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

Examen JEE et Middleware

Réalisé par :

EL MOUTAOUAKIL Abdellah

II-BDCC2

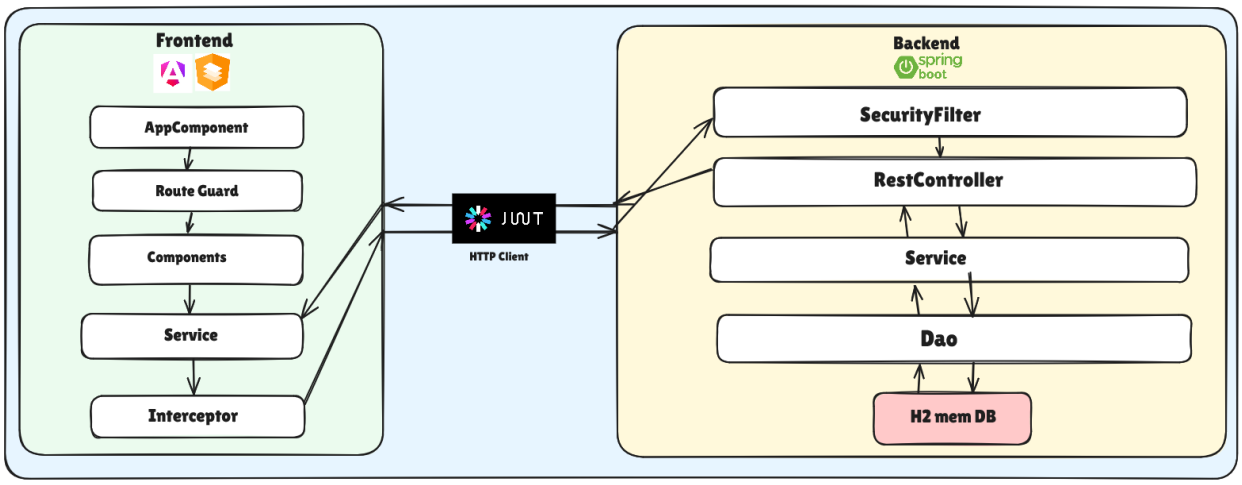
Année universitaire :

2024-2025

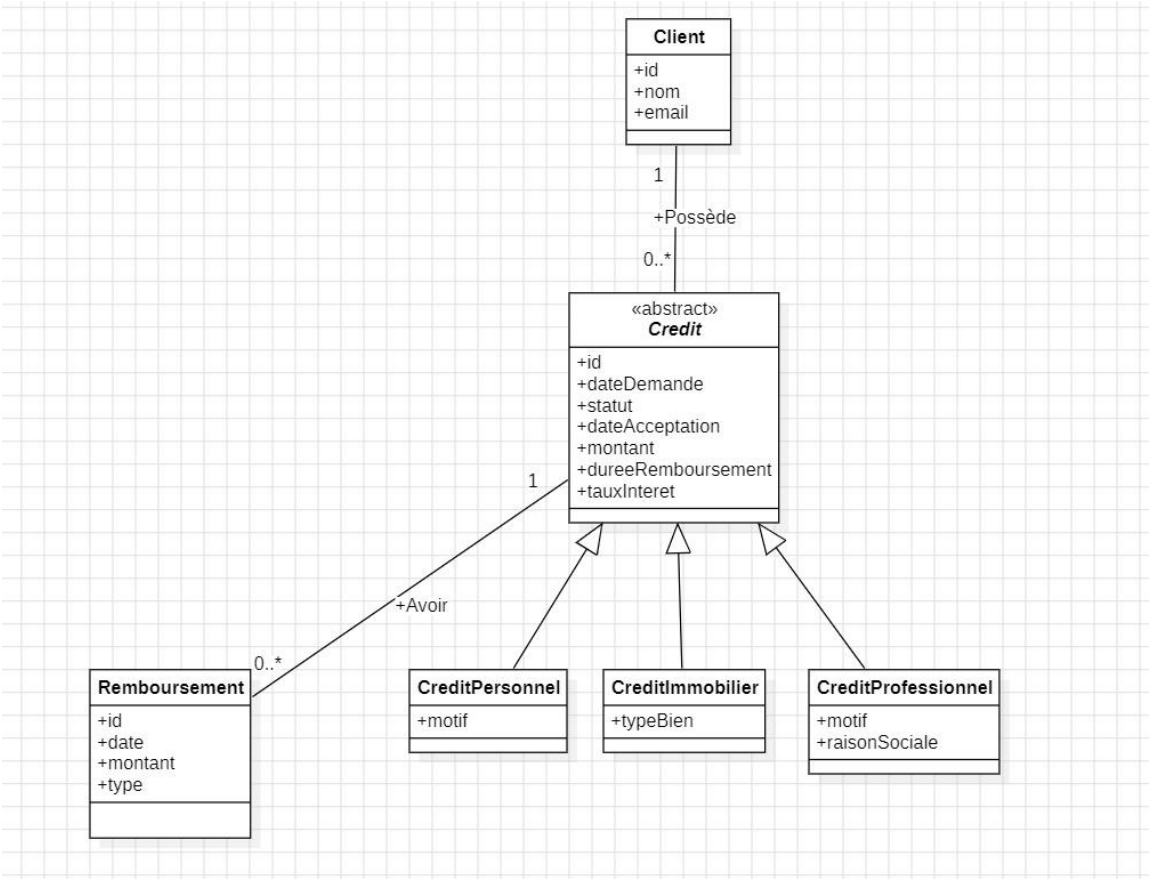
Table de matière

1.Conception

1. Architecture technique du projet



2. Diagramme de classe



1.Implémentation

1. Couche DAO

1. Créer les entités JPA

Client :

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.Id;
import lombok.*;

@Entity @Getter @Setter @AllArgsConstructor @NoArgsConstructor @Builder
public class Client {
    @Id
    @GeneratedValue(strategy = jakarta.persistence.GenerationType.IDENTITY)
    private Long id;
    private String nom;
    private String email;
}
```

Credit :

```
package ma.abdellahelmoutaouakil.backend.entities;

import jakarta.persistence.*;
import lombok.*;
import java.util.Date;
import java.util.List;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "type_credit")
public abstract class Credit {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date dateDemande;
    private String statut;
    private Date dateAcceptation;
    private double montant;
    private int dureeRemboursement;
    private double tauxInteret;

    @ManyToOne
    private Client client;

    @OneToMany(mappedBy = "credit")
    private List<Remboursement> remboursements;
}
```

Creditimmobilier :

```
package ma.abdellahelmoutaouakil.backend.entities;

import jakarta.persistence.DiscriminatorValue;
import jakarta.persistence.Entity;
import lombok.*;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
@Entity
@DiscriminatorValue("IMMOBILIER")
public class CreditImmobilier extends Credit {
    private String typeBien;
}
```

Creditpersonnel :

```
package ma.abdellahelmoutaouakil.backend.entities;

import jakarta.persistence.DiscriminatorValue;
import jakarta.persistence.Entity;
import lombok.*;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
@Entity
@DiscriminatorValue("PERSONNEL")
public class CreditPersonnel extends Credit {
    private String motif;
}
```

CreditProfessionnel :

```
package ma.abdellahelmoutaouakil.backend.entities;

import jakarta.persistence.DiscriminatorValue;
import jakarta.persistence.Entity;
import lombok.*;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
@Entity
@DiscriminatorValue("PROFESSIONNEL")
public class CreditProfessionnel extends Credit {
    private String motif;
    private String raisonSociale;
}
```

Remboursement :

```
package ma.abdellahelmoutaouakil.backend.entities;

import jakarta.persistence.*;
import lombok.*;
import java.util.Date;

@Getter @Setter @NoArgsConstructor @AllArgsConstructor @Builder
```

```java
@Entity
public class Remboursement {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date date;
    private double montant;
    private String type;

    @ManyToOne
    private Credit credit;
}
```

les enums :

```java
package ma.abdellahelmoutaouakil.backend.enums;

public enum StatutCredit {
    EN_COURS,
    ACCEPTE,
    REJETE
}
```

```java
package ma.abdellahelmoutaouakil.backend.enums;

public enum TypeBien {
    APPARTEMENT,
    MAISON,
    LOCAL_COMMERCIAL
}
```

```java
package ma.abdellahelmoutaouakil.backend.enums;

public enum TypeRemboursement {
    MENSUALITE,
    REMBOURSEMENT_ANTICIPE
}
```

2. Créer les interfaces JPA Repository basées sur Spring Data

ClientRepository :

```java
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.Client;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ClientRepository extends JpaRepository<Client, Long> {
}
```

cREDITiMMOBILIERrEPOSITORY :

```
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.CreditImmobilier;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CreditImmobilierRepository extends
JpaRepository<CreditImmobilier, Long> {
}
```

CreditPersonnelRepository :

```
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.CreditPersonnel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CreditPersonnelRepository extends
JpaRepository<CreditPersonnel, Long> {
}
```

CreditProfessionnelRepository :

```
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.CreditProfessionnel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CreditProfessionnelRepository extends
JpaRepository<CreditProfessionnel, Long> {
}
```

CreditRepository :

```
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.CreditProfessionnel;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CreditProfessionnelRepository extends
JpaRepository<CreditProfessionnel, Long> {
}
```

RemboursementRepository :

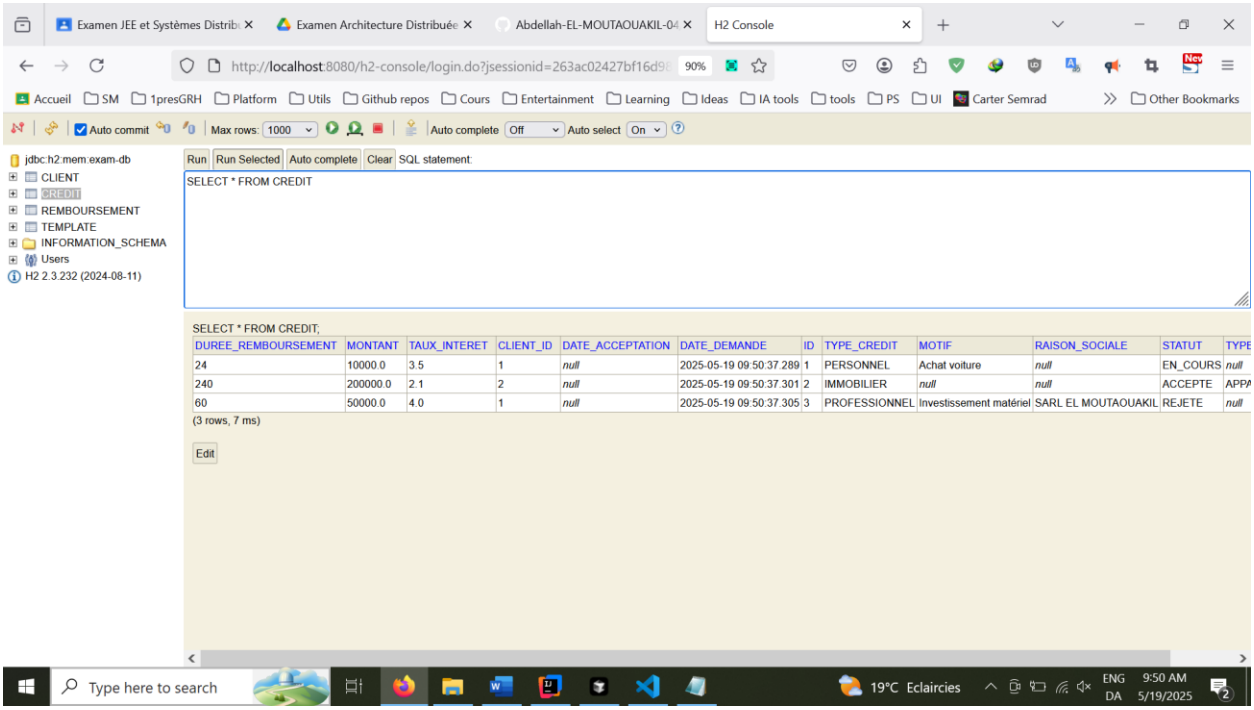
```
package ma.abdellahelmoutaouakil.backend.repositories;

import ma.abdellahelmoutaouakil.backend.entities.Remboursement;
import org.springframework.data.jpa.repository.JpaRepository;

