

# Projet digital Banking

## sommaire

INTRODUCTION : .....	2
LES OUTILS DE DEVELOPPEMENTS : .....	2
Création de la couche Dao : .....	2
Creation des repositories: .....	2
Tester la couche DAO : .....	3
Création de la couche Web : .....	4
La sécurité avec jwt : .....	5
Création de filtre : .....	6
Partie frontend : .....	7
Structure de projet .....	7
Exécution : .....	9
Page home : .....	9
Conclusion .....	15

Réaliser par : Ouassime Maarouf GLSID 2

## INTRODUCTION :

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.

## LES OUTILS DE DEVELOPPEMENTS :

Dans ce tp on va travailler avec IntelliJ Ultimate Edition

### Création de la couche Dao :

Créer l'entité JPA customer:

La première chose à faire est de créer l'entité à persister, c'est-à-dire la classe customer,

Alors on crée une classe nommée customer et on la modifie comme ci-dessous

```
import ...
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    @OneToMany(mappedBy = "customer")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<BankAccount> bankAccounts;
}
```

Et voilà notre entité est prêt à être mappé vers une table dans la bdd.

### Creation des repositories:

Créer l'interface CustomerRepository :

Alors puisque Spring Data a déjà créé des interfaces génériques et des implémentations

Génériques qui permettent de gérer les entités JPA, on n'aura plus besoin de faire appel à l'objet

EntityManager pour gérer la persistance. Spring Data le fait à notre place.

Il suffit de créer une interface qui hérite de l'interface JpaRepository pour hériter toutes les

Méthodes classiques qui permettent de gérer les entités JPA, de plus si on a besoin d'autre

méthodes nous avons la possibilité d'ajouter d'autres méthodes en les déclarant à l'intérieur de

L'interface JpaRepository, sans avoir besoin de les implémenter. Spring Data le fera à notre place.

Le résultat finale semble à la figure ci-dessous :

```
import ...

public interface CustomerRepository extends JpaRepository<Customer, Long> {
    @Query("select c from Customer c where c.name like :kw")
    List<Customer> searchCustomer(@Param("kw") String keyword);

    @Query("SELECT c FROM Customer c JOIN FETCH c.bankAccounts WHERE c.id = (:id)")
    Customer findByIdAndFetchBankAccountsEagerly(@Param("id") Long id);

    @EntityGraph(attributePaths = { "bankAccounts" })
    Customer getById(Long id);
}
```

Configurer l'unité de persistance : application.properties :

il suffit d'ajouter une ligne qui définit l'url de cette bdd dans le fichier application.properties comme ci-dessous :

```
application.properties
1 spring.datasource.url=jdbc:mysql://localhost:3306/E-BANK?createDatabaseIfNotExist=true
2 spring.datasource.username=root
3 spring.datasource.password=
4 spring.jpa.hibernate.ddl-auto = update
5 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MariaDBDialect
6 JavaCoder.app.jwtSecret= JavaCoderSecretKey
7 JavaCoder.app.jwtExpirationMs= 87400000
8 spring.devtools.restart.enabled=false
9
```

## Tester la couche DAO :

Nous modifions la classe EtudiantAppApplication pour avoir le résultat suivant :

Nous avons implémenté l'interface CommandLineRunner et réimplémenter la méthode Run, alors maintenant dès que notre Spring Container est prêt il va exécuter le code dedans la méthode run donc c'est dedans qu'on va tester notre couche dao.

On injecte une CustomerRepositories dans la classe et on commence à enregistrer et afficher les customers :

```

@Bean
CommandLineRunner commandLineRunner(BankAccountService bankAccountService){
    return args -> {
        Stream.of("Hassan", "Imane", "Mohamed").forEach(name->{
            CustomerDTO customer=new CustomerDTO();
            customer.setName(name);
            customer.setEmail(name+"@gmail.com");
            bankAccountService.saveCustomer(customer);
        });
        bankAccountService.listCustomers().forEach(customer->{
            try {
                bankAccountService.saveCurrentBankAccount
                    ( initialBalance: Math.random()*90000, overDraft: 9000, customer.getId());
                bankAccountService.saveSavingBankAccount
                    ( initialBalance: Math.random()*120000, interestRate: 5.5, customer.getId());
            } catch (CustomerNotFoundException e) {
                e.printStackTrace();
            }
        });
    };
}
    
```

Et de meme pour les autres entites comme **AccountOperation** et **BankAccount** ...

