Réaliser par :

Mohamed amine KHAMMOUR

L’énoncé du projet :

Projet Digital Banking

Mohamed YOUSSFI • 4 mai

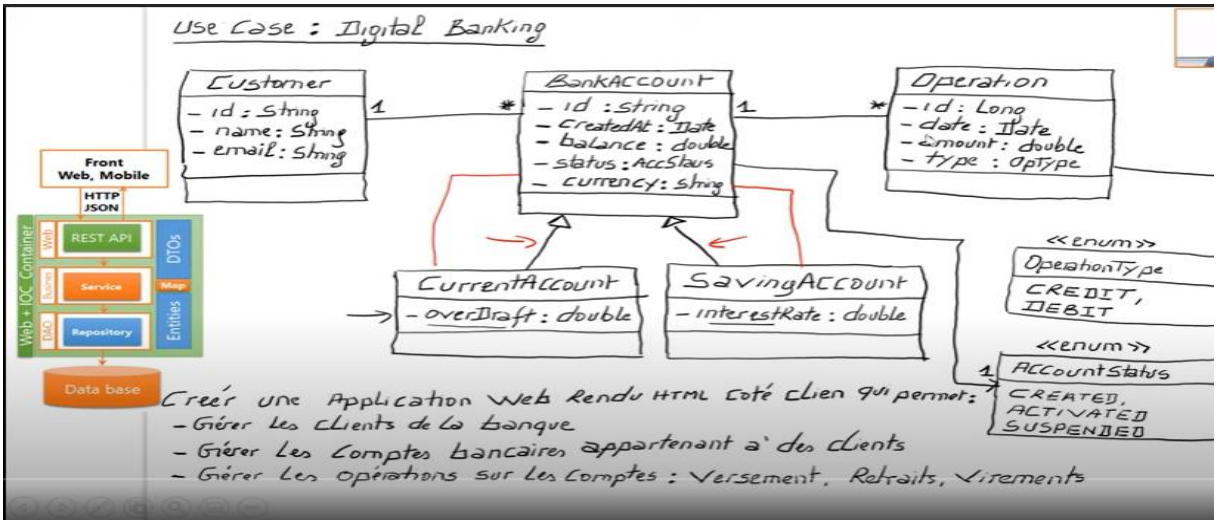
100 points

On souhaite créer une application Web basée sur Spring et Angular qui permet de gérer des comptes bancaires. Chaque compte appartient à un client il existe deux types de comptes : Courant et Epargnes. chaque Compte peut subir des opérations de types Débit ou crédit.
L'application se compose des couches suivantes :
- Couche DAO (Entités JPA et Repositories)
- Couche Service définissant les opérations suivantes :
  - Ajouter des comptes
  - Ajouter des client
  - Effectuer un débit (Retrait)
  - Effectuer un crédit (Versement)
  - Effectuer un virement
  - Consulter un compte
- La couche DTO
- Mappers (DTO <=>Entities)
- La couche Web (Rest Controllers)
- Couche sécurité (Spring Security avec JWT)

Première partie du projet (Voir la vidéo : https://www.youtube.com/watch?v=muuFQWnCQd0)

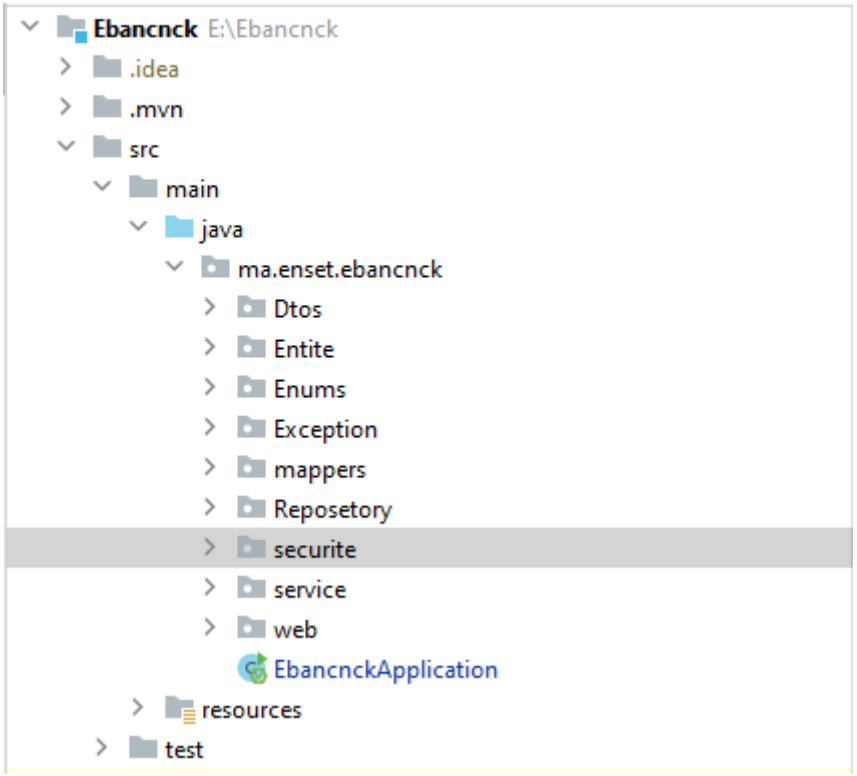
Travail à faire :
1. Créer et tester la couche DAO (Voir la vidéo : https://www.youtube.com/watch?v=muuFQWnCQd0)
2. Créer et tester la couche service
3. Créer et tester la couche Web (Rest Controller)
4. Modifier la couche service et la couche web en utilisant les DTO
5. Créer un service d'authentification séparé basé sur Spring Security et JWT (Voir https://www.youtube.com/watch?v=3q3w-RT1sg0)
6. Sécuriser l'application Digital Banking en utilisant Spring Security et JWT
7. Créer la partie Frontend Web en utilisant Angular
8. Créer la partie Frontend Mobile avec Flutter