public interface RemboursementRepository extends JpaRepository<Remboursement,
Long> {
}
```

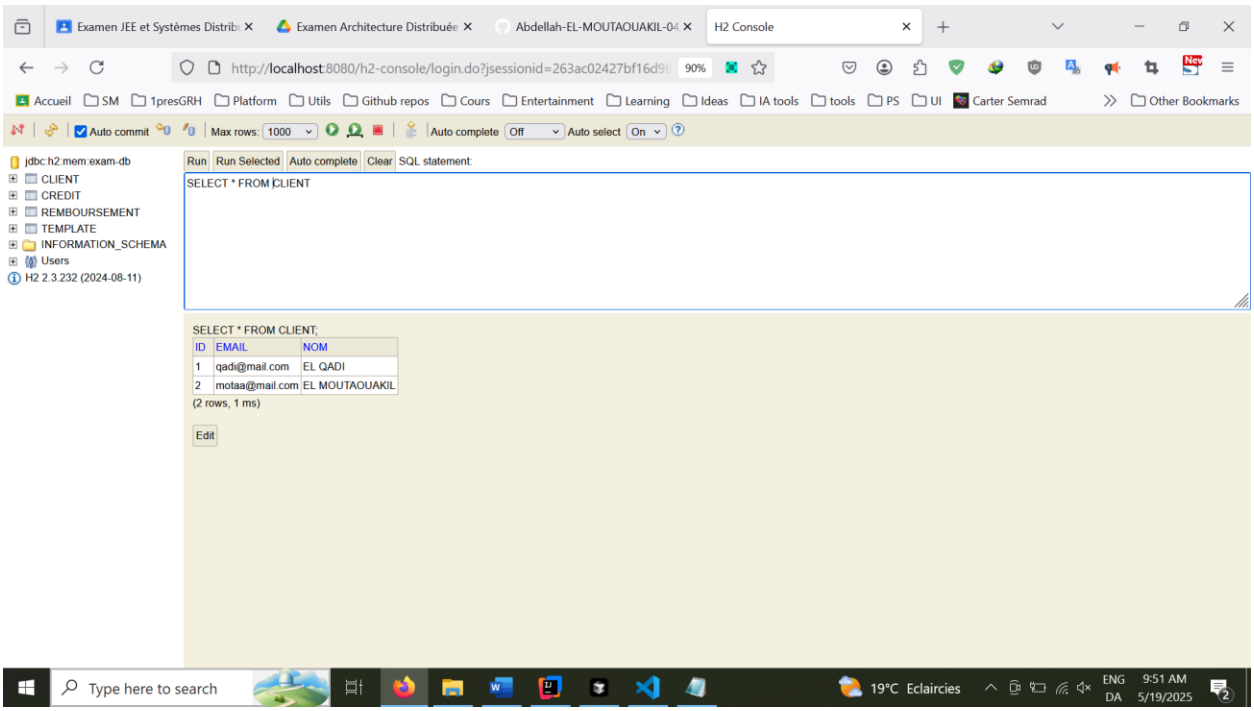
3. Tester la couche DAO avec une application qui alimente la base de données avec quelques enregistrements de test.

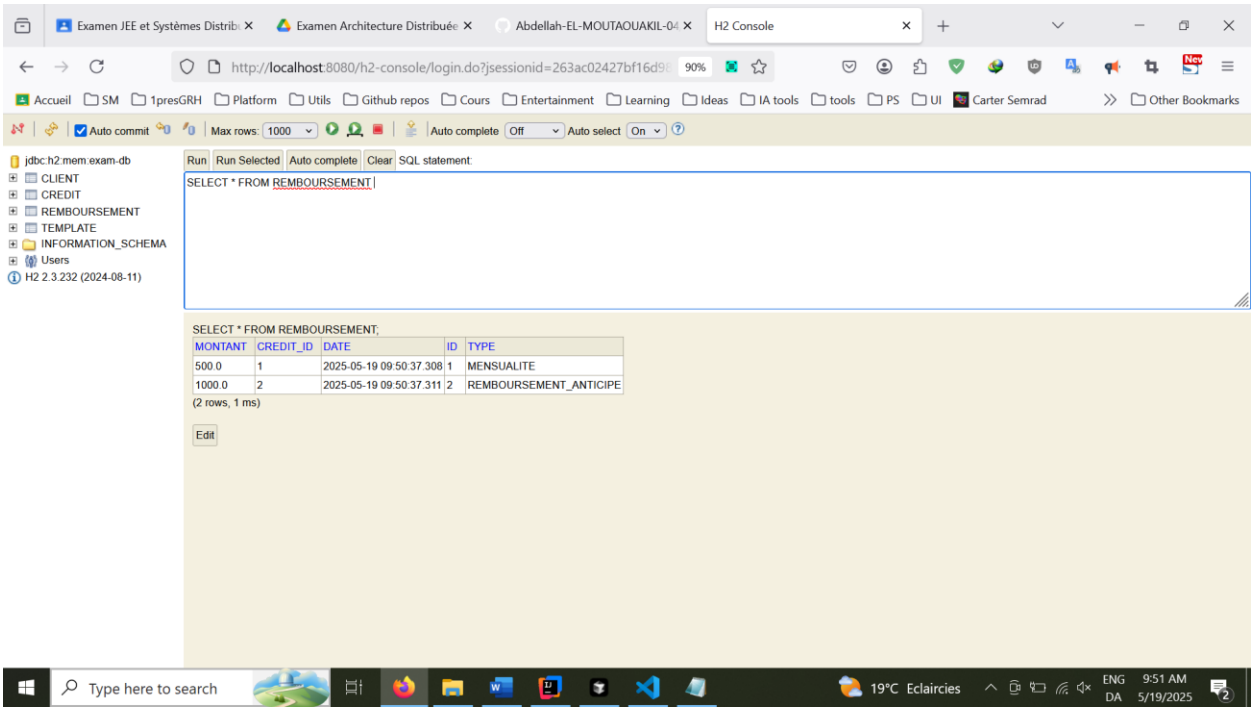
```
@Bean
CommandLineRunner start(ClientRepository clientRepo,
                        CreditRepository creditRepo,
                        CreditPersonnelRepository creditPersonnelRepo,
                        CreditImmobilierRepository creditImmobilierRepo,
                        CreditProfessionnelRepository
creditProfessionnelRepo,
                        RemboursementRepository remboursementRepo) {
    return args -> {
        Client client1 = clientRepo.save(Client.builder().nom("EL
QADI").email("qadi@mail.com").build());
        Client client2 = clientRepo.save(Client.builder().nom("EL
MOUTAOUAKIL").email("motaa@mail.com").build());

        CreditPersonnel cp = CreditPersonnel.builder()
                .dateDemande(new Date())
                .statut(StatutCredit.EN_COURS)
                .montant(10000)
                .dureeRemboursement(24)
                .tauxInteret(3.5)
                .motif("Achat voiture")
                .client(client1)
                .build();
        creditPersonnelRepo.save(cp);

        CreditImmobilier ci = CreditImmobilier.builder()
                .dateDemande(new Date())
                .statut(StatutCredit.ACCEPTE)
                .montant(200000)
                .dureeRemboursement(240)
                .tauxInteret(2.1)
                .typeBien(TypeBien.APPARTEMENT)
                .client(client2)
                .build();
        creditImmobilierRepo.save(ci);

        CreditProfessionnel cpro = CreditProfessionnel.builder()
                .dateDemande(new Date())
                .statut(StatutCredit.REJETE)
                .montant(50000)
                .dureeRemboursement(60)
                .tauxInteret(4.0)
                .motif("Investissement matériel")
                .raisonSociale("SARL EL MOUTAOUAKIL")
                .client(client1)
                .build();
        creditProfessionnelRepo.save(cpro);

        remboursementRepo.save(Remboursement.builder()
                .date(new Date())
                .montant(500)
                .type(TypeRemboursement.MENSUALITE)
                .credit(cp)
```

```
                .build());
        remboursementRepo.save(Remboursement.builder()
                .date(new Date())
                .montant(1000)
                .type(TypeRemboursement.REMBOURSEMENT_ANTICIPE)
                .credit(ci)
                .build());
    };
}
```

Les donnes sont bien ajouter a la base de donnes :




Examen JEE et Systèmes Distribu
Examen Architecture Distribuée
Abdellah-EL-MOUTAOUAKIL-04
H2 Console
http://localhost:8080/h2-console/login.do?jsessionid=263ac02427bf16d9
Accueil SM 1presGRH Platform Utils Github repos Cours Entertainment Learning Ideas IA tools tools PS UI Carter Semrad Other Bookmarks
Auto commit Max rows: 1000 Auto complete Off Auto select On
jdbc:h2:mem:exam-db
CLIENT
CREDIT
REMBOURSEMENT
TEMPLATE
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run Run Selected Auto complete Clear SQL statement:
SELECT * FROM CLIENT
SELECT * FROM CLIENT;
ID EMAIL NOM
1 qadi@mail.com EL QADI
2 motaa@mail.com EL MOUTAOUAKIL
(2 rows, 1 ms)
Edit
Type here to search
19°C Eclaircies
ENG DA
9:51 AM 5/19/2025


Examen JEE et Systèmes Distribu
Examen Architecture Distribuée
Abdellah-EL-MOUTAOUAKIL-04
H2 Console
http://localhost:8080/h2-console/login.do?jsessionid=263ac02427bf16d9
Accueil SM 1presGRH Platform Utils Github repos Cours Entertainment Learning Ideas IA tools tools PS UI Carter Semrad Other Bookmarks
Auto commit Max rows: 1000 Auto complete Off Auto select On
jdbc:h2:mem:exam-db
CLIENT
CREDIT
REMBOURSEMENT
TEMPLATE
INFORMATION_SCHEMA
Users
H2 2.3.232 (2024-08-11)
Run Run Selected Auto complete Clear SQL statement:
SELECT * FROM REMBOURSEMENT
SELECT * FROM REMBOURSEMENT;
MONTANT CREDIT_ID DATE ID TYPE
500.0 1 2025-05-19 09:50:37.308 1 MENSUALITE
1000.0 2 2025-05-19 09:50:37.311 2 REMBOURSEMENT_ANTICIPE
(2 rows, 1 ms)
Edit
Type here to search
19°C Eclaircies
ENG DA
9:51 AM 5/19/2025

2. Créer une couche service en créant les DTOs et les Mappers, en proposant les fonctionnalités que vous voyez importantes

ClientDTO :

```
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;

@Data
public class ClientDTO {
    private Long id;
    private String nom;
    private String email;
}
```

CreditDTO :

```
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;
import java.util.Date;
import ma.abdellahelmoutaouakil.backend.enums.StatutCredit;

@Data
public class CreditDTO {
    private Long id;
    private Date dateDemande;
    private StatutCredit statut;
    private Date dateAcceptation;
    private double montant;
    private int dureeRemboursement;
    private double tauxInteret;
    private Long clientId;
}
```

CreditImmobilierDTO :

```
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;
import lombok.EqualsAndHashCode;
import ma.abdellahelmoutaouakil.backend.enums.TypeBien;

@Data
@EqualsAndHashCode(callSuper = true)
public class CreditImmobilierDTO extends CreditDTO {
    private TypeBien typeBien;
}
```

CreditPersonnelDTO :

```
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;
import lombok.EqualsAndHashCode;
```

```java
@Data
@EqualsAndHashCode(callSuper = true)
public class CreditPersonnelDTO extends CreditDTO {
    private String motif;
}
```

CreditProfessionnelDTO :

```java
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;
import lombok.EqualsAndHashCode;

@Data
@EqualsAndHashCode(callSuper = true)
public class CreditProfessionnelDTO extends CreditDTO {
    private String motif;
    private String raisonSociale;
}
```

RemboursementDTO :

```java
package ma.abdellahelmoutaouakil.backend.dtos;

import lombok.Data;
import java.util.Date;
import ma.abdellahelmoutaouakil.backend.enums.TypeRemboursement;

@Data
public class RemboursementDTO {
    private Long id;
    private Date date;
    private double montant;
    private TypeRemboursement type;
    private Long creditId;
}
```

La creation de mapper:

```java
package ma.abdellahelmoutaouakil.backend.mappers;

import ma.abdellahelmoutaouakil.backend.dtos.*;
import ma.abdellahelmoutaouakil.backend.entities.*;
import org.springframework.beans.BeanUtils;
import org.springframework.stereotype.Service;

@Service
public class BankMapperImpl {
    // Client
    public ClientDTO fromClient(Client client) {
        ClientDTO dto = new ClientDTO();
        BeanUtils.copyProperties(client, dto);
        return dto;
    }
    public Client fromClientDTO(ClientDTO dto) {
        Client client = new Client();
        BeanUtils.copyProperties(dto, client);
        return client;
    }
```

```
    // Credit (abstrait)
    public CreditDTO fromCredit(Credit credit) {
        CreditDTO dto = new CreditDTO();
        BeanUtils.copyProperties(credit, dto);
        if (credit.getClient() != null)
dto.setClientId(credit.getClient().getId());
        return dto;
    }
    public Credit fromCreditDTO(CreditDTO dto) {
        Credit credit = new CreditPersonnel(); // Par défaut, à spécialiser
selon le type réel
        BeanUtils.copyProperties(dto, credit);
        return credit;
    }

    // CreditPersonnel
    public CreditPersonnelDTO fromCreditPersonnel(CreditPersonnel credit) {
        CreditPersonnelDTO dto = new CreditPersonnelDTO();
        BeanUtils.copyProperties(credit, dto);
        if (credit.getClient() != null)
dto.setClientId(credit.getClient().getId());
        return dto;
    }
    public CreditPersonnel fromCreditPersonnelDTO(CreditPersonnelDTO dto) {
        CreditPersonnel credit = new CreditPersonnel();
        BeanUtils.copyProperties(dto, credit);
        return credit;
    }

    // CreditImmobilier
    public CreditImmobilierDTO fromCreditImmobilier(CreditImmobilier credit)
{
        CreditImmobilierDTO dto = new CreditImmobilierDTO();
        BeanUtils.copyProperties(credit, dto);
        if (credit.getClient() != null)
dto.setClientId(credit.getClient().getId());
        return dto;
    }
    public CreditImmobilier fromCreditImmobilierDTO(CreditImmobilierDTO dto)
{
        CreditImmobilier credit = new CreditImmobilier();
        BeanUtils.copyProperties(dto, credit);
        return credit;
    }

    // CreditProfessionnel
    public CreditProfessionnelDTO fromCreditProfessionnel(CreditProfessionnel
credit) {
        CreditProfessionnelDTO dto = new CreditProfessionnelDTO();
        BeanUtils.copyProperties(credit, dto);
        if (credit.getClient() != null)
dto.setClientId(credit.getClient().getId());
        return dto;
    }
    public CreditProfessionnel
fromCreditProfessionnelDTO(CreditProfessionnelDTO dto) {
```

```
        CreditProfessionnel credit = new CreditProfessionnel();
        BeanUtils.copyProperties(dto, credit);
        return credit;
    }