### Création de la couche Web :

La creation de la couche dao va être facilité avec l'utilisation de du module SPRING MVC, c'est spring qui gère la creation du servlet, et nous fournit toutes ses fonctionnalité sans complications et sans code technique, il suffit qu'on crée une classe et l'y ajouter l'annotation @Controller. Mais sous le capot Toutes les requêtes HTTP sont traitées par un contrôleur frontal fourni par Spring. C'est une servlet nommée DispatcherServlet, c'est cette servlet qui devrait executer une opération associée à chaque action. Ces opérations sont implémentées dans une classe appelée CustomerRestController qui représente un sous contrôleur ou un contrôleur secondaire.

```

@RestController
@AllArgsConstructor
@Slf4j
public class CustomerRestController {
    private BankAccountService bankAccountService;

    @GetMapping("/customers")
    @PreAuthorize("hasRole('ADMIN')")
    public List<CustomerDTO> customers() { return bankAccountService.listCustomers(); }

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

Créer l'application Web qui permet de chercher les comptes d'un customer

Afficher tous les opérations sur un compte:

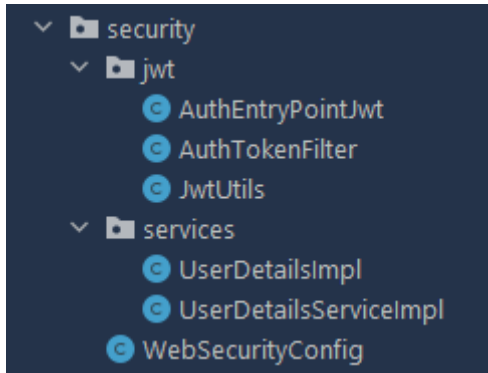
Pour afficher toutes les opérations, tout d'abord il faut créer le contrôleur secondaire et dedans on crée une méthode qui interroge la base de données, récupère la liste des opérations

```

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

La sécurité avec jwt :

Pour la sécurité on a utilisé la sécurité avec jwt (token)



Création de filtre :

```
package org.sid.ebankingbackend.security.jwt;

public class AuthTokenFilter extends OncePerRequestFilter {
    @Autowired
    private JwtUtils jwtUtils;

    @Autowired
    private UserDetailsService userDetailsService;

    private static final Logger logger = LoggerFactory.getLogger(AuthTokenFilter.class);

    @Override
    protected void doFilterInternal(
        HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {
        try {
            String jwt = parseJwt(request);
            if (jwt != null && jwtUtils.validateJwtToken(jwt)) {
                String username = jwtUtils.getUserNameFromJwtToken(jwt);

                UserDetails userDetails = userDetailsService.loadUserByUsername(username);
                UsernamePasswordAuthenticationToken authentication = new
                UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new
                WebAuthenticationDetailsSource().buildDetails(request));
            }
        } catch (Exception e) {
            logger.error("JWT Token is invalid: " + jwt);
        }
        filterChain.doFilter(request, response);
    }
}
```

```

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

filterChain.doFilter(request, response);
}

private String parseJwt(HttpServletRequest request) {
    String headerAuth = request.getHeader("Authorization");

    if (StringUtils.hasText(headerAuth) && headerAuth.startsWith("Bearer ")) {
        return headerAuth.substring(7, headerAuth.length());
    }

    return null;
}
}

```

Il faut aussi déterminer la clé de sécurité et le temp de d'expiration de token

```

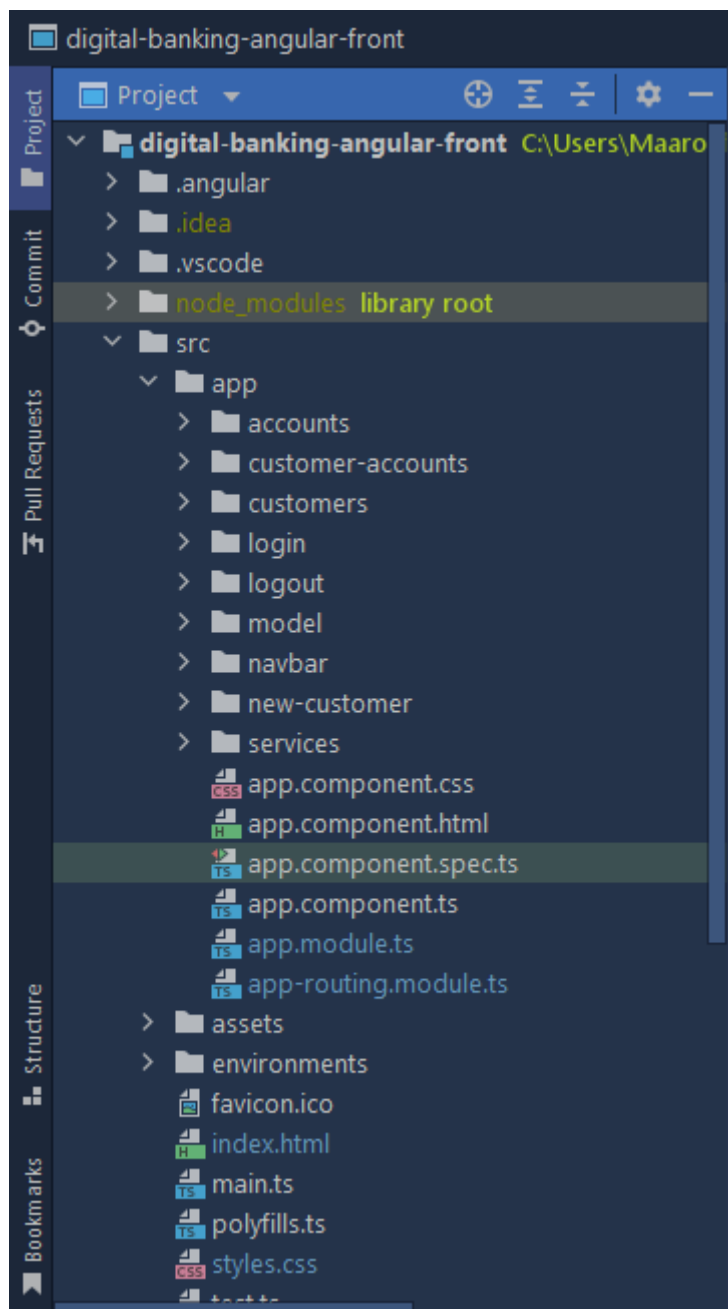
JavaCoder.app.jwtSecret= JavaCoderSecretKey
JavaCoder.app.jwtExpirationMs= 87400000

