

SOMMAIRE

Introduction	2
Énoncé	3
Travail à faire	4
1. Schéma de l'architecture technique de l'application	4
2. Diagramme de classe	5
3. Couche DAO	6
a. Entité JPA	6
b. Interface JPA JpaRepository	7
c. Test de la couche DAO	7
4. Couche Web	9
d. Gérer les clients	9
e. Gérer les abonnements	11
5. Web service RESTful	13
6. Sécurité	16
Conclusion	19



Le présent rapport concerne le développement d'une application de gestion des abonnements d'un opérateur télécom, basée sur Spring. Cette application permet de gérer les clients ainsi que leurs abonnements (type d'abonnement, solde, montant mensuel, etc.). Le développement de cette application a été réalisé en respectant une architecture technique basée sur un SGBD relationnel, Spring Data, JPA, Hibernate, Spring MVC avec Thymeleaf, et Spring Security.

Dans ce rapport, nous allons présenter les différentes étapes de développement de l'application, en commençant par une description de l'architecture technique. Nous allons ensuite détailler les différentes fonctionnalités implémentées, telles que la gestion des clients et des abonnements, l'affichage des abonnements d'un client, la mise à jour du solde de l'abonnement, etc. Enfin, nous allons conclure en présentant les résultats obtenus et les perspectives d'amélioration pour l'application.

On souhaite développer une application JEE basée sur Spring qui permet de gérer les abonnements d'un opérateur télécom. Chaque client peut avoir plusieurs abonnements.

- Un client est défini par : son id, son nom, son email et son username
- Un abonnement est défini par : son id, la date d'abonnement, le type d'abonnement (GSM, INTERNET, TELEPHONE_FIXE), son solde, et le montant mensuel

L'architecture de l'application est basée sur :

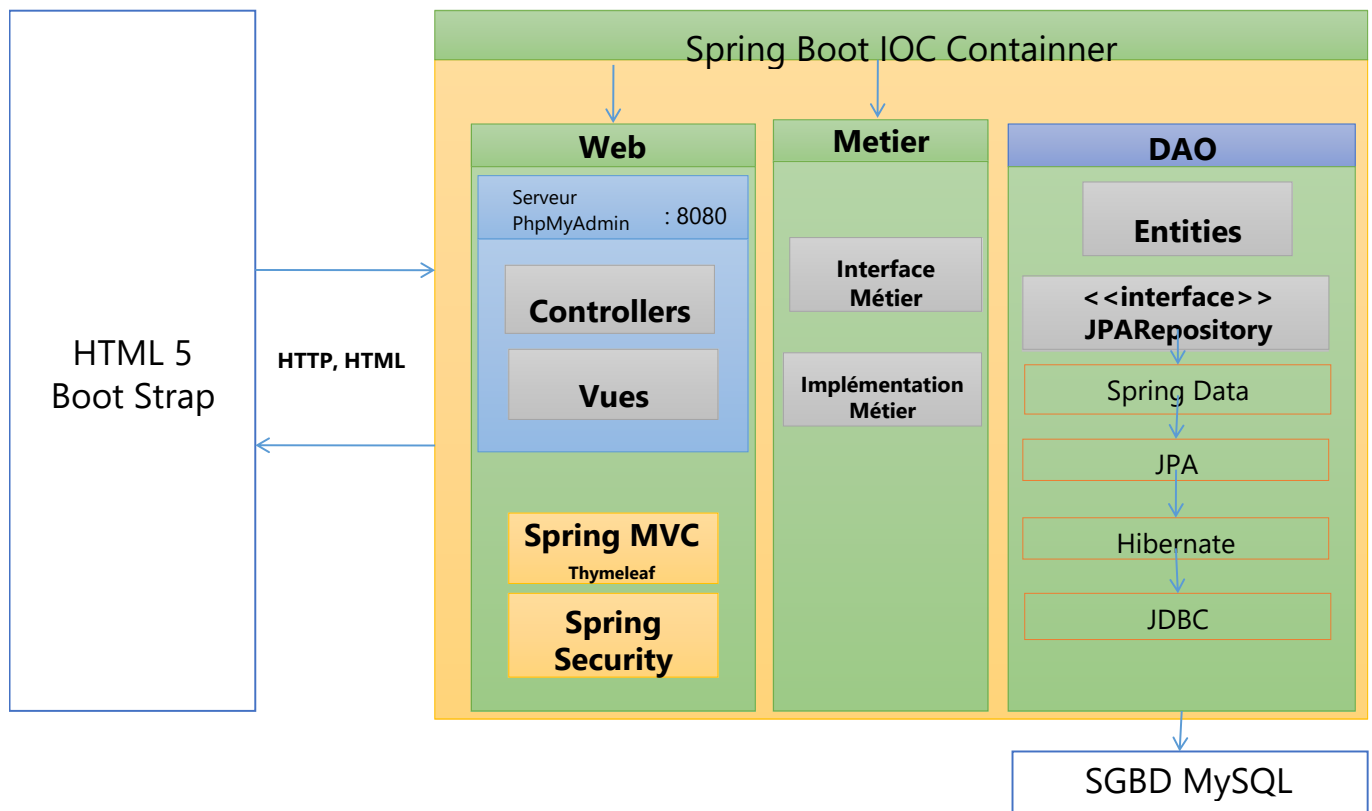
- Un SGBD relationnel de votre Choix (H2, MySQL, PostGres, etc..)
- Spring Data, JPA, Hibernate
- Spring MVC avec Thymeleaf
- Spring Security