    // Remboursement
    public RemboursementDTO fromRemboursement(Remboursement remboursement) {
        RemboursementDTO dto = new RemboursementDTO();
        BeanUtils.copyProperties(remboursement, dto);
        if (remboursement.getCredit() != null)
dto.setCreditId(remboursement.getCredit().getId());
        return dto;
    }
    public Remboursement fromRemboursementDTO(RemboursementDTO dto) {
        Remboursement remboursement = new Remboursement();
        BeanUtils.copyProperties(dto, remboursement);
        return remboursement;
    }
}
```

La creation des services :

Services Client :

```
package ma.abdellahelmoutaouakil.backend.services;

import ma.abdellahelmoutaouakil.backend.dtos.ClientDTO;
import java.util.List;

public interface ClientService {
    ClientDTO saveClient(ClientDTO clientDTO);
    ClientDTO getClient(Long id);
    List<ClientDTO> listClients();
    void deleteClient(Long id);
}
```

```
package ma.abdellahelmoutaouakil.backend.services;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.ClientDTO;
import ma.abdellahelmoutaouakil.backend.entities.Client;
import ma.abdellahelmoutaouakil.backend.mappers.BankMapperImpl;
import ma.abdellahelmoutaouakil.backend.repositories.ClientRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
@RequiredArgsConstructor
public class ClientServiceImpl implements ClientService {
    private final ClientRepository clientRepository;
    private final BankMapperImpl mapper;
```

```
    @Override
    public ClientDTO saveClient(ClientDTO clientDTO) {
        Client client = mapper.fromClientDTO(clientDTO);
        return mapper.fromClient(clientRepository.save(client));
    }

    @Override
    public ClientDTO getClient(Long id) {
        return clientRepository.findById(id)
                .map(mapper::fromClient)
                .orElse(null);
    }

    @Override
    public List<ClientDTO> listClients() {
        return clientRepository.findAll().stream()
                .map(mapper::fromClient)
                .collect(Collectors.toList());
    }

    @Override
    public void deleteClient(Long id) {
        clientRepository.deleteById(id);
    }
}
```

Credit service :

```
package ma.abdellahelmoutaouakil.backend.services;

import ma.abdellahelmoutaouakil.backend.dtos.*;
import java.util.List;

public interface CreditService {
    CreditDTO saveCredit(CreditDTO creditDTO);
    CreditDTO getCredit(Long id);
    List<CreditDTO> listCredits();
    void deleteCredit(Long id);
    List<CreditDTO> findCreditsByClient(Long clientId);

    CreditPersonnelDTO saveCreditPersonnel(CreditPersonnelDTO dto);
    CreditImmobilierDTO saveCreditImmobilier(CreditImmobilierDTO dto);
    CreditProfessionnelDTO saveCreditProfessionnel(CreditProfessionnelDTO
dto);
}
```

```
package ma.abdellahelmoutaouakil.backend.services;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.*;
import ma.abdellahelmoutaouakil.backend.entities.*;
import ma.abdellahelmoutaouakil.backend.mappers.BankMapperImpl;
import ma.abdellahelmoutaouakil.backend.repositories.*;
import org.springframework.stereotype.Service;
```

```java
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
@RequiredArgsConstructor
public class CreditServiceImpl implements CreditService {
    private final CreditRepository creditRepository;
    private final CreditPersonnelRepository creditPersonnelRepository;
    private final CreditImmobilierRepository creditImmobilierRepository;
    private final CreditProfessionnelRepository
creditProfessionnelRepository;
    private final ClientRepository clientRepository;
    private final BankMapperImpl mapper;

    @Override
    public CreditDTO saveCredit(CreditDTO creditDTO) {
        Credit credit = mapper.fromCreditDTO(creditDTO);
        if (creditDTO.getClientId() != null) {

clientRepository.findById(creditDTO.getClientId()).ifPresent(credit::setClien
t);
        }
        return mapper.fromCredit(creditRepository.save(credit));
    }

    @Override
    public CreditDTO getCredit(Long id) {
        return creditRepository.findById(id)
                .map(mapper::fromCredit)
                .orElse(null);
    }

    @Override
    public List<CreditDTO> listCredits() {
        return creditRepository.findAll().stream()
                .map(mapper::fromCredit)
                .collect(Collectors.toList());
    }

    @Override
    public void deleteCredit(Long id) {
        creditRepository.deleteById(id);
    }

    @Override
    public List<CreditDTO> findCreditsByClient(Long clientId) {
        return creditRepository.findAll().stream()
                .filter(c -> c.getClient() != null &&
c.getClient().getId().equals(clientId))
                .map(mapper::fromCredit)
                .collect(Collectors.toList());
    }

    @Override
```

```
    public CreditPersonnelDTO saveCreditPersonnel(CreditPersonnelDTO dto) {
        CreditPersonnel credit = mapper.fromCreditPersonnelDTO(dto);
        if (dto.getClientId() != null) {

clientRepository.findById(dto.getClientId()).ifPresent(credit::setClient);
        }
        return
mapper.fromCreditPersonnel(creditPersonnelRepository.save(credit));
    }

    @Override
    public CreditImmobilierDTO saveCreditImmobilier(CreditImmobilierDTO dto)
{
        CreditImmobilier credit = mapper.fromCreditImmobilierDTO(dto);
        if (dto.getClientId() != null) {

clientRepository.findById(dto.getClientId()).ifPresent(credit::setClient);
        }
        return
mapper.fromCreditImmobilier(creditImmobilierRepository.save(credit));
    }

    @Override
    public CreditProfessionnelDTO
saveCreditProfessionnel(CreditProfessionnelDTO dto) {
        CreditProfessionnel credit = mapper.fromCreditProfessionnelDTO(dto);
        if (dto.getClientId() != null) {

clientRepository.findById(dto.getClientId()).ifPresent(credit::setClient);
        }
        return
mapper.fromCreditProfessionnel(creditProfessionnelRepository.save(credit));
    }
}
```

Remboursement service :

```
package ma.abdellahelmoutaouakil.backend.services;

import ma.abdellahelmoutaouakil.backend.dtos.RemboursementDTO;
import java.util.List;

public interface RemboursementService {
    RemboursementDTO saveRemboursement(RemboursementDTO dto);
    RemboursementDTO getRemboursement(Long id);
    List<RemboursementDTO> listRemboursements();
    void deleteRemboursement(Long id);
    List<RemboursementDTO> findRemboursementsByCredit(Long creditId);
}
```

```
package ma.abdellahelmoutaouakil.backend.services;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.RemboursementDTO;
import ma.abdellahelmoutaouakil.backend.entities.Credit;
```

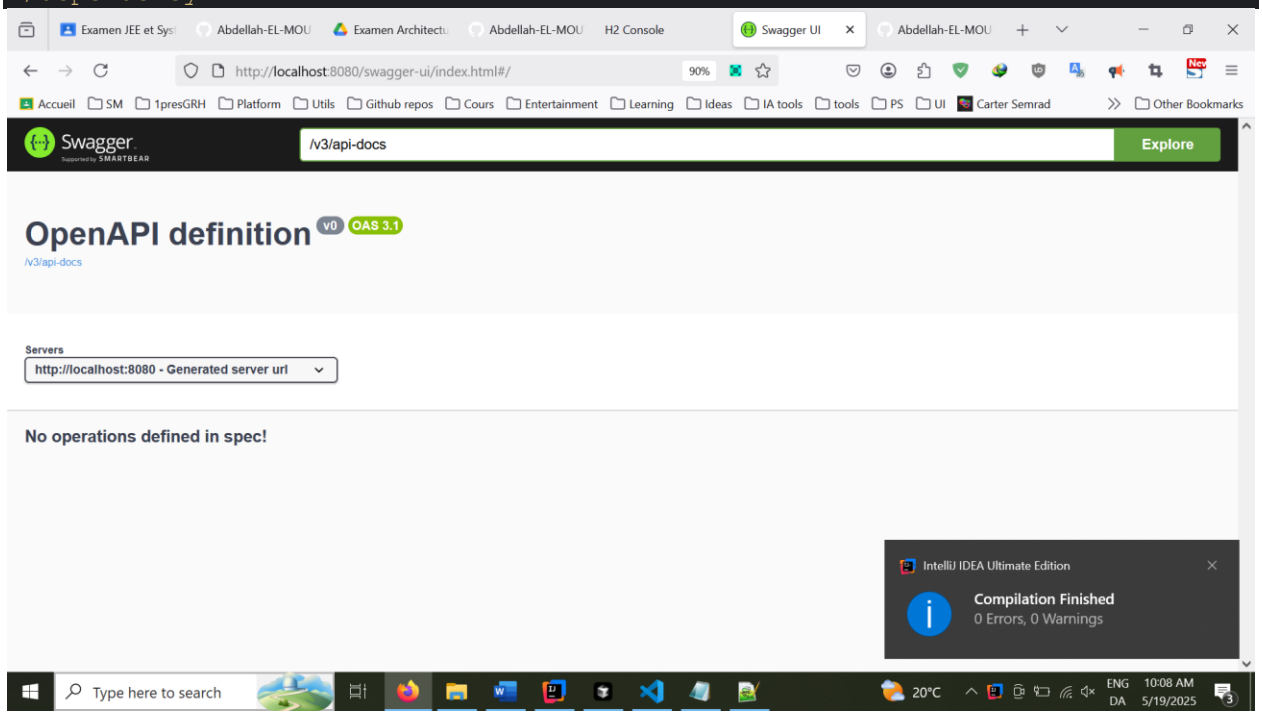
```java
import ma.abdellahelmoutaouakil.backend.entities.Remboursement;
import ma.abdellahelmoutaouakil.backend.mappers.BankMapperImpl;
import ma.abdellahelmoutaouakil.backend.repositories.CreditRepository;
import ma.abdellahelmoutaouakil.backend.repositories.RemboursementRepository;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional
@RequiredArgsConstructor
public class RemboursementServiceImpl implements RemboursementService {
    private final RemboursementRepository remboursementRepository;
    private final CreditRepository creditRepository;
    private final BankMapperImpl mapper;

    @Override
    public RemboursementDTO saveRemboursement(RemboursementDTO dto) {
        Remboursement remboursement = mapper.fromRemboursementDTO(dto);
        if (dto.getCreditId() != null) {

creditRepository.findById(dto.getCreditId()).ifPresent(remboursement::setCred
it);
        }
        return
mapper.fromRemboursement(remboursementRepository.save(remboursement));
    }

    @Override
    public RemboursementDTO getRemboursement(Long id) {
        return remboursementRepository.findById(id)
                .map(mapper::fromRemboursement)
                .orElse(null);
    }

    @Override
    public List<RemboursementDTO> listRemboursements() {
        return remboursementRepository.findAll().stream()
                .map(mapper::fromRemboursement)
                .collect(Collectors.toList());
    }

    @Override
    public void deleteRemboursement(Long id) {
        remboursementRepository.deleteById(id);
    }

    @Override
    public List<RemboursementDTO> findRemboursementsByCredit(Long creditId) {
        return remboursementRepository.findAll().stream()
                .filter(r -> r.getCredit() != null &&
r.getCredit().getId().equals(creditId))
                .map(mapper::fromRemboursement)
                .collect(Collectors.toList());
```

```
    }
}
```

2. Créer les Web services (Rest Controllers) en proposant les fonctionnalités que vous estimez importantes. Tester les REST API en générant la documentations SWAGGER (Open API Doc)

En ajoute maintenant la dependance de swagger ui :

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
    <version>2.8.6</version>
</dependency>
```



En créer maintenant les rest controllers :

ClientController :

```
package ma.abdellahelmoutaouakil.backend.web;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.ClientDTO;
import ma.abdellahelmoutaouakil.backend.services.ClientService;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```java
@RestController
@RequestMapping("/clients")
@RequiredArgsConstructor
public class ClientController {
    private final ClientService clientService;

    @GetMapping
    public List<ClientDTO> allClients() {
        return clientService.listClients();
    }

    @GetMapping("/{id}")
    public ClientDTO getClient(@PathVariable Long id) {
        return clientService.getClient(id);
    }

    @PostMapping
    public ClientDTO createClient(@RequestBody ClientDTO dto) {
        return clientService.saveClient(dto);
    }

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

CreditController :

```java
package ma.abdellahelmoutaouakil.backend.web;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.*;
import ma.abdellahelmoutaouakil.backend.services.CreditService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/credits")
@RequiredArgsConstructor
public class CreditController {
    private final CreditService creditService;

    @GetMapping
    public List<CreditDTO> allCredits() {
        return creditService.listCredits();
    }