```

Maintenant on passe a créé le cote utilisateur frontend

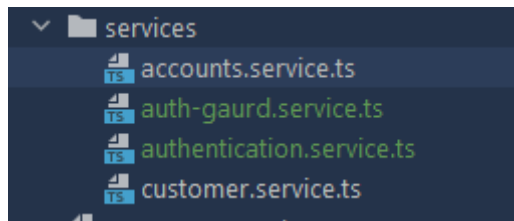
Partie frontend :

Structure de projet :





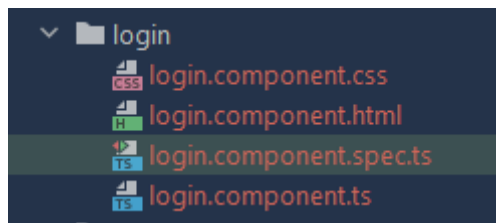
commence par la création de couche service qui va permet de faire des traitements des données reçu à partir de backend



On crée 4 fichiers 2 pour la sécurité et les autres pour manipuler les opérations sur les customers et les comptes

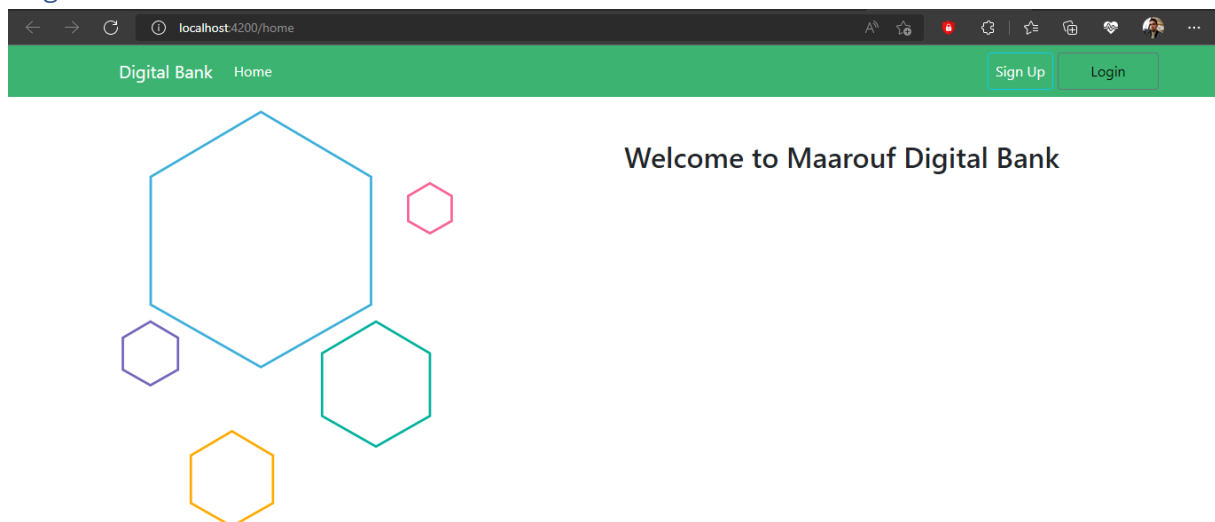
Après on passe à créer les différents composants qui vont afficher une partie déterminée dans la partie ui

Exemple login :



Exécution :

Page home :



Après on crée compte utilisateur normal :

Username

ouassime

Email

ouassime@gmail.com

Password

.....