Travail à faire:

Rendre le code source du projet et un rapport au format PDF contenant les réponses aux questions suivantes :

1. Schéma de l'architecture technique de l'application
2. Diagramme de classe représentant les données manipulées par l'application
3. Couche DAO :
 - a. Créer les entités JPA
 - b. Créer les interfaces JpaRepository basées sur Spring Data
 - c. Tester la couche DAO
4. Couche Web : Créer une applications Web qui permet de :
 - a. Gérer les clients (Chercher, Pagination, Ajout, Edition et Suppression)
 - b. Gérer les abonnements :
 - Afficher les abonnements d'un clients
 - Charger le solde de l'abonnement avec un montant
 - Autres opérations de gestion des abonnements
5. Créer un web service RESTful qui permet de gérer les clients et les abonnements
6. Sécurité : Sécuriser l'accès à l'application en développant un système d'authentification statefull basé sur Spring Security avec deux rôles CLIENT et ADMIN. l'application doit répondre aux critères suivants :
 - a. Authentification avec le rôle CLIENT : le client ne peut voir que : son profile, ses abonnements et peut charger ses abonnements
 - b. Authentification avec le rôle ADMIN : l'administrateur peut gérer les clients et les abonnements avec tous les droits possible. En plus il peut créer de nouveau utilisateurs et affecter des rôles aux utilisateurs.

1. Schéma de l'architecture technique de l'application



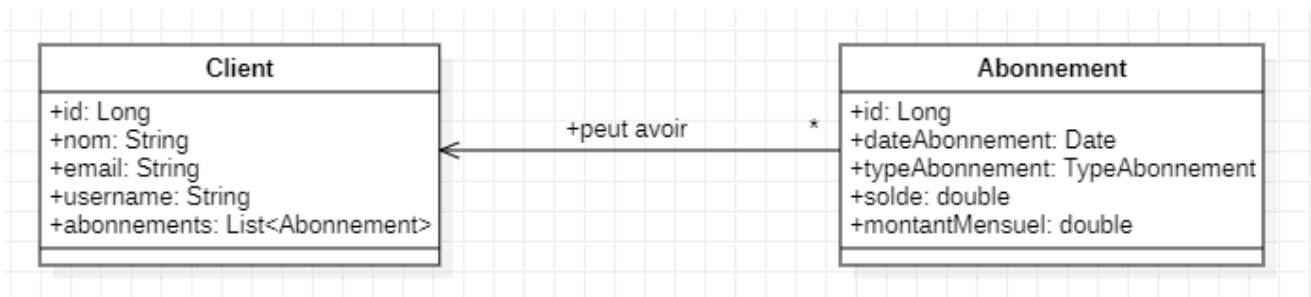
La vue est responsable de l'interface utilisateur de l'application, qui est créée à l'aide de Thymeleaf, HTML, CSS et JavaScript. Le contrôleur gère les requêtes HTTP provenant de la vue et les achemine vers la couche de service appropriée.