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

    @GetMapping("/by-client/{clientId}")
    public List<CreditDTO> creditsByClient(@PathVariable Long clientId) {
```

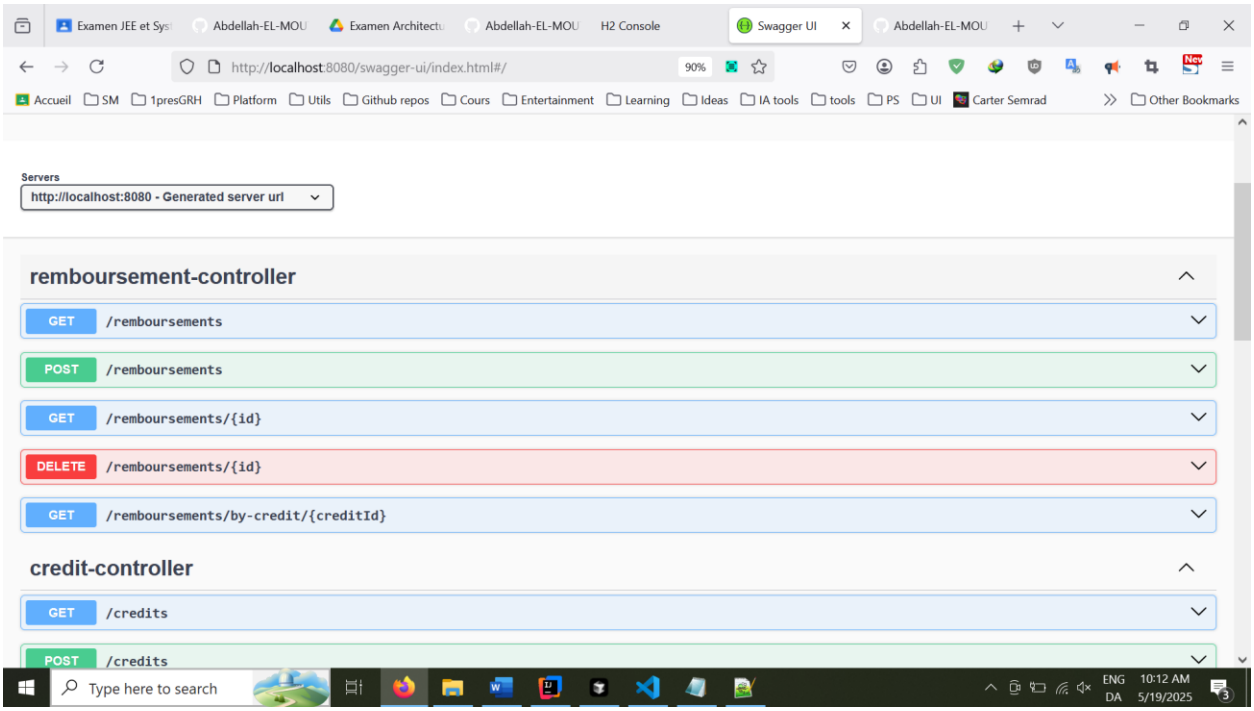
```
        return creditService.findCreditsByClient(clientId);
    }

    @PostMapping
    public CreditDTO createCredit(@RequestBody CreditDTO dto) {
        return creditService.saveCredit(dto);
    }

    @PostMapping("/personnel")
    public CreditPersonnelDTO createCreditPersonnel(@RequestBody
CreditPersonnelDTO dto) {
        return creditService.saveCreditPersonnel(dto);
    }

    @PostMapping("/immobilier")
    public CreditImmobilierDTO createCreditImmobilier(@RequestBody
CreditImmobilierDTO dto) {
        return creditService.saveCreditImmobilier(dto);
    }

    @PostMapping("/professionnel")
    public CreditProfessionnelDTO createCreditProfessionnel(@RequestBody
CreditProfessionnelDTO dto) {
        return creditService.saveCreditProfessionnel(dto);
    }

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

RemboursementController :

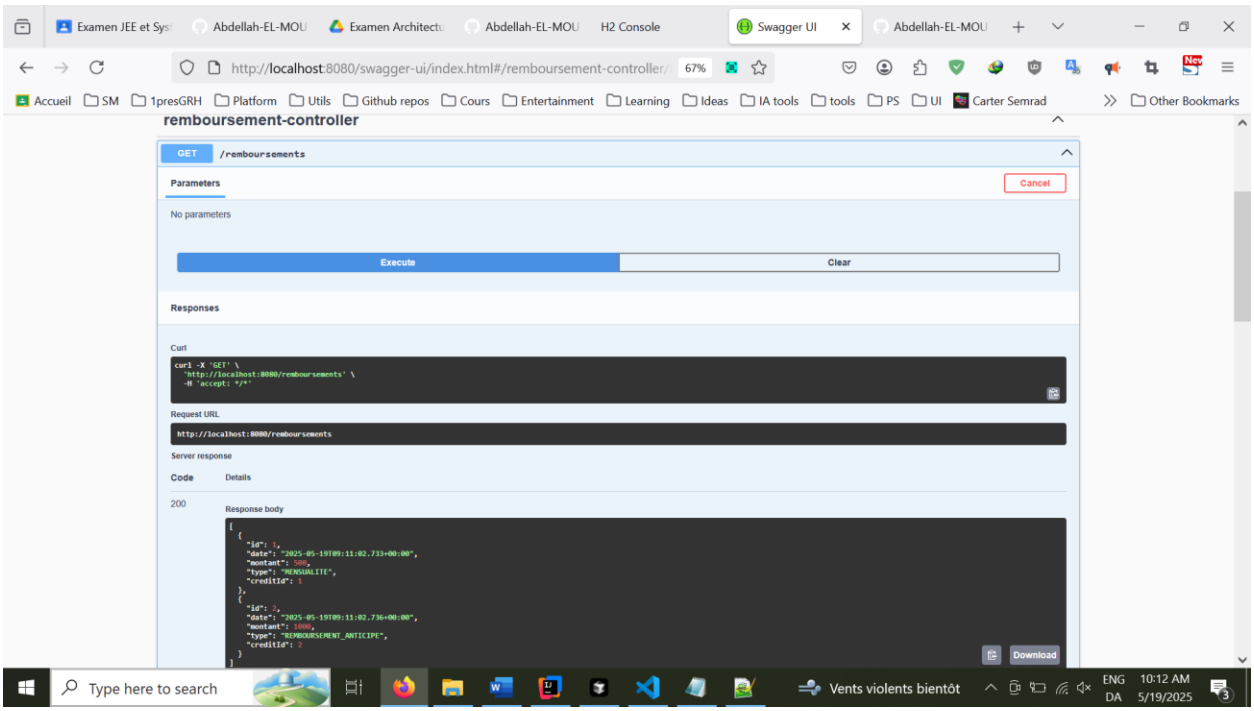
```
package ma.abdellahelmoutaouakil.backend.web;

import lombok.RequiredArgsConstructor;
import ma.abdellahelmoutaouakil.backend.dtos.RemboursementDTO;
import ma.abdellahelmoutaouakil.backend.services.RemboursementService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/remboursements")
@RequiredArgsConstructor
public class RemboursementController {
    private final RemboursementService remboursementService;

    @GetMapping
    public List<RemboursementDTO> allRemboursements() {
        return remboursementService.listRemboursements();
    }

    @GetMapping("/{id}")
    public RemboursementDTO getRemboursement(@PathVariable Long id) {
```

```
        return remboursementService.getRemboursement(id);
    }

    @GetMapping("/by-credit/{creditId}")
    public List<RemboursementDTO> remboursementsByCredit(@PathVariable Long
creditId) {
        return remboursementService.findRemboursementsByCredit(creditId);
    }

    @PostMapping
    public RemboursementDTO createRemboursement(@RequestBody RemboursementDTO
dto) {
        return remboursementService.saveRemboursement(dto);
    }

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

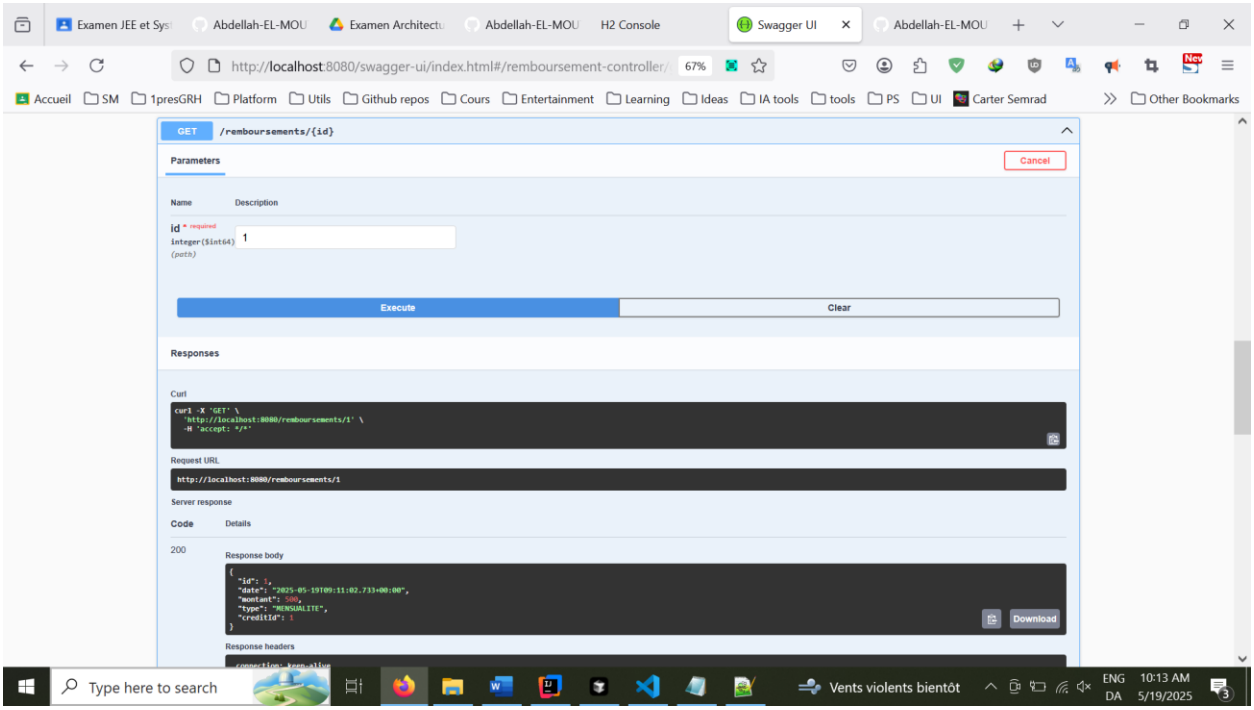
En verifier les endpoints sur swagger-ui :



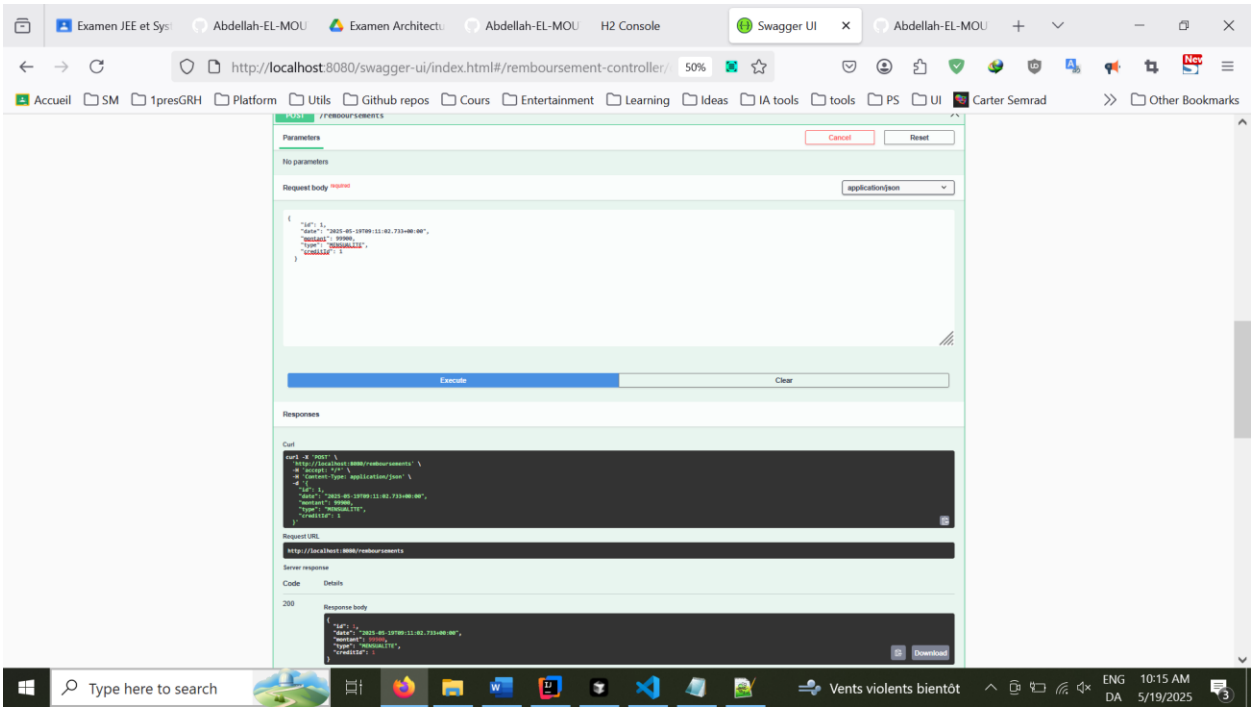
En verifier les remboursements :



Le remboursement avec l'id 1 :

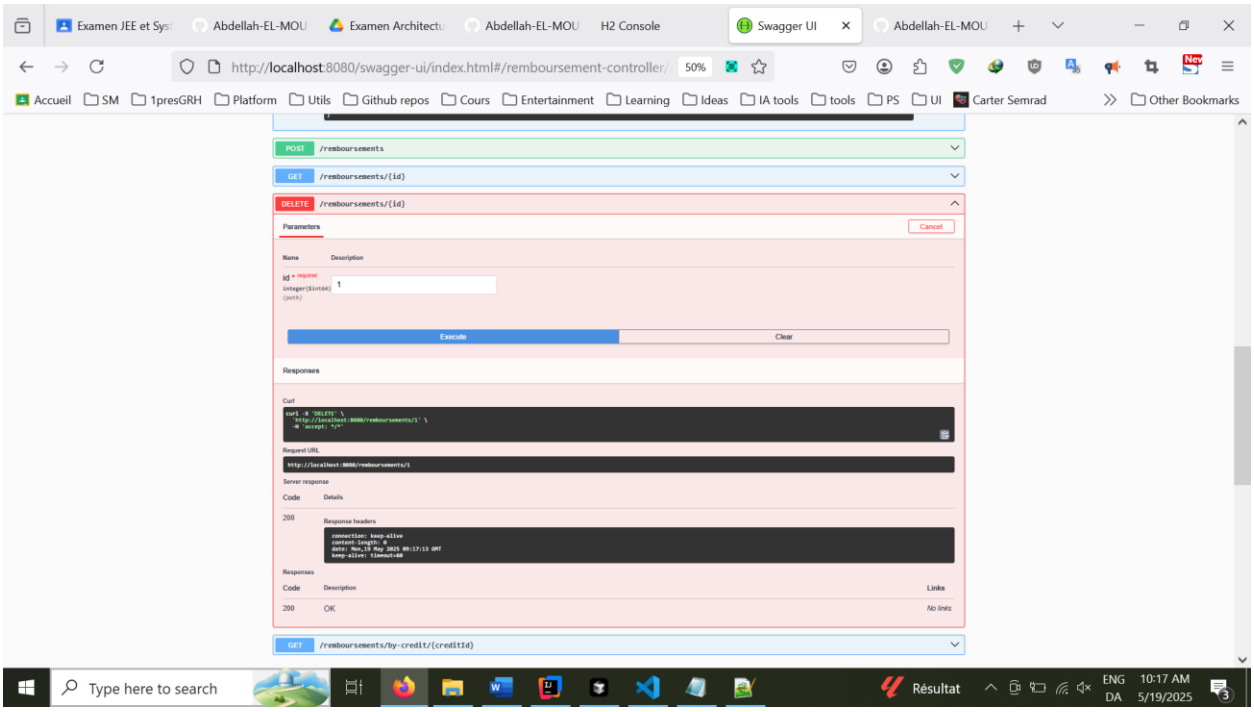


En modifie un remboursement :



Le remboursement est bien modifie

En supprime le reboursement avec l'id 1:



Comme vous vouez le code http indique quelle est bien supprimer.

Pour les autres si la meme chose si pour cela j'ai fait seulement l'exmple pour remboursement.

3.Security de backend

En ajoute premierement la dependance de spring security :

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

En créer maintenant le security config :

