

Architecture Distribuée et Middlewares

Contrôle Architecture JEE et Middleware

Rapport

Année Universitaire
2022/2023
GLSID2

Présenté par
AKASMIOU Ouassima

Encadré par
M. YOUSSEFI Mohammed



Enoncé

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

Schéma de l'architecture technique de l'application

L'application JEE basée sur Spring pour la gestion des abonnements d'un opérateur télécom est structurée en couches. Les couches sont les suivantes :

Couche de présentation : Cette couche est responsable de la présentation des données à l'utilisateur final. Elle utilise Spring MVC avec Thymeleaf pour créer des vues dynamiques qui sont renvoyées à l'utilisateur via le navigateur web.

Couche de contrôleur : Cette couche est responsable de la gestion des demandes de l'utilisateur. Elle utilise les contrôleurs Spring MVC pour interagir avec les vues et les services de l'application.

Couche de service : Cette couche est responsable de la logique métier de l'application. Elle utilise les services Spring pour gérer les opérations de l'application telles que la gestion des abonnements et des clients.

Couche d'accès aux données : Cette couche est responsable de la gestion de la persistance des données. Elle utilise Spring Data, JPA, Hibernate pour interagir avec le SGBD relationnel MySQL. Les entités du domaine de l'application (abonnement, client) sont mappées sur des tables dans la base de données.

Couche de sécurité : Cette couche est responsable de la sécurité de l'application. Elle utilise Spring Security pour gérer l'authentification et l'autorisation des utilisateurs.

Chaque couche est indépendante et peut être testée séparément, ce qui facilite la maintenance et l'évolutivité de l'application.

Le schéma de l'architecture technique de l'application pourrait ressembler à ceci :