La couche de service est responsable de la logique métier de l'application et utilise Spring Data, JPA et Hibernate pour interagir avec la couche d'accès aux données. Cette couche peut également inclure des annotations `@Transactional` pour gérer les transactions de la base de données.

La couche d'accès aux données est responsable de la lecture et de l'écriture des données dans la base de données. Elle utilise JDBC, Hibernate, JPA et Spring Data pour interagir avec le SGBDR.

Le SGBDR est le système de gestion de base de données relationnel qui stocke les données de l'application

2. Diagramme de classe



La classe Client contient des attributs pour stocker l'identifiant, le nom, l'e-mail et le nom d'utilisateur d'un client. La classe Abonnement contient des attributs pour stocker l'identifiant, la date d'abonnement, le type d'abonnement (qui est une énumération de type AbonnementType), le solde et le montant mensuel.

La classe AbonnementType est une énumération qui contient les différents types d'abonnements disponibles : GSM, INTERNET et TELEPHONE_FIXE.

3. Couche DAO

a. Entité JPA

Client.java

```
@Entity
@Table(name = "clients")
@Data @AllArgsConstructor @NoArgsConstructor
public class Client {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    // @NotEmpty //Validation
    // @Size(min = 3,max = 20) //Validation
    private String nom;
    // @NotEmpty
    // @Size(min = 10,max = 40)
    private String email;
    // @NotEmpty
    // @Size(min = 3,max = 15)
    private String username;
    @OneToMany(mappedBy="client")
    private Collection<Abonnement> abonnements;
}
```

TypeAbonnement.java

```
package com.etoullali.enums;

public enum TypeAbonnement {
    GSM, INTERNET, TELEPHONE_FIXE
}
```

Abonnement.java

```
@Entity
@Table(name = "Abonnements")
@Data @AllArgsConstructor @NoArgsConstructor
public class Abonnement {

    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd") //pour unifier le format (ex) remplir
    formulaire
    private Date dateAbonnement;
    private TypeAbonnement typeAbonnement;
    private double solde;
    private double montantMensuel;
    @ManyToOne
    @JoinColumn(name = "client_id")
    private Client client;
}
```

b. Interface JPA JpaRepository

ClientRepository.java

```
public interface ClientRepository extends JpaRepository<Client, Long> {  
  
    Page<Client> findByNomContains(String keyword , Pageable pageable);  
    Optional<Client> findByUsername(String username2);  
    List<Client> findByNom(String nom1);  
}
```

AbonnementRepository.java

```
public interface AbonnementRepository extends JpaRepository<Abonnement, Long> {  
    Collection<Abonnement> findByClientId(Long clientId);  
    Collection<Abonnement> findByClient(Client client);  
}
```

c. Test de la couche DAO

Application.java

```
@SpringBootApplication  
public class TestApplication {  
    public static void main(String[] args) {  
        ApplicationContext cxt = SpringApplication.run(TestApplication.class, args);  
        ClientRepository clientRepository = cxt.getBean(ClientRepository.class);  
        AbonnementRepository abonnementRepository =  
cxt.getBean(AbonnementRepository.class);  
  
        List<Abonnement> listAbonnement1=new ArrayList<>();  
  
        Abonnement a1=new Abonnement(),a2=new Abonnement();  
        a1.setSolde(Math.random() * 100);  
        a1.setDateAbonnement(new Date());  
        a1.setTypeAbonnement(TypeAbonnement.GSM);  
        a1.setMontantMensuel(Math.random() * 100);  
        a2.setSolde(Math.random() * 100);  
        a2.setDateAbonnement(new Date());  
        a2.setTypeAbonnement(TypeAbonnement.TELEPHONE_FIXE);  
        a2.setMontantMensuel(Math.random() * 100);  
        listAbonnement1.add(a1);  
        listAbonnement1.add(a2);  
  
        Stream.of("Ayoub","Samira", "Ihssan", "Radouan", "Ayoub", "Mustapha",  
"Hayat")  
            .forEach(name -> {  
                Client c1 = new Client();  
                c1.setNom(name);  
                c1.setEmail(name+"@gmail.com");  
                c1.setUsername(name);  
                c1.setAbonnements(listAbonnement1);  
                clientRepository.save(c1);  
            });  
        Client c1 = new Client();  
        c1.setNom("Hassan");  
        c1.setEmail("Hassan@gmail.com");  
        c1.setUsername("Hassan");  
        c1.setAbonnements(listAbonnement1);  
        clientRepository.save(c1);  
  
        Client c2 = new Client();  
        c2.setNom("Salim");  
        c2.setEmail("Salim@gmail.com");  
        c2.setUsername("Salim");
```



```

        c2.setAbonnements(listAbonnement1);
        clientRepository.save(c2);

        a1.setClient(c1);
        a2.setClient(c2);

        abonnementRepository.save(a1);
        abonnementRepository.save(a2);

        a1.setClient(c1);
        clientRepository.findAll().forEach(c->{
            System.out.println(c.getNom());
        });

        System.out.println("All Clients : ");
        clientRepository.findAll().forEach(System.out::println);
    }
}