```
package ma.abdellahelmoutaouakil.backend.security;

import com.nimbusds.jose.jwk.source.ImmutableJWKSet;
import com.nimbusds.jose.jwk.source.ImmutableSecret;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.ProviderManager;
import
org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.method.configuration.EnableMet
hodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSec
urity;
import
org.springframework.security.config.annotation.web.configurers.oauth2.server.
resource.OAuth2ResourceServerConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.oauth2.jose.jws.MacAlgorithm;
import org.springframework.security.oauth2.jwt.JwtDecoder;
import org.springframework.security.oauth2.jwt.JwtEncoder;
import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;
import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import javax.crypto.spec.SecretKeySpec;
import java.util.List;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
```

```
public class SecurityConfig {
    @Value("${jwt.secret}")
    private String secretKey;
    @Bean
public InMemoryUserDetailsManager inMemoryUserDetailsManager() {
    PasswordEncoder passwordEncoder=passwordEncoder();
    return new InMemoryUserDetailsManager(
            User.withUsername("clt1")
                    .password(passwordEncoder.encode("1234"))
                    .roles("CLIENT")
                    .build(),
            User.withUsername("emp1")
                    .password(passwordEncoder.encode("1234"))
                    .roles("EMPLOYE")
                    .build(),
            User.withUsername("admin")
                    .password(passwordEncoder.encode("1234"))
                    .roles("ADMIN","EMPLOYE")
                    .build()
    );
}

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

    @Bean
    JwtEncoder jwtEncoder() {
        //give me a scret key with 64 caracteres
        //String
secretKey="12345678901234567890123456789012345678901234567890123456789012345678901234";
        return new NimbusJwtEncoder(new
ImmutableSecret<>(secretKey.getBytes()));
    }
    @Bean
    public JwtDecoder jwtDecoder() {
        //String
secretKey="12345678901234567890123456789012345678901234567890123456789012345678901234";
        SecretKeySpec secretKeySpec=new
SecretKeySpec(secretKey.getBytes(),"RSA");
        return
NimbusJwtDecoder.withSecretKey(secretKeySpec).macAlgorithm(MacAlgorithm.HS512
).build();
    }

    @Bean
    public AuthenticationManager authenticationManager(UserDetailsService
userDetailsService) {
        DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        authenticationProvider.setPasswordEncoder(passwordEncoder());
        authenticationProvider.setUserDetailsService(userDetailsService);
        return new ProviderManager(authenticationProvider);
    }
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity)
```

```
throws Exception {
        return httpSecurity
                .sessionManagement(sm-
>sm.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .csrf(cs->cs.disable())
                .cors(Customizer.withDefaults())
                .authorizeHttpRequests(ar->
ar.requestMatchers("/auth/login/**","/v3/api-docs/**","/swagger-ui/**","/h2-
console/**").permitAll())
                .authorizeHttpRequests(ar-> ar.anyRequest().authenticated())
                //.httpBasic(Customizer.withDefaults())
                .oauth2ResourceServer(oa->oa.jwt(Customizer.withDefaults()))
                .build();
    }
    @Bean
    CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration corsConfiguration = new CorsConfiguration();
        corsConfiguration.addAllowedOrigin("*");
        corsConfiguration.addAllowedHeader("*");
        corsConfiguration.addAllowedMethod("*");
        //corsConfiguration.setExposedHeaders(List.of("x-auth-token"));
        UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", corsConfiguration);
        return source;
    }
}
```

J'ai utiliser la strategie inMemoryUserDetailsManger pour créer les utilisateur et j'ai créer trois un client avec le role CLIENT et un employe avec le role EMPLOYE et un admin avec les roles EMPLOYE ET ADMIN

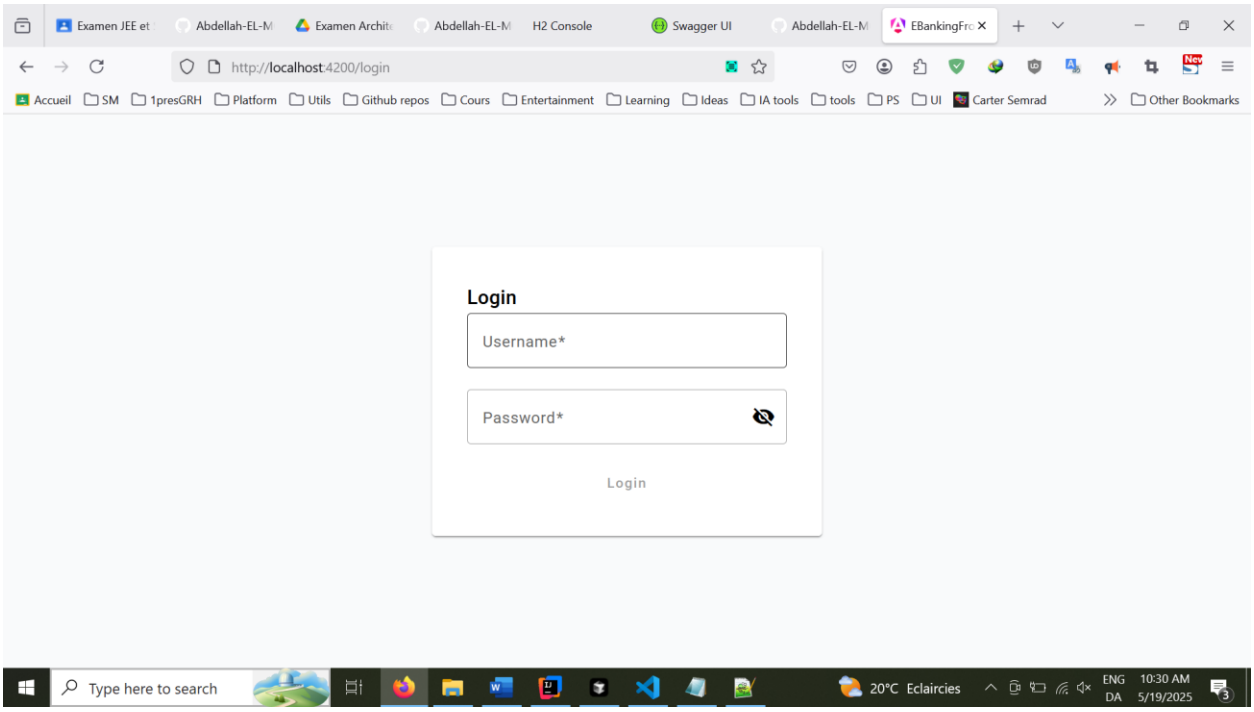
en cree le securityController :

```
package ma.abdellahelmoutaouakil.backend.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationTok
en;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.oauth2.jose.jws.MacAlgorithm;
import org.springframework.security.oauth2.jwt.*;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Map;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/auth")
public class SecurityController {
```

```
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JwtEncoder jwtEncoder;
    @GetMapping("/profile")
    public Authentication authentication(Authentication authentication) {
        return authentication;
    }
    @PostMapping("/login")
    public Map<String,String> login(String username,String password){
        Authentication authentication= authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(username,password)
        );
        //create a jwt token
        Instant instant=Instant.now();
        String
scope=authentication.getAuthorities().stream().map(GrantedAuthority::getAutho
rity).collect(Collectors.joining(" "));
        JwtClaimsSet jwtClaimsSet=JwtClaimsSet.builder()
                .issuer("http://localhost:8085")
                .issuedAt(instant)
                .expiresAt(instant.plus(10, ChronoUnit.MINUTES))
                .subject(username)
                .claim("scope",scope)
                .build();
        JwtEncoderParameters jwtEncoderParameters=JwtEncoderParameters.from(
                JwsHeader.with(MacAlgorithm.HS512)
                        .build(),
                jwtClaimsSet
        );
        String jwt=jwtEncoder.encode(jwtEncoderParameters).getTokenValue();
        return Map.of("access-token",jwt);
    }
}
```

3.FrontEnd

La page de login :



En cree leur service pour recpurer le jwt de backend :

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpHeaders, HttpParams} from '@angular/common/http';
import {jwtDecode} from 'jwt-decode';
import {Router} from '@angular/router';
import {environment} from '../../environments/environment';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  isAuthenticated:boolean=false;
  roles:any;
  username:any;
  accessToken!:any;

  constructor(private http:HttpClient,private router:Router) { }

  public login(username: string, password: string){
    let options = {
      headers:new HttpHeaders({}).set("Content-type", "application/x-www-
form-urlencoded")
    };
    let params=new HttpParams().set('username', username).set('password',
password);
    return this.http.post(environment.backendHost+"/auth/login", params,
```

```
options);

  }

  public logout(){
    this.isAuthenticated=false;
    this.roles=undefined;
    this.username=undefined;
    this.accessToken=undefined;
    window.localStorage.removeItem('jwt-token');
    this.router.navigateByUrl('/login').then();
  }

  loadProfile(data: any) {
    this.accessToken = data['access-token'];

    console.log('Access token received:', this.accessToken);

    if (typeof this.accessToken === 'string' && this.accessToken.trim() !==
'') {
      this.isAuthenticated = true;
      const decodedJwt = jwtDecode(this.accessToken) as any;
      this.username = decodedJwt.sub;
      this.roles = decodedJwt.scope;
      window.localStorage.setItem('jwt-token', this.accessToken);
    } else {
      console.error('Invalid access token received:', this.accessToken);
      this.isAuthenticated = false;
    }
  }
  loadJwtTokenFromLocalStorage(){
    let token=window.localStorage.getItem('jwt-token');
    if(token){
      this.loadProfile({ 'access-token': token });
      console.log("Token loaded from local storage:", token);
      //this.router.navigateByUrl('/admin/customers').then();
    }
  }
}
```

maintenant en cree l'intercepteur pour a chaque fois en envoi une requete au backend en envoi ave lui un header qui contient le jwt :

```
import { HttpInterceptorFn } from '@angular/common/http';
import {inject} from '@angular/core';
import {AuthService} from '../services/auth.service';
import {catchError, throwError} from 'rxjs';

export const appHttpInterceptor: HttpInterceptorFn = (req, next) => {
  if(!req.url.includes('auth/login')) {
    const authService = inject(AuthService);
    let newRequest = req.clone({
      headers: req.headers.set('Authorization', 'Bearer ' +
authService.accessToken)
    });
```

```
    return next(newRequest).pipe(
      catchError((error) => {
        if (error.status === 401) {
          authService.logout();
        }
        return throwError(() => error);
      })
    );
  }else {
    return next(req);
  }
};
```

maintenant en cree les guards :

pour securiser l'acces a notre frontend pour les utilisateurs non authentifie en va cree l'authentication guard pour bloquer les acces pour les utilisateurs non authentife et le redirecter vers la page de login:

