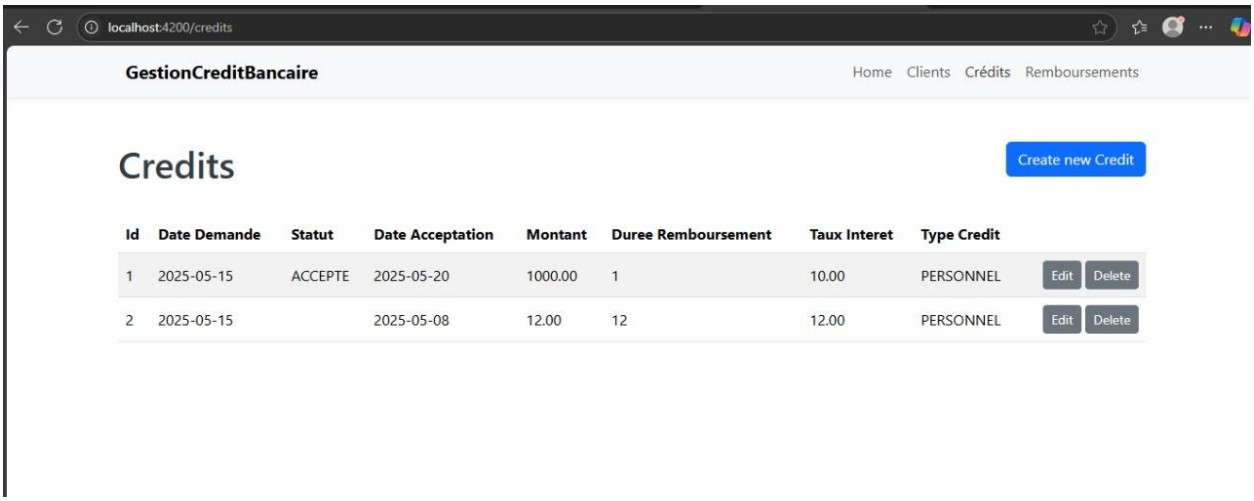



*Figure 7 token*


localhost:4200/remboursements/add
GestionCreditBancaire
Home Clients Crédits Remboursements
Add Remboursement
Back to list
Date*
2025-05-04
Montant*
12
Type
Please select
MENSUALITE
REMBOURSEMENT_ANTICIPE
Credit*
1
Add Remboursement


localhost:4200/remboursements
GestionCreditBancaire
Home Clients Crédits Remboursements
Remboursements
Create new Remboursement
Id Date Montant Type Credit
1 2025-05-14 12.00 MENSUALITE 1
Edit Delete


localhost:4200/credits
GestionCreditBancaire
Home Clients Crédits Remboursements
Credits
Create new Credit
Id Date Demande Statut Date Acceptation Montant Duree Remboursement Taux Interet Type Credit
1 2025-05-15 ACCEPTE 2025-05-20 1000.00 1 10.00 PERSONNEL
Edit Delete
2 2025-05-15 2025-05-08 12.00 12 12.00 PERSONNEL
Edit Delete