Digramme de Class :

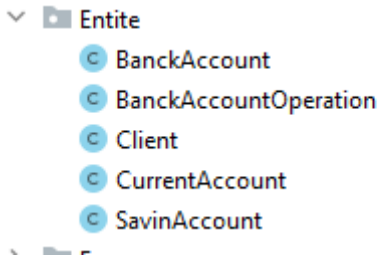


La parité Backend :

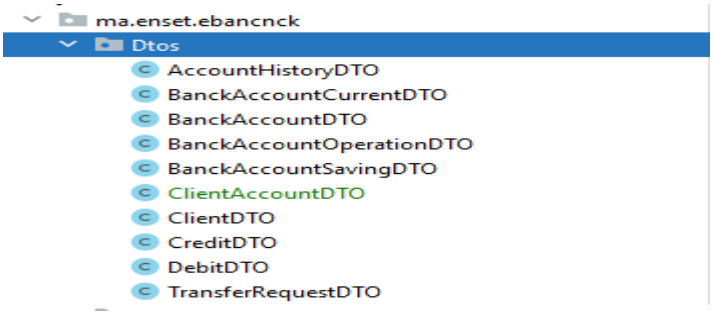
La Structure du projet :



Les entités principales de notre projet :



Les DTO : Data Transfert Object



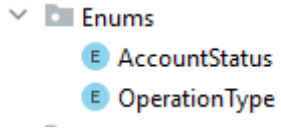
La Class **ClientAccountDTO**

```
@Data
@AllArgsConstructor
@NoArgsConstructor
    public class ClientAccountDTO {

    private Long clientId;
    private int currentPage;
    private int totalPages;
    private int pageSize;
    private List<BanckAccountDTO> accounts;

}
```

Les enumérateurs :

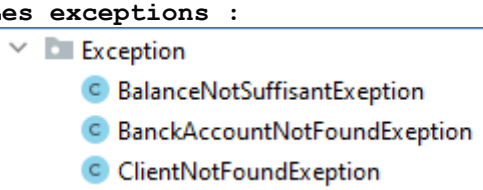


L'enumérateur **OperationType**

```
package ma.enset.ebancnck.Enums;

public enum OperationType {
    DEBIT,CEREDIT
}
```

Les exceptions :

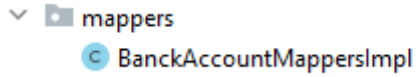


L'exception **ClientNotFoundExeption**

```
package ma.enset.ebancnck.Exception;

public class ClientNotFoundExeption extends Exception {
    public ClientNotFoundExeption(String message) {
        super(message);
    }
}
```

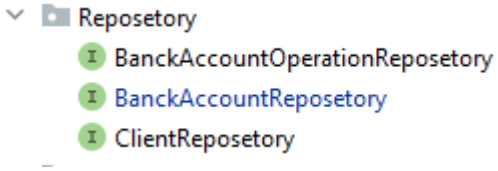
les mapper



Extrais de la class **BanckAccountMappersImpl**

```
@Service
public class BanckAccountMappersImpl {
 public ClientDTO formCustomer(Client client){
         ClientDTO clientDTO=new ClientDTO();
     BeanUtils.copyProperties(client,clientDTO);
     return clientDTO;
 }
    public Client formCustomerDTO(ClientDTO clientDTO){
        Client client1=new Client();
        BeanUtils.copyProperties(clientDTO,client1);
        return client1;
    }
```

Les repositories :



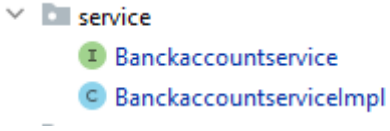
L'interface **BanckAccountRepository : la méthode findBanckAccountByClientId permet de retourner une page des compte a travers l'id du client**

```
import java.util.List;

public interface BanckAccountRepository extends JpaRepository<BanckAccount,String> {
    public Page<BanckAccount> findBanckAccountByClient_Id(long id,Pageable pageable);

}
```