```

Résultat

```

Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into clients (email, nom, username) values (?, ?, ?)
Hibernate: insert into abonnements (client_id, date_abonnement, montant_mensuel, solde, type_abonnement) values (?, ?, ?, ?, ?)
Hibernate: insert into abonnements (client_id, date_abonnement, montant_mensuel, solde, type_abonnement) values (?, ?, ?, ?, ?)

```

*** liste des Clients ***

```

Hibernate: select c1_0.id,c1_0.email,c1_0.nom,c1_0.username from clients c1_0

```

```

Ayoub
Samira
Ihssan
Radouan
Ayoub
Mustapha
Hayat
Hassan
Salim


```

4. Couche Web

d. Gérer les clients

Chercher

```
<form class="d-flex" method="get" th:action="${index}">
  <label>Key Word</label>
  <input class="form-control me-2" type="text" name="keyword" th:value="${keyword}"
placeholder="Search">
  <!-- value : par défaut vide -->
  <button type="submit" class="btn btn-primary">Chercher</button>
</form>
```

Chercher

Pagination

```
<ul class="nav nav-pills">
  <li th:each="page,status:${pages}">

    <a
      th:class="${status.index==currentPage? 'btn btn-warning m-2':'btn
btn-outline-info m-2'}"
      th:text="${status.index}"
      th:href="@{/client/index(page=${status.index},keyword=${keyword})}"
    ></a>

  </li>
</ul>
```



Ajout

```
@PostMapping(path = "/admin/saveClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String saveClient(Model model, @Valid Client client, BindingResult
bindingResult, //BindingResult : collection des erreurs
    @RequestParam(defaultValue = "") String keyword,
    @RequestParam(defaultValue = "0") int page) {
  if (bindingResult.hasErrors()) return "formClients";
  clientRepository.save(client);
  return "redirect:/client/index?page" + page + "&keyword" + keyword;
}
```

Nom

Email

Save

Edition

```
@GetMapping(path = "/admin/editClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String editClient(Model model, Long id, String keyword, int page) {
    Client client = clientRepository.findById(id).orElse(null);
    if (client == null) throw new RuntimeException("Client not found");
    model.addAttribute("client", client);
    model.addAttribute("page", page);
    model.addAttribute("keyword", keyword);
    return "editClient";
}
```

Id : 5

Nom :

Ayoub


Email :

Ayoub@gmail.com

Save

Suppression

```
@GetMapping("/admin/deleteClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteClient(Long id, int page, String keyword) { //par défaut
    @RequestParam (conserve name)
    clientRepository.deleteById(id);
    return "redirect:/client/index?page=" + page + "&keyword=" + keyword;
}
```

[Home](#) [Clients](#)  localhost:8080 indique sur ? admin ▾

Liste des clients

ID	Nom	Email	UserName	Les abonnements		
2	Samira	Samira@gmail.com	Samira	<input type="button" value="Voir"/>	<input type="button" value="Modifier"/>	<input type="button" value="Supprimer"/>
3	Ihssan	Ihssan@gmail.com	Ihssan	<input type="button" value="Voir"/>	<input type="button" value="Modifier"/>	<input type="button" value="Supprimer"/>
4	Radouan	Radouan@gmail.com	Radouan	<input type="button" value="Voir"/>	<input type="button" value="Modifier"/>	<input type="button" value="Supprimer"/>
5	Ayoub	Ayoub@gmail.com	Ayoub	<input type="button" value="Voir"/>	<input type="button" value="Modifier"/>	<input type="button" value="Supprimer"/>
6	Mustapha	Mustapha@gmail.com	Mustapha	<input type="button" value="Voir"/>	<input type="button" value="Modifier"/>	<input type="button" value="Supprimer"/>