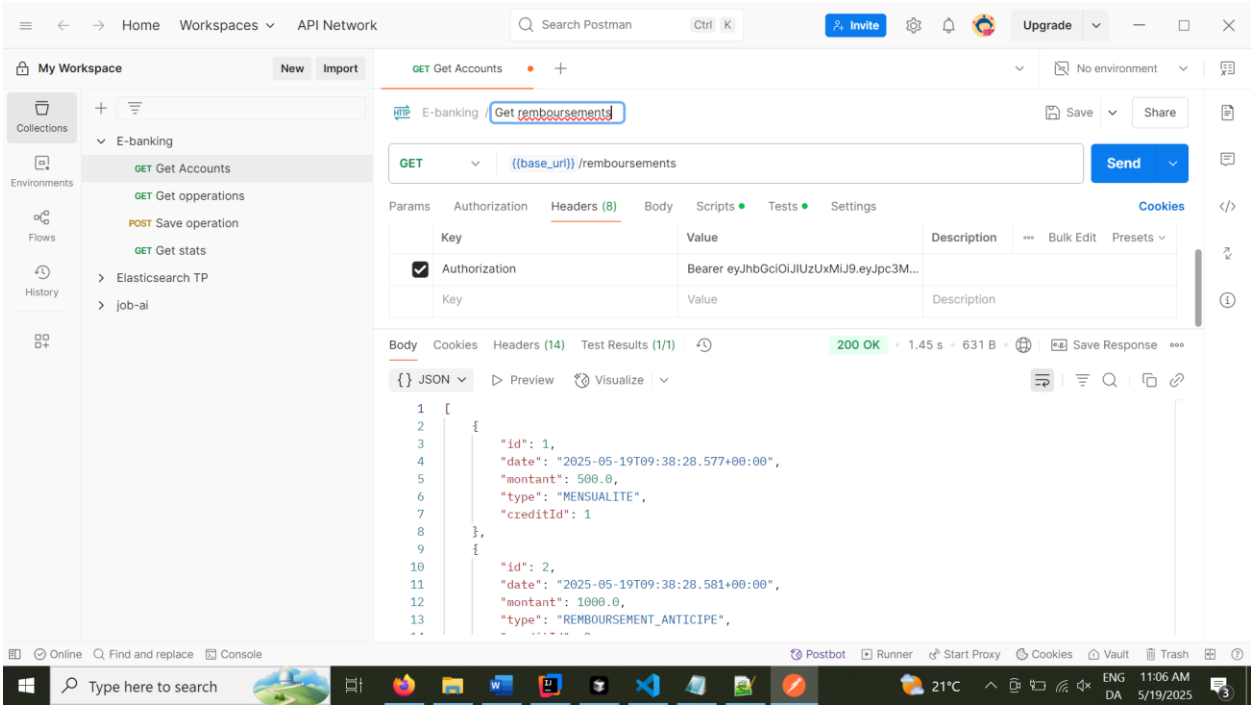
```
import {CanActivateFn, Router} from '@angular/router';
import {AuthService} from '../services/auth.service';
import {inject} from '@angular/core';

export const authenticationGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);
  if(authService.isAuthenticated) {
    return true;
  }else{
    router.navigateByUrl("/login").then();
    return false;
  }
};
```

en creer authorization guard pour limite l'acces pour certain utilisateur a quellques endroits :

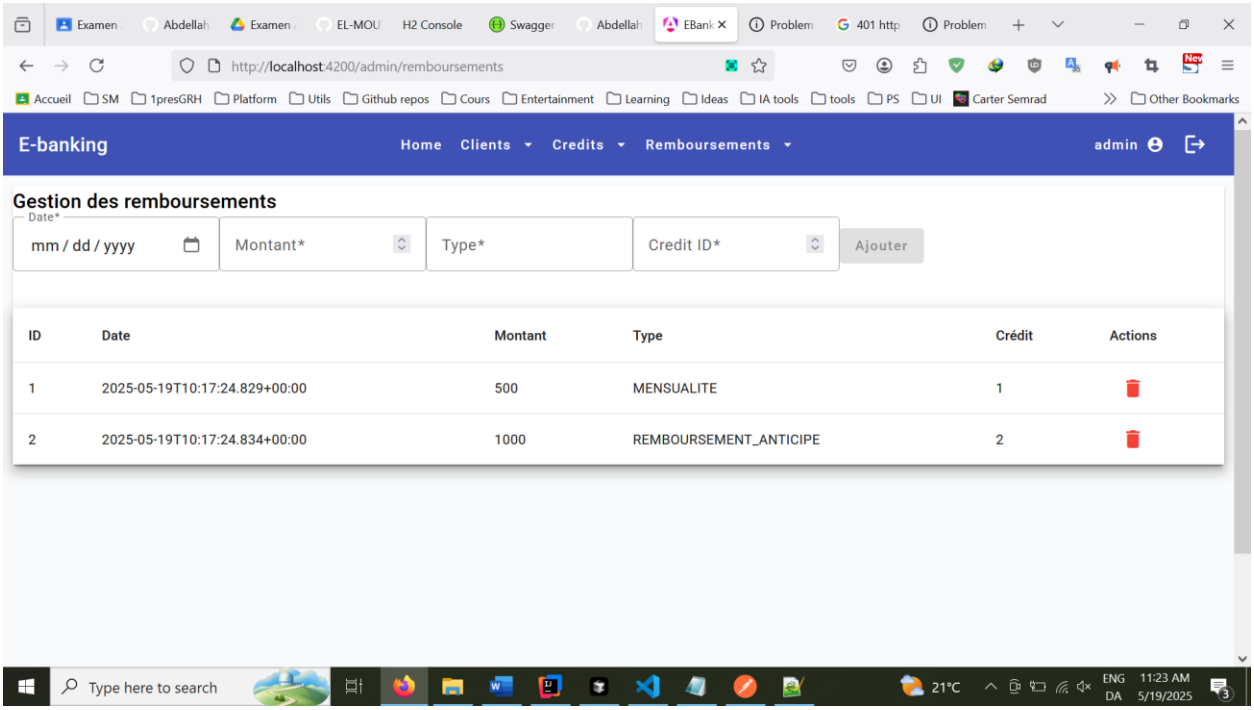
```
import {CanActivateFn, Router} from '@angular/router';
import {inject} from '@angular/core';
import {AuthService} from '../services/auth.service';

export const authorizationGuard: CanActivateFn = (route, state) => {
  const authService = inject(AuthService);
  const router = inject(Router);
  if(authService.roles.includes('ROLE_ADMIN')) {
    return true;
  }else {
    router.navigateByUrl("/not-authorized").then();
    return false;
  }
};
```

si l'utilisateur n'a pas le role il le redirige vers la page not-authorized

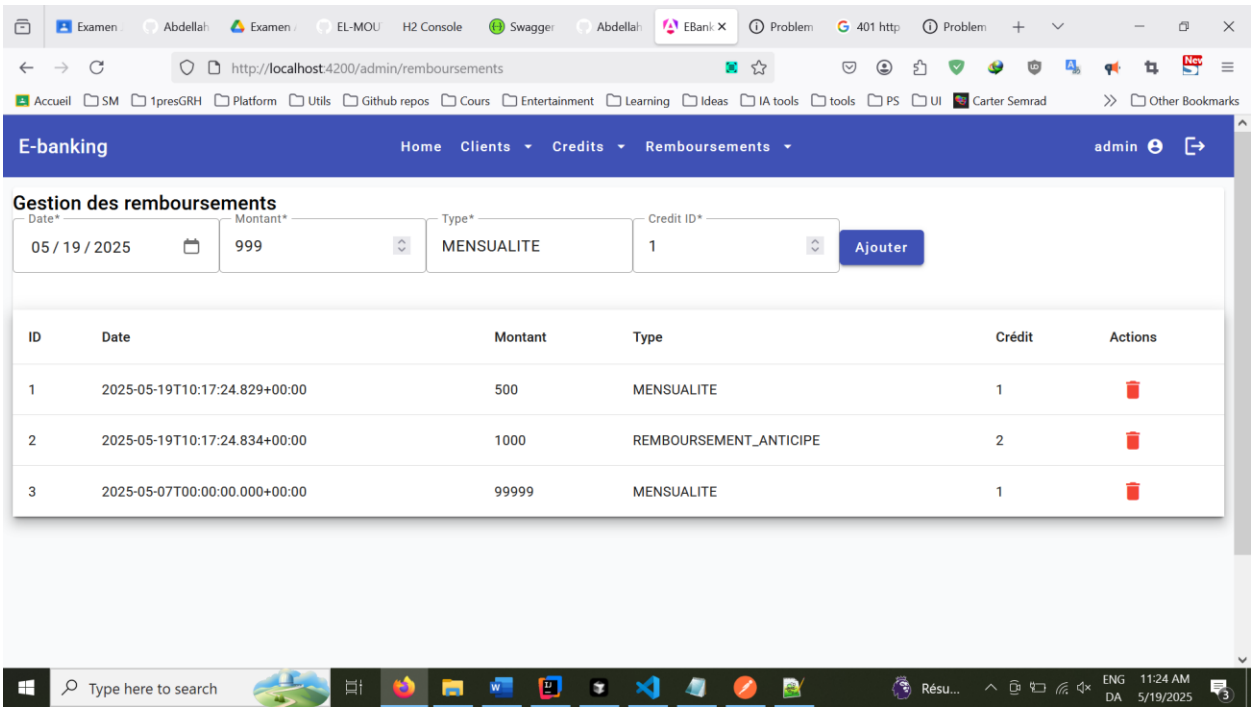
en teste avec postman en envoyant le jwt :



- **La page des remboursement :**

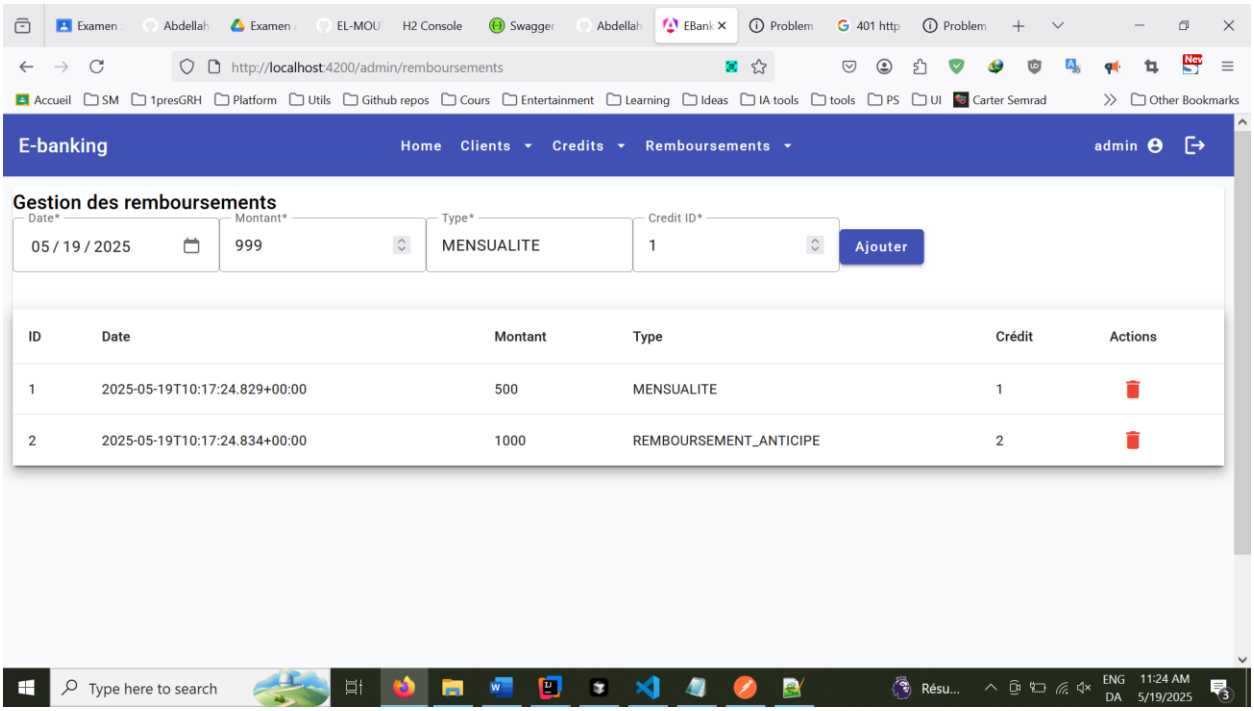


En ajoute un rembourssment :