Les services



Extrais de l'interface **Banckaccountservice**

```java
public interface Banckaccountservice {
    ClientDTO getClient(long idClient) throws ClientNotFoundExeption;
    ClientDTO saveClient(ClientDTO client);

    BanckAccountCurrentDTO savaCurrentBanckacount(double innistialBlanc, double overDraft, Long clientId) throws ClientNotFoundExeption;
    BanckAccountSavingDTO savaSvingBanckacount(double innistialBlanc, double intersRate, Long clientId) throws ClientNotFoundExeption;
    List<ClientDTO> listClients();
    BanckAccountDTO getBanckAccount(String idAccount) throws BanckAccountNotFoundExeption;
    void debit(String accountId,double amount,String description) throws BanckAccountNotFoundExeption, BalanceNotSuffisantExeption;
    void credit(String accountId,double amount,String description) throws BalanceNotSuffisantExeption, BanckAccountNotFoundExeption;
    void transfert(String accountId,String accountDestination,Double amount) throws BalanceNotSuffisantExeption, BanckAccountNotFoundExeption;
    List<BanckAccountDTO> banckAccountList();
    ClientDTO updatClient(ClientDTO client);
```

La class **BanckaccountserviceImpl** qui implémenté l'interface **Banckaccountservice**

```java
@Service
@Transactional
@AllArgsConstructor
public class BanckaccountserviceImpl implements Banckaccountservice {

    private ClientRepository clientRepository;
    private BanckAccountRepository banckAccountRepository;

    private BanckAccountOperationRepository banckAccountOperationRepository;

    private BanckAccountMappersImpl banckAccountMappers;

    @Override
    public ClientDTO getClient(long idClient) throws ClientNotFoundException {
        return banckAccountMappers.formCustomer(clientRepository.findById(idClient).orElseThrow(()->
    }
```

Les Controller :

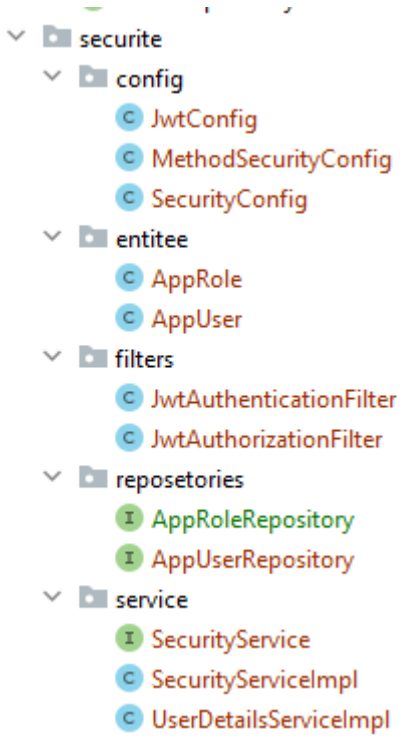
La class ClientController

```java
@RestController
@Slf4j
@AllArgsConstructor
@CrossOrigin("*")
public class ClientRestController {
    private Banckaccountservice banckaccountservice;
    private BanckAccountRepository banckAccountRepository;
@GET("/clients")
    public List<ClientDTO> clients() { return banckaccountservice.listClients(); }

    @GET("/clients/search")
    public List<ClientDTO> clientsSerch(@Request Param(name = "keyword",defaultValue = "")String keyword){
        return banckaccountservice.gestSerchClient( keyword: "%"+keyword+"%");
}

@GET("/clients/{id}")
public ClientDTO getClient(@Path Variable(name = "id") Long idClient) throws ClientNotFoundExeption {
    return  banckaccountservice.getClient(idClient);
}
```

La partie Sécurité avec JWT :



La class JwConfig

```
public class JwtConfig {
    public static String SECRET_PHRASE = "SECRET_PHRASE";
    public static int ACCESS_TOKEN_EXPIRATION = 24*60*60*1000; // 24hrs
    public static String AUTHORIZATION_HEADER = "Authorization";
    public static String TOKEN_HEADER_PREFIX = "Bearer ";
    public static String REFRESH_PATH = "/V1/refresh-token";
    public static int REFRESH_TOKEN_EXPIRATION = 48*60*60*1000; // 48hrs
}
```

La Class MethodSecuriyConfig

```
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration;

@Configuration
@EnableGlobalMethodSecurity( prePostEnabled = true)
public class MethodSecurityConfig
        extends GlobalMethodSecurityConfiguration {
}
```

La Class **sécuriteConfig** dans laquelle on a travailler avec la Stratégie User Detaille Service

```
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    private UserDetailsServiceImpl detailsService;


    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService( detailsService );
    }
```

Les filtres :
Extrais de La class **JwtAuthenticationFilter**

```
@Slf4j
public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {

    private AuthenticationManager authenticationManager;

    public JwtAuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }
```

Extrait de la class **JwtAuthorizationFilter**

```
@Slf4j
public class JwtAuthorizationFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterCh
        if (request.getServletPath().equals(JwtConfig.REFRESH_PATH)) { // refresh token
            filterChain.doFilter(request, response);// continue
            return;
        }
        String jwt_token = request.getHeader(JwtConfig.AUTHORIZATION_HEADER); //Authorization: Bearer xxx
        if (jwt_token != null && jwt_token.startsWith(JwtConfig.TOKEN_HEADER_PREFIX)) { // does it have a passport
            try {
                Algorithm algorithm = Algorithm.HMAC256(JwtConfig.SECRET_PHRASE); // secret phrase
                JWTVerifier verifier = JWT.require(algorithm).build(); // build the verifier
                String jwt = jwt_token.substring(7); // remove "Bearer "
                DecodedJWT decodedJWT = verifier.verify(jwt); // verify the token
```