e. Gérer les abonnements

Afficher les abonnements d'un client

```
@GetMapping("/admin/AbonnementClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String AbonnementClient(Model model, Long id) {
    Client client = clientRepository.findById(id).orElse(null);

    Collection<Abonnement> abonnement = abonnementRepository.findByClient(client);

    model.addAttribute("client", client);
    model.addAttribute("abonnement", abonnement);
    return "AbonnementClient_ADMIN";
}
```


[Home](#) [Clients](#)  [Chercher](#) admin

La liste des Abonnements de client : Salim

ID	DateAbonnement	MontantMensuel	TypeAbonnement	Solde	Abonnement	Action
2	2023-04-26	15.02351839371724	TELEPHONE_FIXE	85.0824745681596	Charger	Modifier Supprimer


Charger le solde de l'abonnement avec un montant

```
@PostMapping("/client/rechargerSolde")
public String rechargerSolde(@PathVariable Long abonnementId,
@RequestParam("montant") double montant) {
    abonnementService.rechargerSolde(abonnementId, montant);
    return "AbonnementClient_ADMIN";
}
```

[Home](#) [Clients](#)  [Chercher](#) admin

La liste des Abonnements de client : Salim

ID	DateAbonnement	MontantMensuel	TypeAbonnement	Solde	Abonnement	Action
4	2023-04-26	62.540489630733006	TELEPHONE_FIXE	18.293852014145063	Charger	Modifier Supprimer


[Home](#) [Clients](#)  [Chercher](#) ayoub

La liste de votre Abonnement

ID	Date Abonnement	Montant Mensuel	Type Abonnement	Solde	Abonnement
1	12/12/2013	1222	9888	88899	Charger

Autres opérations de gestion des abonnements

```
@GetMapping("/admin/deleteAbonnement")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteAbonnement(Long id) { //par défaut @RequestParam (conserve name)
    abonnementRepository.deleteById(id);
    return "redirect:/admin/AbonnementClient?id="+id;
}
```

[Home](#) [Clients](#)  localhost:8080 indique sur ? [Chercher](#) admin

La liste des Abonnements de client : Salim

ID	DateAbonnement	MontantMensuel	TypeAbonnement	Solde	Abonnement	Action
2	2023-04-26	15.02351839371724	TELEPHONE_FIXE	85.0824745681596	Charger	Modifier Supprimer