Sign Up

Your registration is successful!

Maintenant on se connecter avec ce compte

Username

ouassime

Password

.....

Login

Logged in as ROLE\_USER.

La connexion et passe correctement le rôle par défaut est user

## Transfer

Digital Bank
Home
Accounts

Operations

☐ DEBIT:
☐ CREDIT:
☐ TRANSFER:

Amount :

0

Description :

Save Operation

Accounts

Account Id : 23a0803c-5375-4b27-b91f-8be92a7eb909

Account ID :23a0803c-5375-4b27-b91f-8be92a7eb909

Balance :702,948.71

ID	Date	Type	Amount
144	05-06-2022:12-34-12	CREDIT	39,112.60
145	05-06-2022:12-34-12	DEBIT	2,328.79
146	05-06-2022:12-34-12	CREDIT	99,675.03
147	05-06-2022:12-34-12	DEBIT	9,174.68
148	05-06-2022:12-34-12	CREDIT	70,440.62

0
1
2
3

On remarque qu'un user normal ne peut pas ajouter ou voir la liste des customers et pour cela on va se connecter avec un compte admin

Username

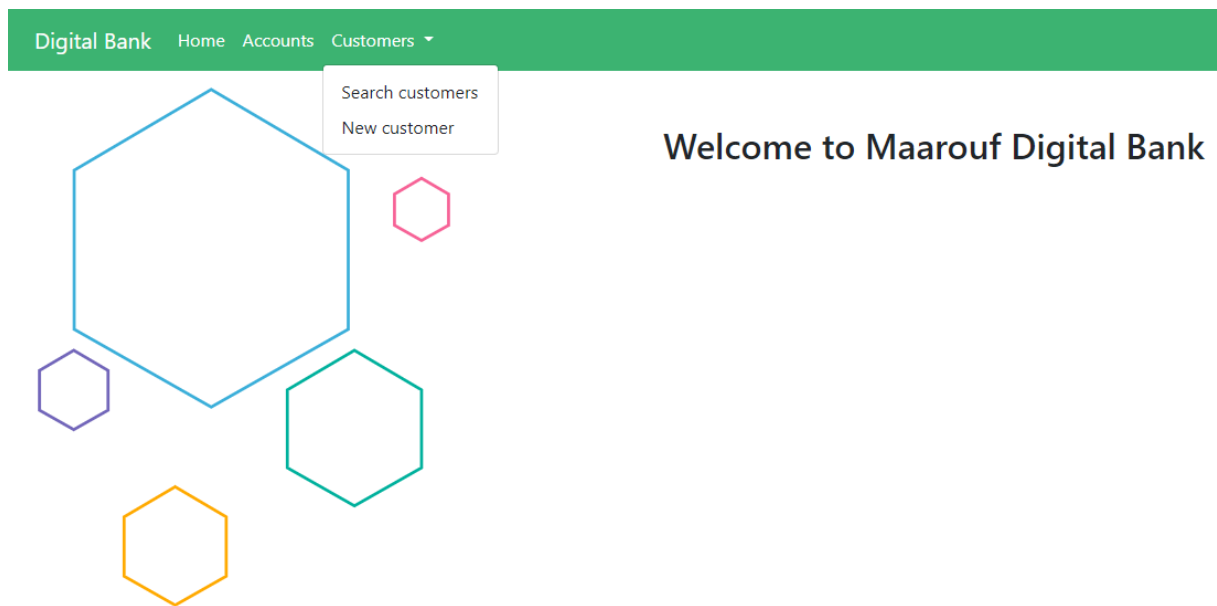
maarouf

Password

.....

Login

Logged in as ROLE\_ADMIN.



On remarque que pour un admin il peut aussi voir la liste des et ajouter un customer

Ajouter un customer









---

### New Customer

Name:

Email:

[Afficher la liste des customers](#)

Digital Bank Home Accounts Customers ▾				
Customers				
Keyword :				
ID	Name	Email		
1	Hassan	Hassan@gmail.com		Accounts
2	Imane	Imane@gmail.com		Accounts
3	Mohamed	Mohamed@gmail.com		Accounts
6	Hassan	Hassan@gmail.com		Accounts
7	Imane	Imane@gmail.com		Accounts
8	Mohamed	Mohamed@gmail.com		Accounts
9	Ouassime Maarouf	o.maarouf_etu@enset-media.ac.ma		Accounts

## Conclusion

A la fin de ce projet nous avons arriver a cree une application complet en utilisons lensemble des outiles quo na déjà cite , et on securiser notre application avec Spring Security et jwt qu est un outil genial qui facilite l'authentification et l'autorisation des utilisateurs, ainsi que la contextualisation des views de notre application.