Les services de la Sécurités

L'interface de la **SecurityService**

```
public interface SecurityService {

    AppUser addNewUser(AppUser appUser);
    AppRole addNewRole(AppRole appRole);
    void addRoleToUser( String username, String roleName);
    AppUser loadUserByUsername( String username);
    List<AppUser> listUsers();
    // login
    AppUser login(AppUser appUser);
    // logout
}
```

Extrait L'implémentation de l'interface **SecurityService**

```
@Override
public ClientDTO saveClient(ClientDTO client) {
   Client saveClint= banckAccountMappers.formCustomerDTO(client);
   Client clientsave=clientReposetory.save(saveClint);
    return banckAccountMappers.formCustomer(clientsave);
}

@Override
public BanckAccountCurrentDTO savaCurrentBanckacount(double innistialBlanc, double overDraft, Long clientId)
    Client client= clientReposetory.findById(clientId).orElse( other: null);
    if(client==null)
        throw new ClientNotFoundExeption(" Client not found");
    CurrentAccount currentAccount=new CurrentAccount();

    currentAccount.setId(UUID.randomUUID().toString());
    currentAccount.setCreateAt(new Date());
    currentAccount.setBalance(innistialBlanc);
    currentAccount.setOverDraft(overDraft);
    currentAccount.setClient(client);
    CurrentAccount savedbanckAcount=banckAccountReposetory.save(currentAccount);
    return banckAccountMappers.fromCurrentAccount(savedbanckAcount);
}
```

Les Repositories de la partie de sécurité

L'interface AppUserReposetory

```
@Repository
public interface AppUserRepository extends JpaRepository<AppUser, String> {
    AppUser findByUsername(String username);
}
```

L'interface AppRoleReposetory

```
@Repository
public interface AppRoleRepository extends JpaRepository<AppRole, Long> {
    AppRole findByRoleName(String roleName);
}
```

La Documentation Swagger :

Les Controller des Clients

http://localhost:8088 - Generated server url

client-rest-controller

PUT	/clients/{idClienet}
GET	/clients
POST	/clients
GET	/clients/{id}
GET	/clients/search
GET	/client/{id}/accounts
DELETE	/clinets/{id}

Les Controller des Comptes

banck-acount-rest-controller

POST	/saveDebit/{id}
POST	/accounts/transfer
POST	/accounts/debit
POST	/accounts/credit
GET	/accounts
GET	/accounts/{idAccount}
GET	/accounts/{idAccount}/pageoperations
GET	/accounts/{idAccount}/operations