5. Web service RESTful

ClientController

```
@Controller
//=> 1ème Solution : vs @RestController qui comprend que la retour doit l'affecter à
dans le body de la réponse (la vue)
@AllArgsConstructor
public class ClientController {
    public ClientRepository clientRepository;
    public AbonnementRepository abonnementRepository;

    @GetMapping("/")
    public String home() {
        if
        (SecurityContextHolder.getContext().getAuthentication().getAuthorities().stream()
            .anyMatch(authority ->
                authority.getAuthority().equals("ROLE_ADMIN"))) {
            return "redirect:/client/index";
        }

        if
        (SecurityContextHolder.getContext().getAuthentication().getAuthorities().stream()
            .anyMatch(authority ->
                authority.getAuthority().equals("ROLE_CLIENT"))) {
            return "AbonnementClient_CLIENT";
        }

        return null;
    }

    //Liste des clients + Abonnements : ADMIN & CLIENT
    @GetMapping(path = "/client/index")
    public String clients(Model model,
        @RequestParam(name = "page", defaultValue = "0") int page,
        @RequestParam(name = "size", defaultValue = "5") int size,
        @RequestParam(name = "keyword", defaultValue = "") String
        keyword) {
        if
        (SecurityContextHolder.getContext().getAuthentication().getAuthorities().stream().any
        Match(
            authority -> authority.getAuthority().equals("ROLE_ADMIN"))) {
            Page<Client> clients = clientRepository.findByNomContains(keyword,
            PageRequest.of(page, size));
            model.addAttribute("listClients", clients.getContent());
            model.addAttribute("pages", new int[clients.getTotalPages()]);
            model.addAttribute("currentPage", page);
            model.addAttribute("keyword", keyword);

            return "clients";
        }

        if
        (SecurityContextHolder.getContext().getAuthentication().getAuthorities().stream().any
        Match(
            authority -> authority.getAuthority().equals("ROLE_CLIENT"))) {
            return "AbonnementClient_CLIENT";
        }

        return null;
    }
}
```

```

//formulaire client => Ajouter client
@GetMapping("/admin/formClients")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String formClients(Model model) {
    model.addAttribute("client", new Client()); //des valeurs par défaut
    return "formClients";
}

//Ajouter client
@PostMapping(path = "/admin/saveClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String saveClient(Model model, @Valid Client client, BindingResult
bindingResult, //BindingResult : collection des erreurs
    @RequestParam(defaultValue = "") String keyword,
    @RequestParam(defaultValue = "0") int page) {
    if (bindingResult.hasErrors()) return "formClients";
    clientRepository.save(client);
    return "redirect:/client/index?page" + page + "&keyword" + keyword;
}

//Editer un client
@GetMapping(path = "/admin/editClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String editClient(Model model, Long id, String keyword, int page) {
    Client client = clientRepository.findById(id).orElse(null);
    if (client == null) throw new RuntimeException("Client not found");
    model.addAttribute("client", client);
    model.addAttribute("page", page);
    model.addAttribute("keyword", keyword);
    return "editClient";
}

//Supprimer un client
@GetMapping("/admin/deleteClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteClient(Long id, int page, String keyword) { //par défaut
@RequestParam (conserve name)
    clientRepository.deleteById(id);
    return "redirect:/client/index?page" + page + "&keyword=" + keyword;
}

//Abonnement de cClient
@GetMapping("/admin/AbonnementClient")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String AbonnementClient(Model model, Long id) {
    Client client = clientRepository.findById(id).orElse(null);

    Collection<Abonnement> abonnement =
abonnementRepository.findByClient(client);

    model.addAttribute("client", client);
    model.addAttribute("abonnement", abonnement);
    return "AbonnementClient_ADMIN";
}

//Recharger Solde
/*@PostMapping("/client/rechargerSolde")
public String rechargerSolde(@PathVariable Long abonnementId,
@RequestParam("montant") double montant) {
    abonnementService.rechargerSolde(abonnementId, montant);
    return "AbonnementClient_ADMIN";
}*/

//Supprimer un abonnement
@GetMapping("/admin/deleteAbonnement")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteAbonnement(Long id) { //par défaut @RequestParam (conserve

```

```
name)
    abonnementRepository.deleteById(id);
    return "redirect:/admin/AbonnementClient?id="+id;
}
}
```


6. Sécurité

Sécuriser l'accès à l'application en développant un système d'authentification statefull basé sur Spring Security avec deux rôles CLIENT et ADMIN.

a. **Authentification avec le rôle CLIENT** : le client ne peut voir que : son profile, ses abonnements et peut charger ses abonnements

b. **Authentification avec le rôle ADMIN** : l'administrateur peut gérer les clients et les abonnements avec tous les droits possibles. En plus il peut créer de nouveau utilisateurs et affecter des rôles aux utilisateurs.

SecurityConfig.java

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true) // [M2] sécurité coté serveur mais
n'oublie pas @PreAuthorize dans le controller
public class SecurityConfig {

    @Bean
    public InMemoryUserDetailsManager inMemoryUserDetailsManager() {

        return new InMemoryUserDetailsManager(

            User.withUsername("ayoub").password(passwordEncoder().encode("ayoub")).roles("CLIENT")
            ).build(), //d'une manière simple on utilise "{noop}1234" | <noop> cad pas utiliser
            encoder (hash)

            User.withUsername("rado").password(passwordEncoder().encode("rado")).roles("CLIENT").
            build(),

            User.withUsername("admin").password(passwordEncoder().encode("admin")).roles("CLIENT"
            ,"ADMIN").build()
            );
    }

    @Bean
    PasswordEncoder passwordEncoder(){ //password encoder
        return new BCryptPasswordEncoder(); // nouveau algo très puissant peut faire
        le hashage de mot de passe
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws
    Exception {
        httpSecurity.formLogin().loginPage("/login").permitAll(); //formulaire d'auth
        httpSecurity.rememberMe();

        httpSecurity.authorizeHttpRequests().requestMatchers("/webjars/**").permitAll();