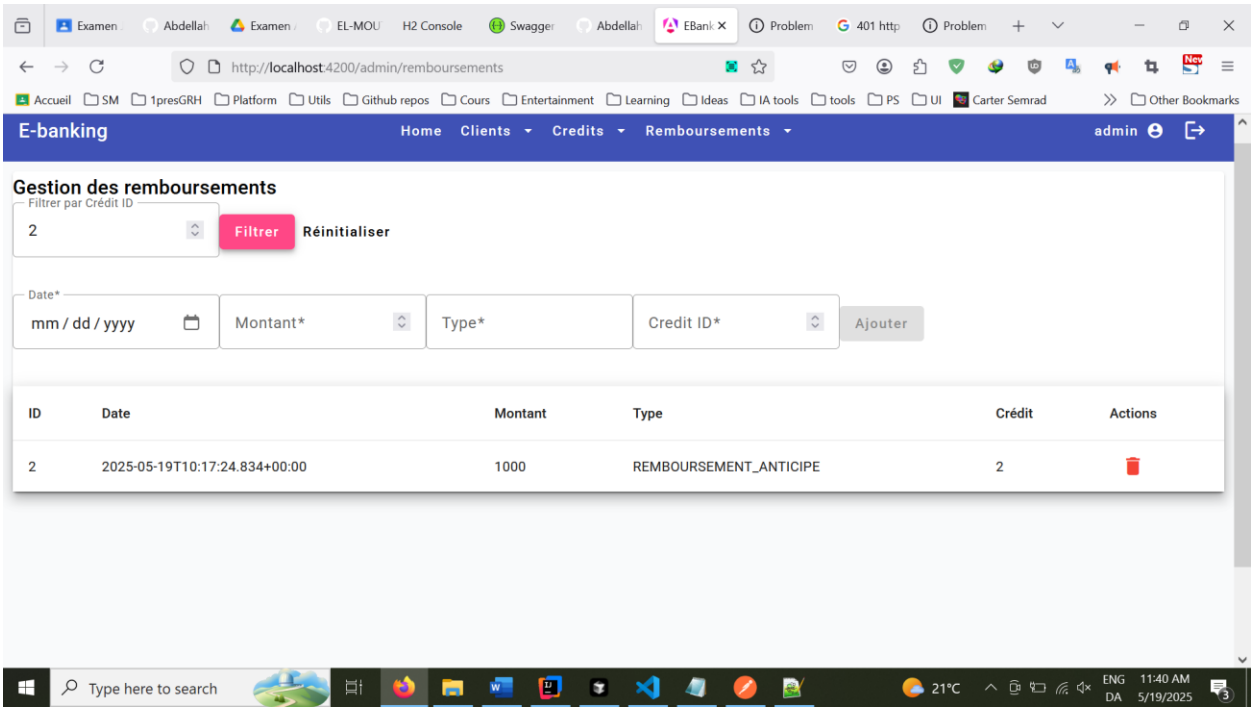


Le remboursement est bien ajouter

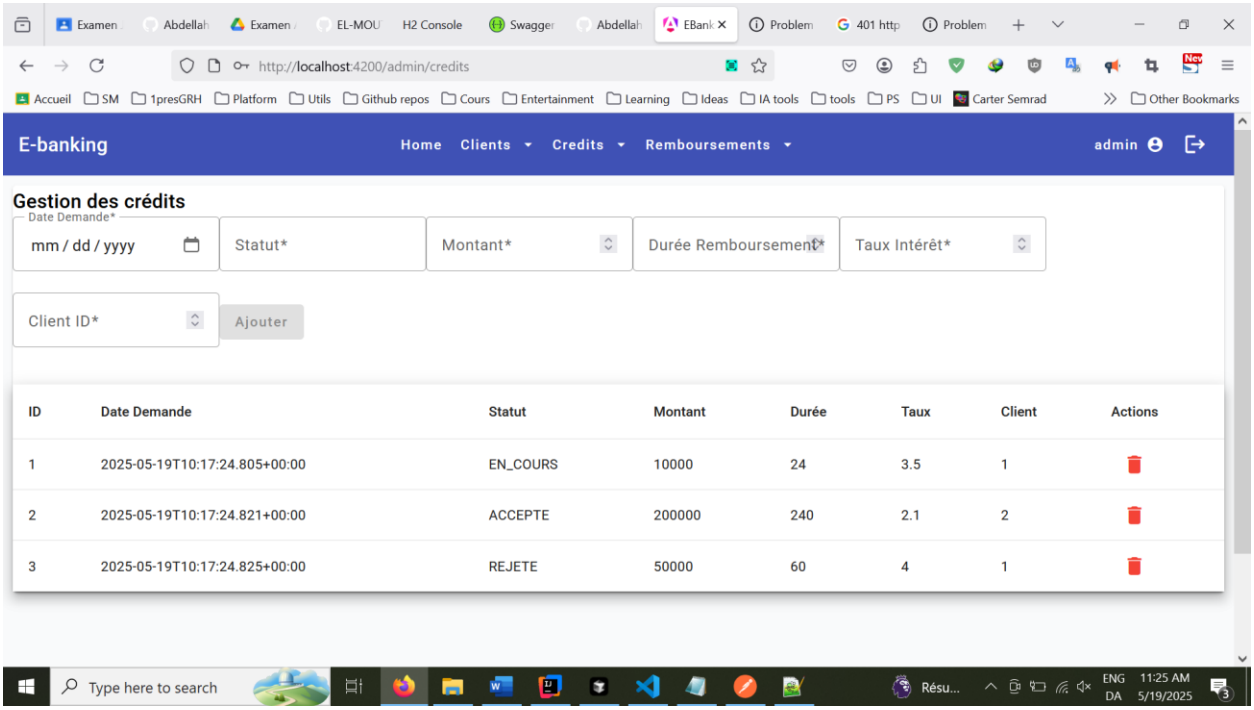
En le supprime :



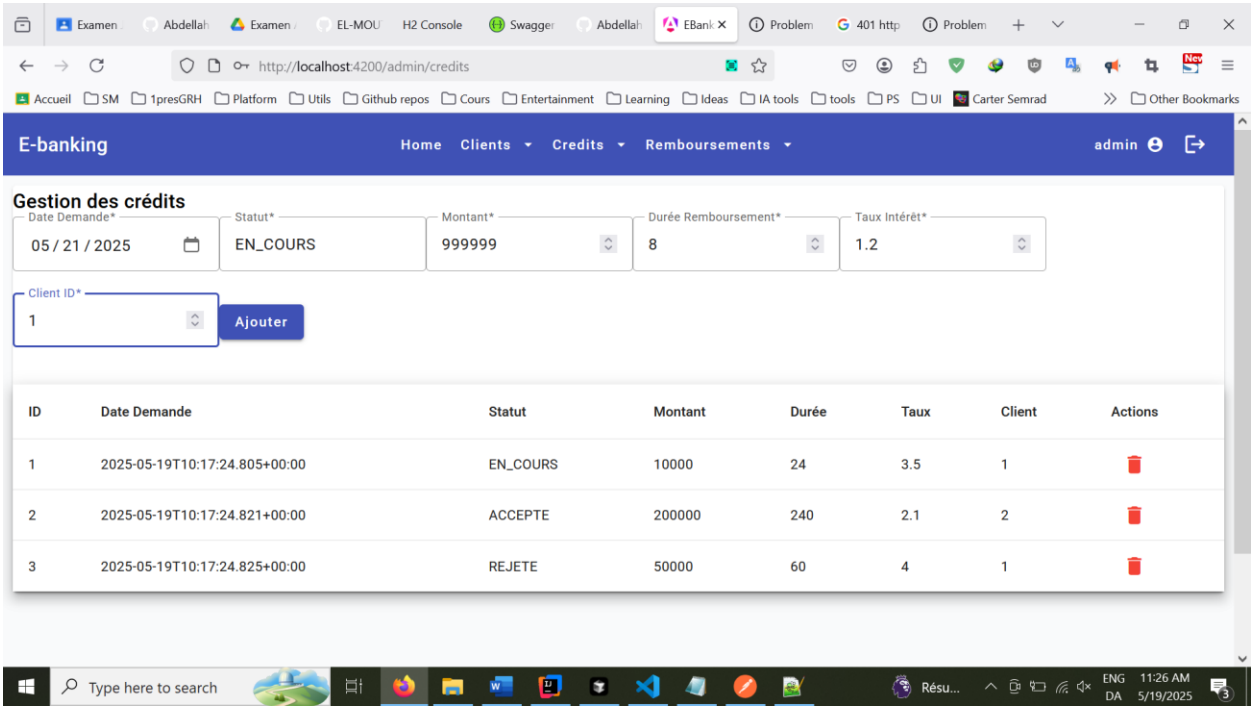
Maintenant on le filtre avec l'id de credit (avec l'id 2):



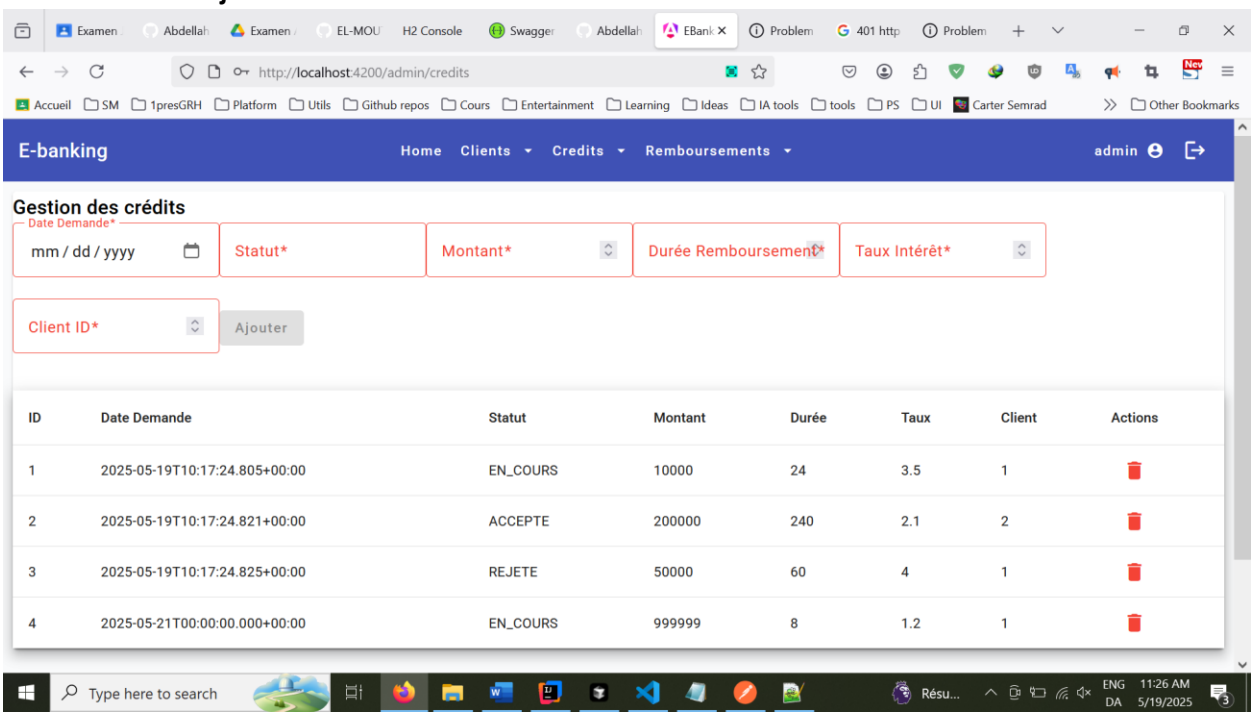
- **La page des credits :**



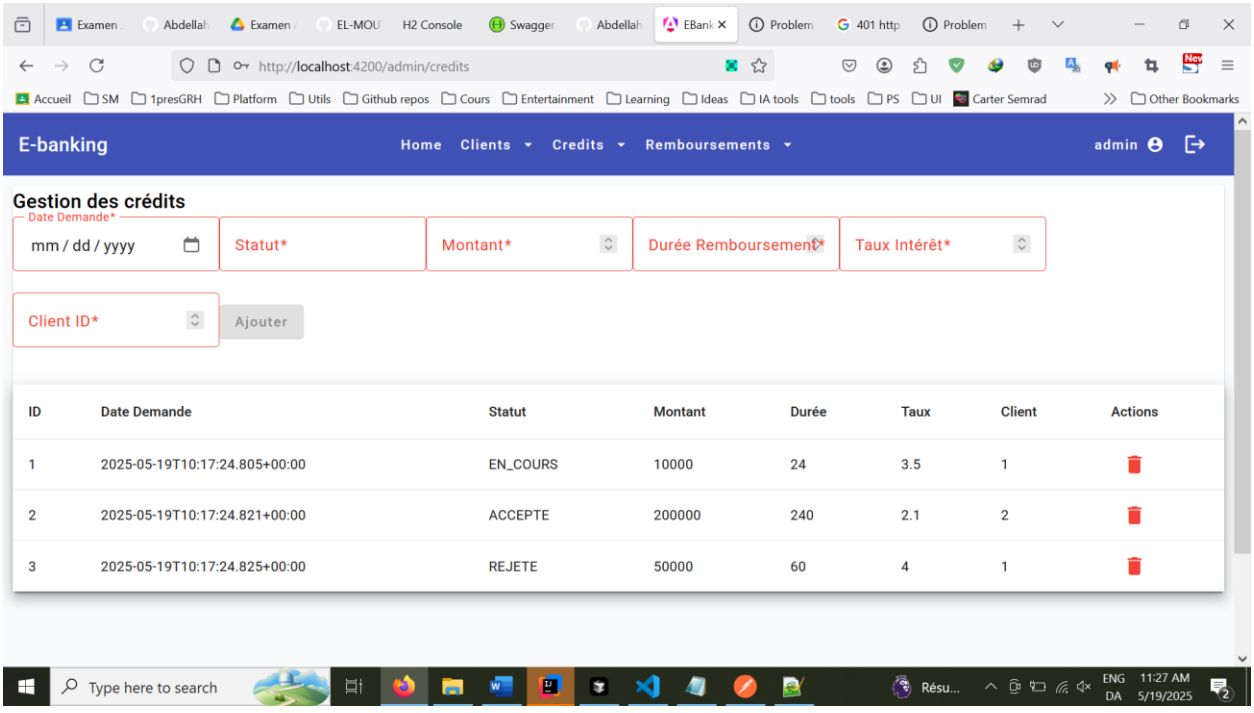
En ajoute un credit :



Le credit et bien ajouter

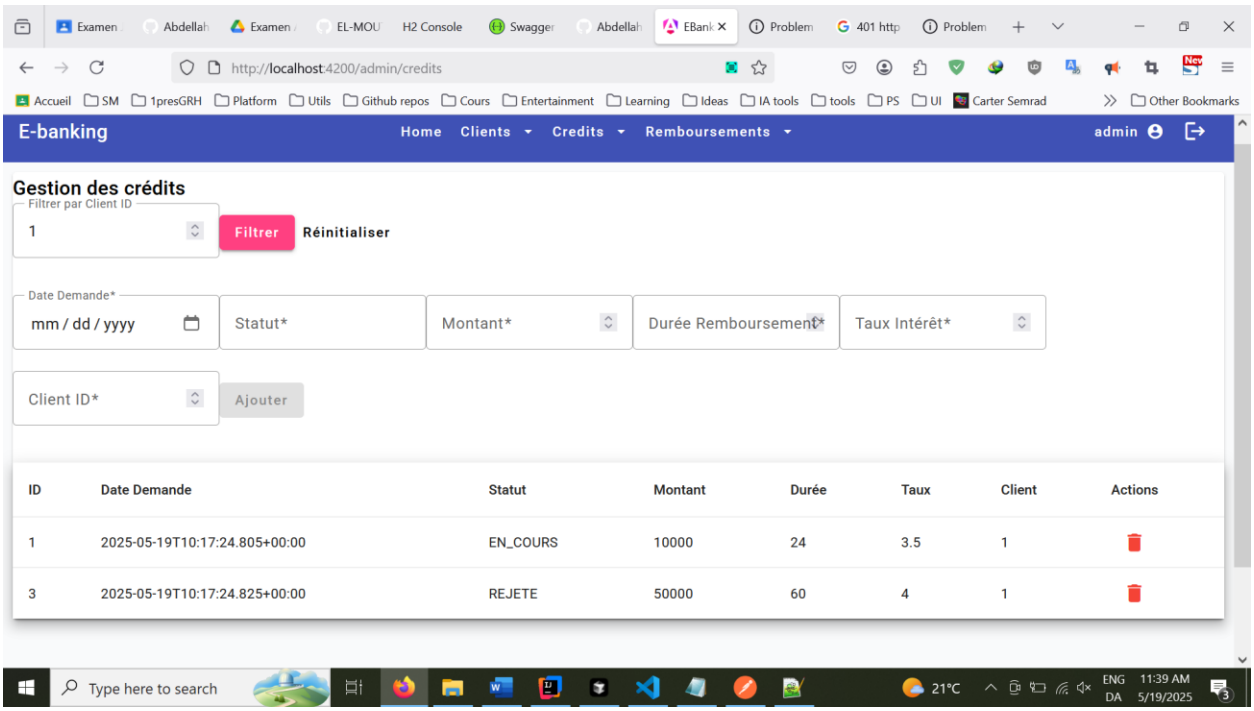


En le supprime maintenant :