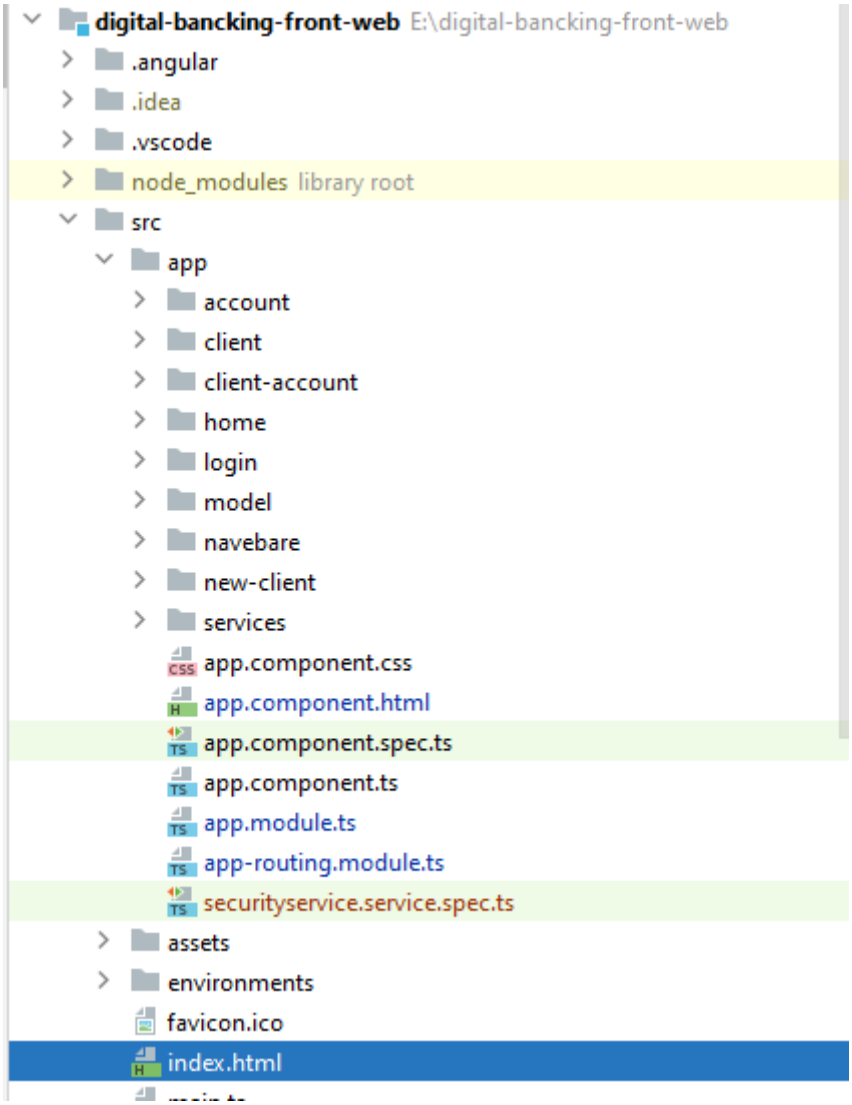
Les Controller de la sécurité

security-rest-controller

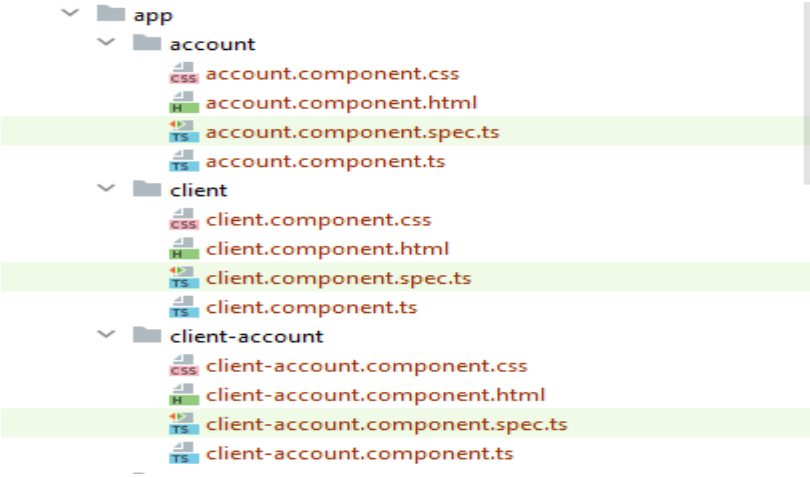
GET	/v1/v1/refresh-token
GET	/v1/v1/profile

La Partie Frontend :**Angullar**

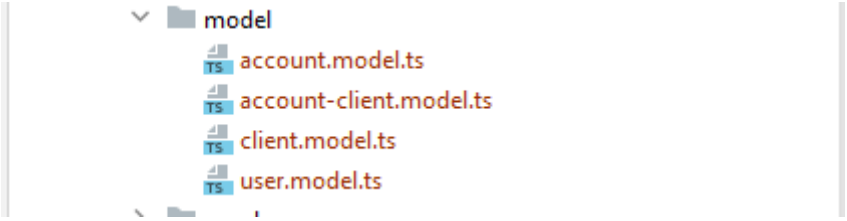
Structure de projet



Les déférent **composant** de notre projet qu'on a crier a l'aide de la command ng g c



Le model de notre projet

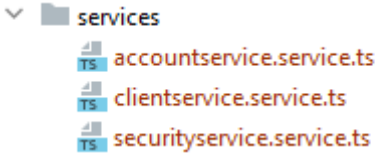


Le Model des comptes

```
export interface AccountDetails {
  accountId:            string;
  balance:              number;
  currentpage:          number;
  totalepages:          number;
  pageize:              number;
  accountOperationDTOS: AccountOperationDTO[];
}

export interface AccountOperationDTO {
  id:            number;
  operationDate: Date;
  amount:        number;
  type:          string;
  description:   string;
}
```

Les services du projet :



AccountserviceService

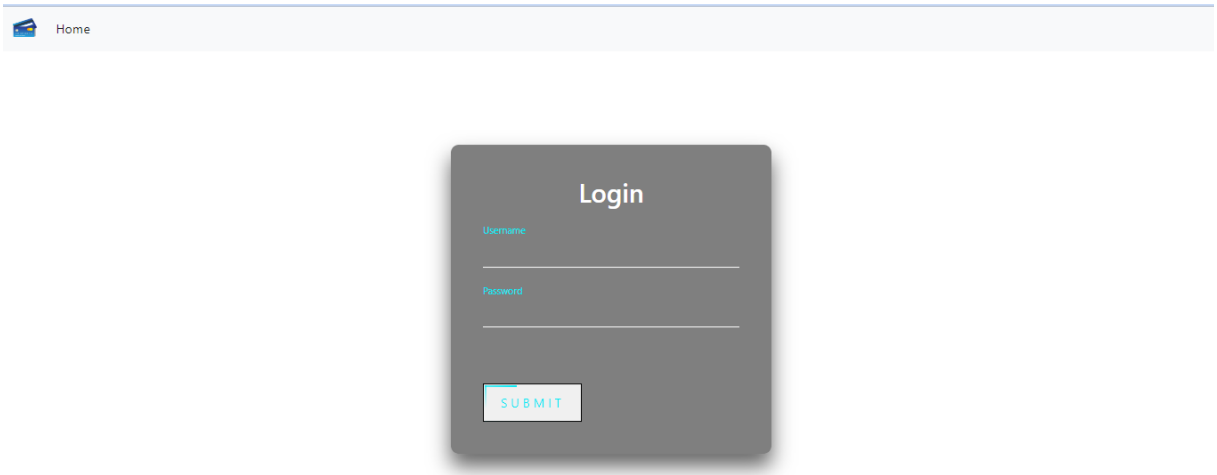
```
  constructor(private http :HttpClient) { }
  public getAccount(accountId:string,page:number,size:number):Observable<AccountDetails>{
    return this.http.get<AccountDetails>( url: environment.backendHoste+"/accounts/"+accountId+"/pageoper
  }
  public debit(accountId : string, amount : number, description:string){
    let data={accountId : accountId, amount : amount, description : description}
    return this.http.post( url: environment.backendHoste+"/accounts/debit",data);
  }
  public credit(accountId : string, amount : number, description:string){
    let data={accountId : accountId, amount : amount, description : description}
    return this.http.post( url: environment.backendHoste+"/accounts/credit",data);
  }
  public transfer(accountSource: string,accountDestination: string, amount : number, description:strin
    let data={accountSource, accountDestination, amount, description }
    return this.http.post( url: environment.backendHoste+"/accounts/transfer",data);
  }

}
```