        // [M1] sécurité coté serveur

        httpSecurity.authorizeHttpRequests().requestMatchers("/client/**").hasRole("CLIENT");

        httpSecurity.authorizeHttpRequests().requestMatchers("/admin/**").hasRole("ADMIN");

        httpSecurity.authorizeHttpRequests().anyRequest().authenticated();
        httpSecurity.exceptionHandling().accessDeniedPage("/notAuthorized");
        return httpSecurity.build();
    }
}
```

SecurityController.java

```
@Controller
public class SecurityController {

    @GetMapping("/notAuthorized")
    public String notAuthorized() {
        return "notAuthorized";
    }

    @GetMapping("/login")
    public String login() {
        return "login";
    }
}
```

login.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Authentification</title>
    <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
    <link rel="stylesheet" href="/webjars/bootstrap-icons/1.10.3/font/bootstrap-
icons.css">
</head>
<body>
<div class="row mt-3">
    <div class="col-md-6 offset-3">
        <div class="card">
            <div class="card-header">Authentification</div>
            <div class="card-body">
                <form method="post" th:action="@{/login}">
                    <div class="mb-3 mt-3">
                        <label for="username" class="form-label">Username</label>
                        <input id="username" class="form-control" type="text"
name="username" placeholder="username">
                    </div>
                    <div class="mb-3 mt-3">
                        <label for="password" class="form-label">Password</label>
                        <input id="password" class="form-control" type="password"
name="password" placeholder="password">
                    </div>
                    <div>
                        <label class="form-check mb-3">
                            <input class="form-check-input" type="checkbox"
name="remember-me"> Remember me
                        </label>
                    </div>
                    <button type="submit" class="btn btn-primary">Login</button>
                </form>
            </div>
        </div>
    </div>
</div>
</body>
</html>
```

Authentication

Username

ayoub

Password

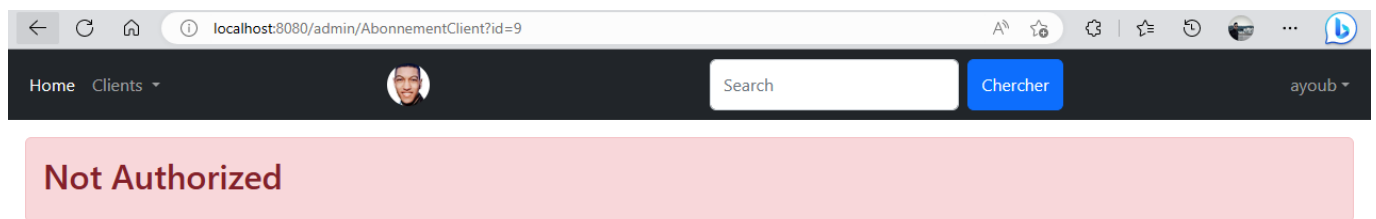
....

☐ Remember me

Login

notAuthorized.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
  xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
  layout:decorate="layout">
<head>
  <meta charset="UTF-8">
  <title>Not Authorized</title>
  <link rel="stylesheet" href="/webjars/bootstrap/5.2.3/css/bootstrap.min.css">
</head>
<body>
  <div layout:fragment="content1">
    <div class="alert alert-danger m-3">
      <h2 class="align-content-center"> Not Authorized </h2>
    </div>
  </div>
</body>
</html>
```



En conclusion, le développement de l'application de gestion des abonnements d'un opérateur télécom basée sur Spring a permis de mettre en pratique les différents concepts appris dans le cadre de notre formation en développement web. Cette application est fonctionnelle et permet de gérer efficacement les clients ainsi que leurs abonnements.

Toutefois, il reste des perspectives d'amélioration pour cette application, telles que l'ajout de fonctionnalités supplémentaires pour les clients et les abonnements, l'amélioration de l'interface utilisateur, la mise en place de tests automatisés, etc.

En somme, ce projet a été très enrichissant pour nous, car il nous a permis de consolider nos connaissances en développement web et en particulier en développement d'applications Spring. Nous espérons que ce rapport permettra de mieux comprendre les différentes étapes de développement de cette application et de ses fonctionnalités.