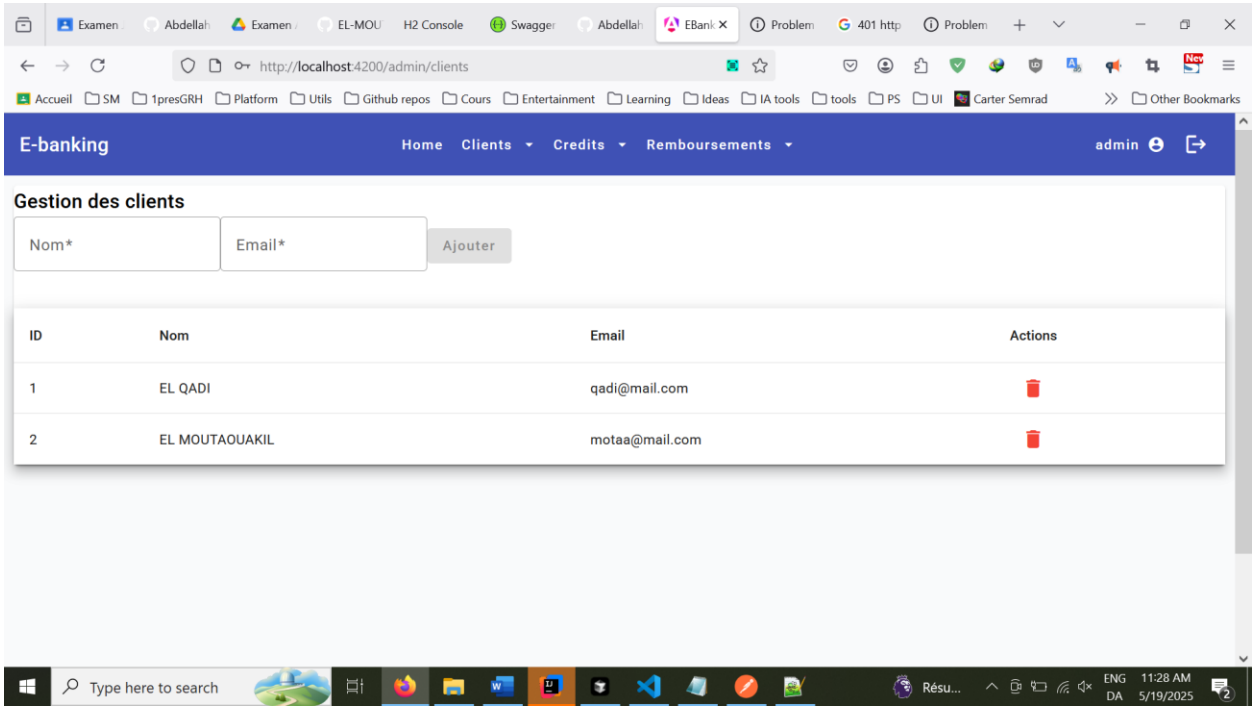


Il y bien supprime

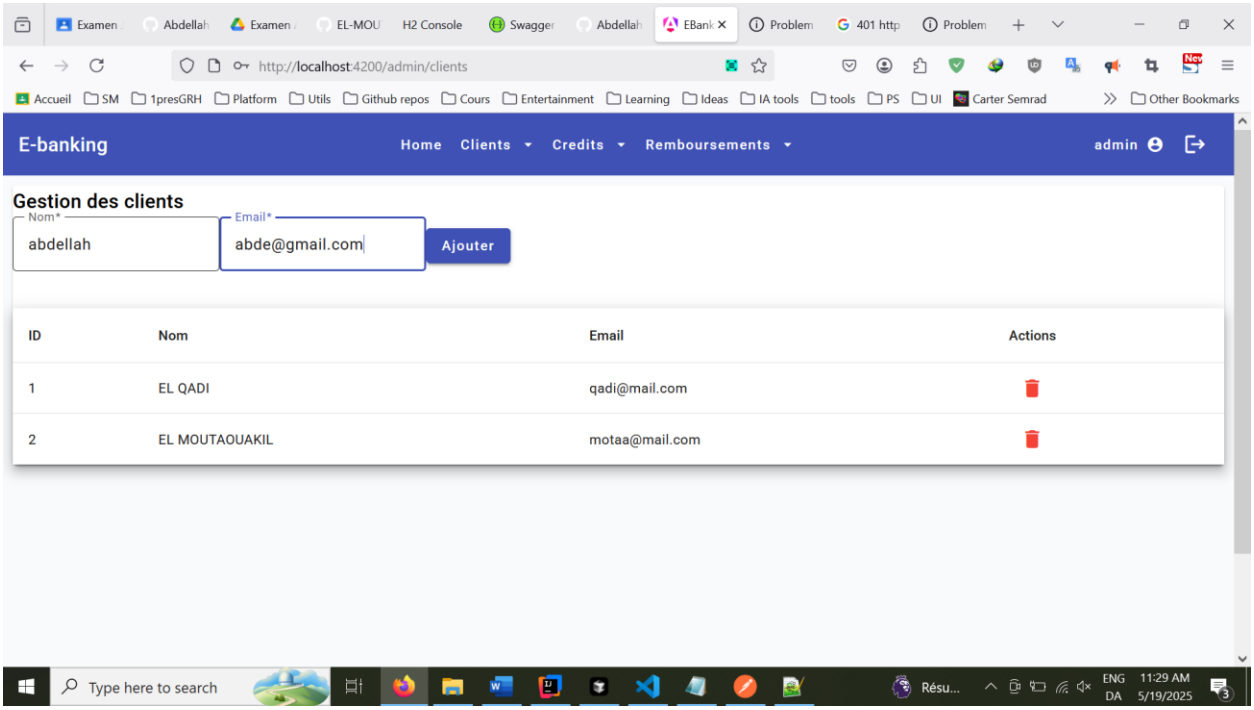
Maintenant en le filtre ave l'id de client(client avec id 1) :


E-banking
Home
Clients
Credits
Remboursements
admin
Gestion des crédits
Filtrer par Client ID
1
Filtrer
Réinitialiser
Date Demande*
mm / dd / yyyy
Statut*
Montant*
Durée Remboursement*
Taux Intérêt*
Client ID*
Ajouter
ID
Date Demande
Statut
Montant
Durée
Taux
Client
Actions
1
2025-05-19T10:17:24.805+00:00
EN_COURS
10000
24
3.5
1
3
2025-05-19T10:17:24.825+00:00
REJETE
50000
60
4
1
Type here to search
21°C
ENG
DA
11:39 AM
5/19/2025

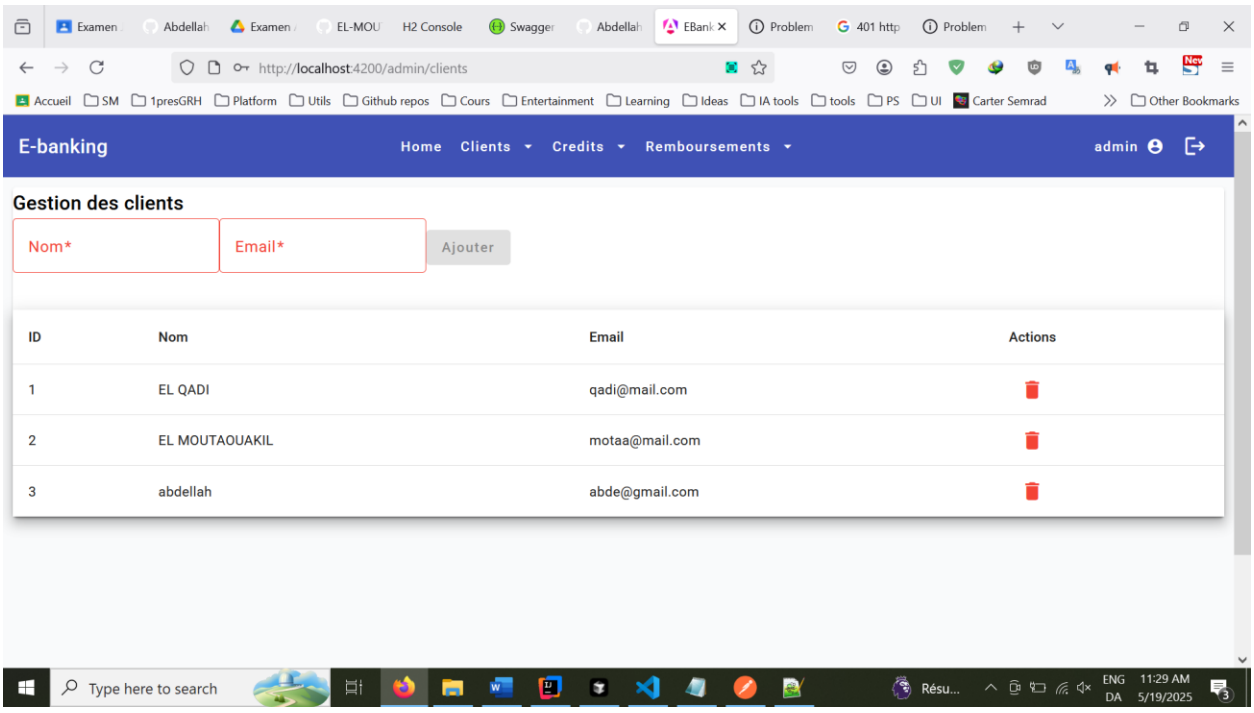
- **La page des clients**



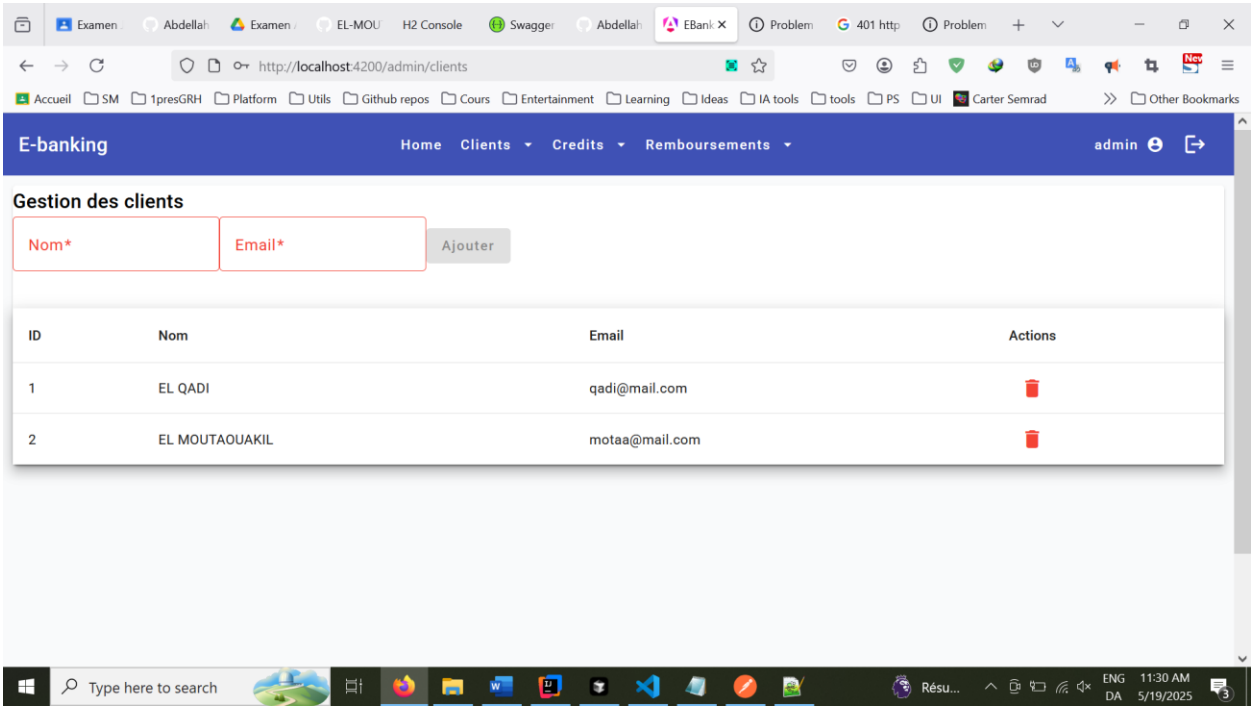
En ajoute un client :



Le client est bien ajouter :



Maintenant en le supprime :



Il y bien supprimer.