La réalisations :

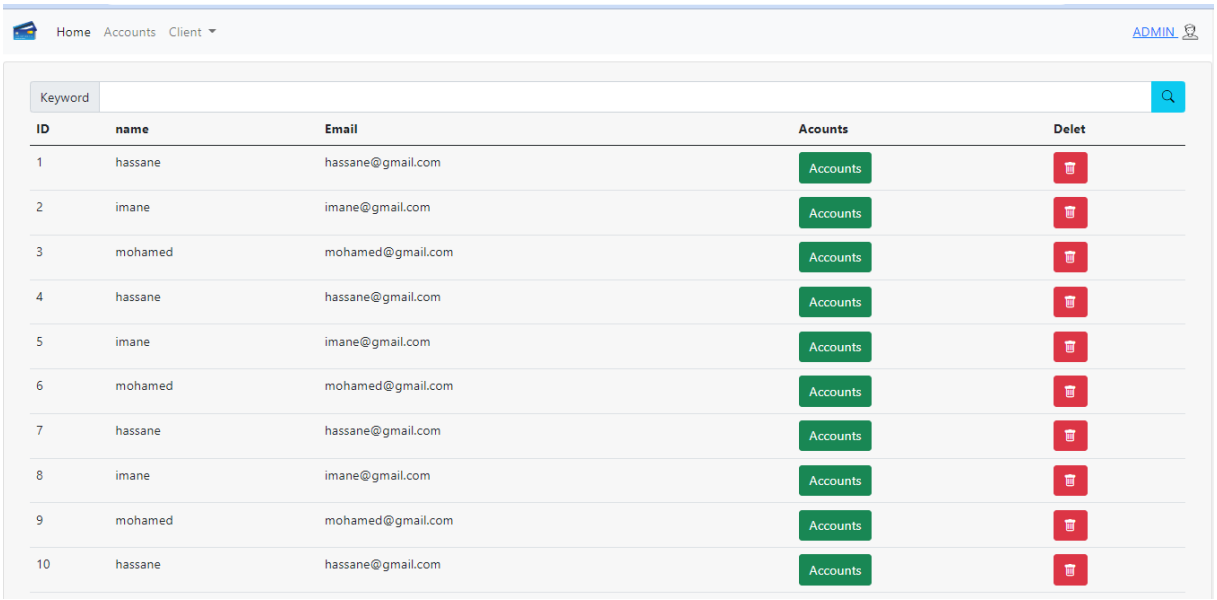
La page Home avant le login

La page de login :



Apres le login

La List des client



ID	name	Email	Acounts	Delet
1	hassane	hassane@gmail.com	Accounts	
2	imane	imane@gmail.com	Accounts	
3	mohamed	mohamed@gmail.com	Accounts	
4	hassane	hassane@gmail.com	Accounts	
5	imane	imane@gmail.com	Accounts	
6	mohamed	mohamed@gmail.com	Accounts	
7	hassane	hassane@gmail.com	Accounts	
8	imane	imane@gmail.com	Accounts	
9	mohamed	mohamed@gmail.com	Accounts	
10	hassane	hassane@gmail.com	Accounts	

Les comptes de chaque client :

Home Accounts Client ADMIN

client ID 1:

acount ID	Date	Type	Amount
0f8682c7-a742-4fb3-baaa-ebb3ce986d6e	03-06-2022:14-44-26	SavinAccount	9,428,084.70
38b4baa2-ea7f-40be-b39d-f7af783b756f	02-06-2022:19-16-29	CurrentAccount	87,398,624.46
65b75696-2999-43da-b46e-697af2841b29	04-06-2022:20-32-38	SavinAccount	9,232,540.68
7c58da4b-2fde-41d0-ae18-aa7dca58803a	02-06-2022:20-33-58	SavinAccount	9,938,829.93
7e561535-992d-4fa1-a14f-aa9663eeb04f	03-06-2022:14-44-26	CurrentAccount	86,265,989.82
a668b617-6b70-4aed-97b0-bd3783721105	04-06-2022:20-32-37	CurrentAccount	85,875,049.50
acc9f8a1-855b-4070-b0a1-80c31a154dff	03-06-2022:00-15-30	SavinAccount	9,530,560.70
d367c02a-75fb-4d57-b4d8-6d24aa4f53bf	03-06-2022:00-15-30	CurrentAccount	86,238,628.68
d5eb53d4-fda3-4f14-9007-ae1bcb7115dd	02-06-2022:19-16-29	SavinAccount	9,537,440.72
e1f1b4aa-6baf-4578-b6cf-2eab8d662c4d	02-06-2022:20-33-58	CurrentAccount	86,215,924.76

Les différents opération d'un compte :

Home Accounts Client ADMIN

Acoout

AccountId: 0f8682c7-a742-4fb3-baaa-ebb3ce986d6e Search

Account ID :**0f8682c7-a742-4fb3-baaa-ebb3ce986d6e**

Balance :**9,428,084.70**

ID	Date	Type	Amount
1265	03-06-2022:14-44-29	CEREDIT	31,238.38
1266	03-06-2022:14-44-29	DEBIT	63,935.96
1267	03-06-2022:14-44-29	CEREDIT	129,186.51
1268	03-06-2022:14-44-29	DEBIT	66,631.68
1269	03-06-2022:14-44-29	CEREDIT	117,455.31

0 1 2 3 4 5 6 7

Operations

DEBIT: CREDIT: TRANSFER:

Amount :

0

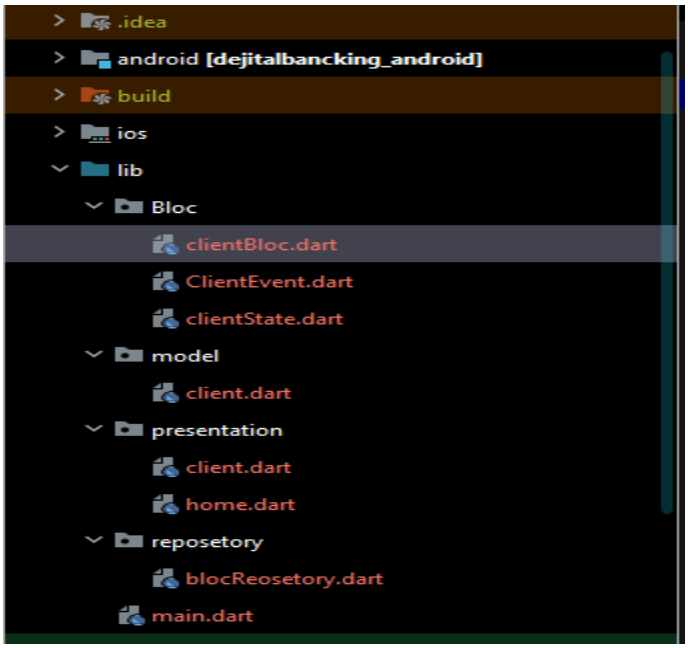
Description :

Save Operation

La partie flutter :

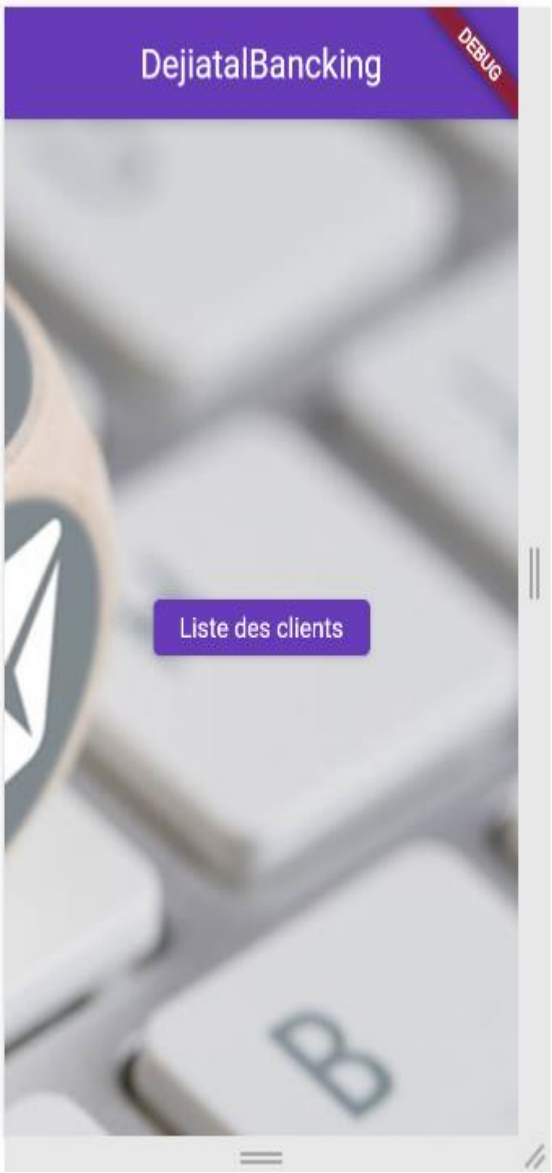
Structure du projet :

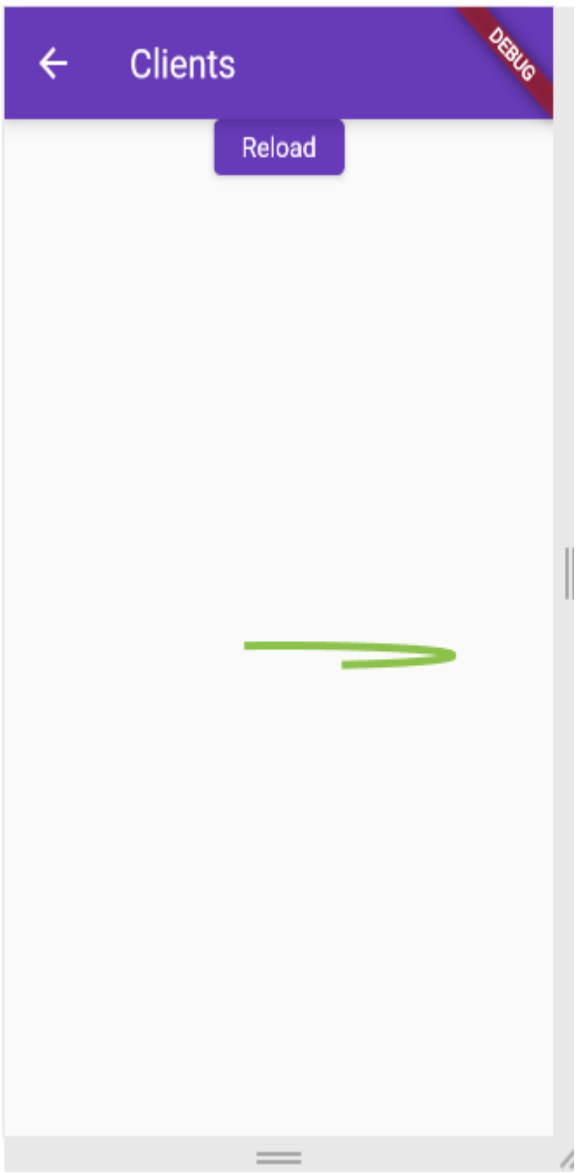
J'ai travaillé avec la stratégie Bloc

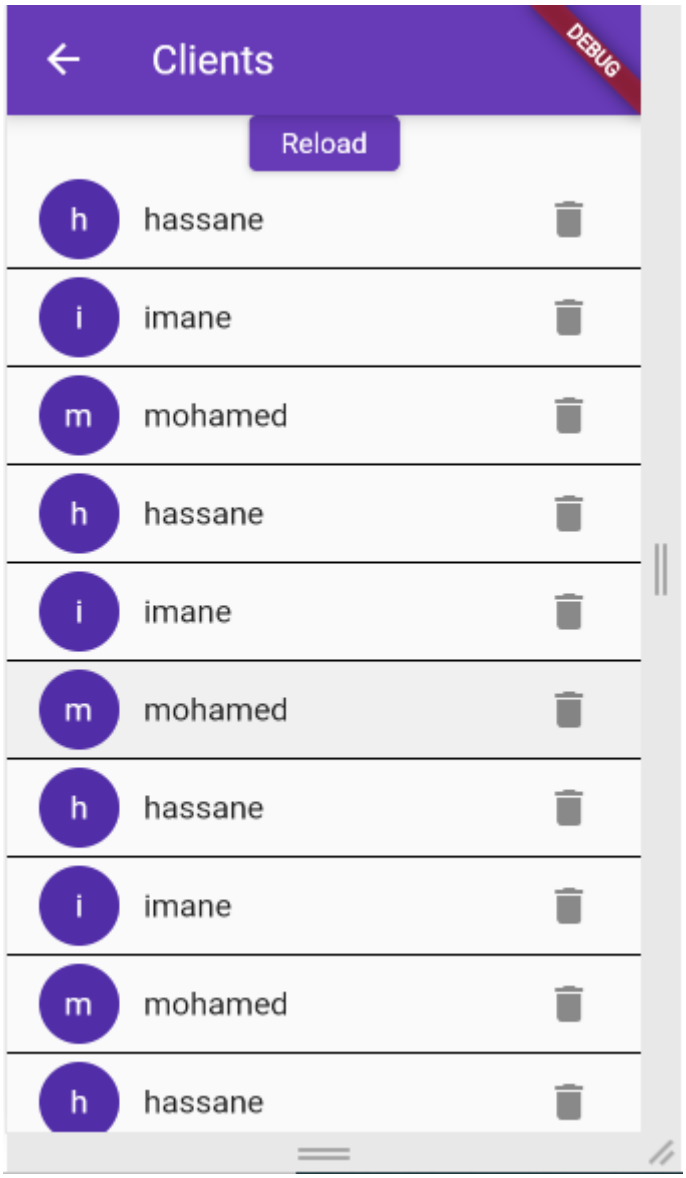


Les dependances

```
dependencies:
  flutter:
    sdk: flutter
  http:
  bloc:
  flutter_bloc:
  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
```






Clients
DEBUG
Reload
h hassane
i imane
m mohamed
h hassane
i imane
m mohamed
h hassane
i imane
m mohamed
h hassane