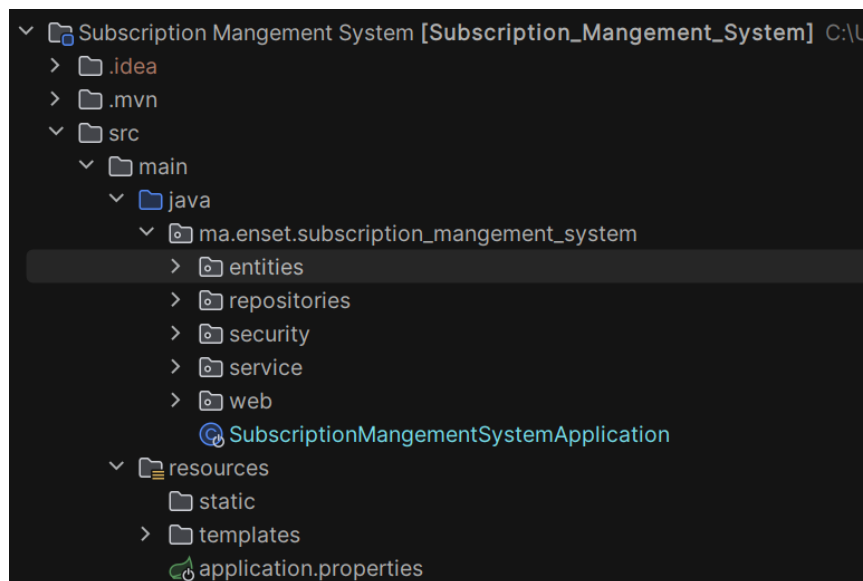
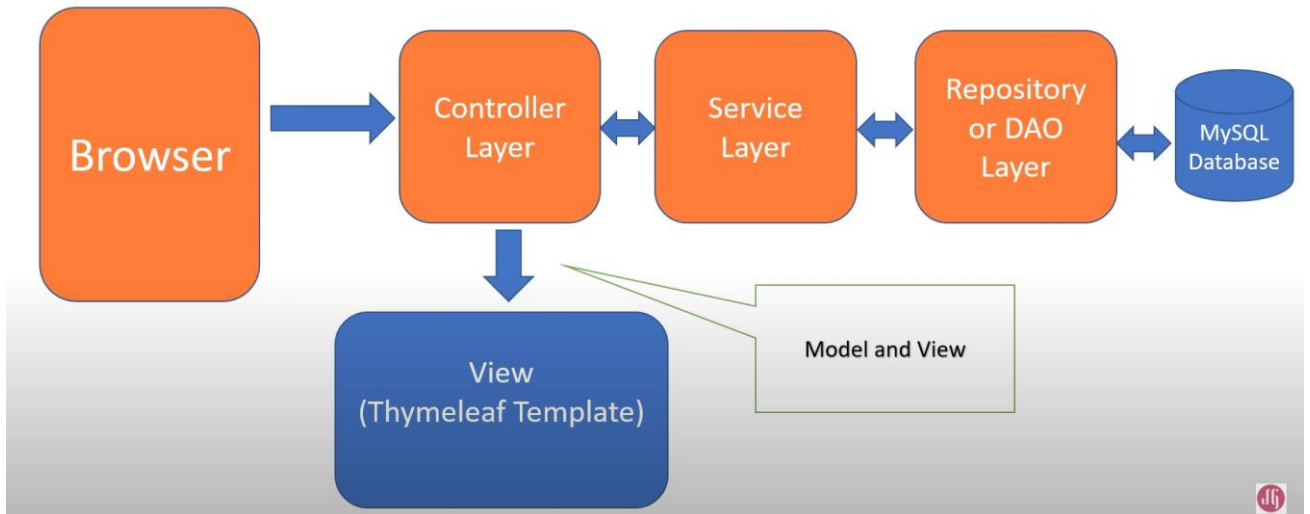
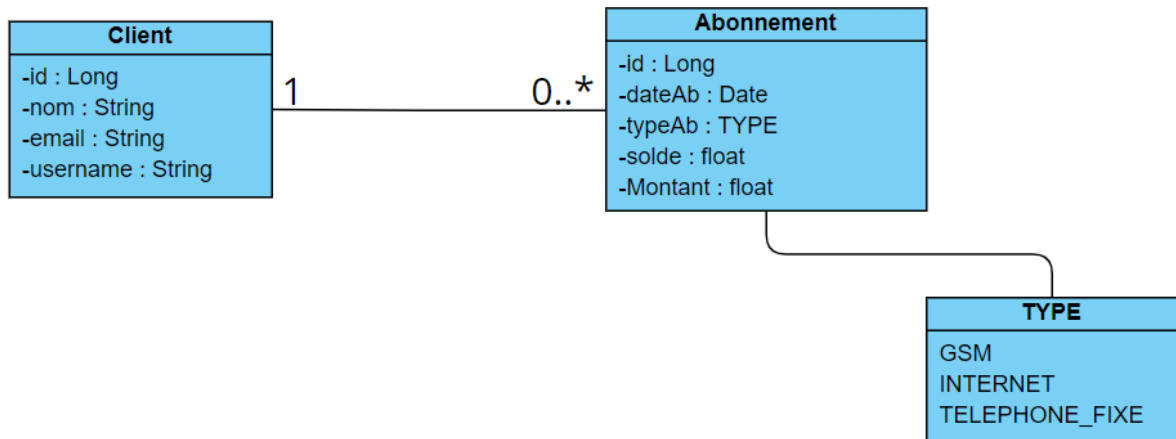
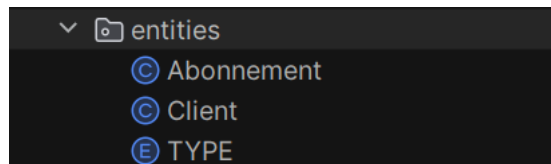


Diagramme de classe représentant les données manipulées par l'application



Couche DAO

- Les entités JPA



✓ Entité Abonnement :

```

@Entity @Data @NoArgsConstructor @AllArgsConstructor
@Builder @ToString
public class Abonnement {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @Temporal(TemporalType.DATE)
    @DateTimeFormat(pattern = "yyyy-MM-dd")
    private Date dateAb;
    no usages
    @Enumerated(EnumType.STRING)
    private TYPE typeAb;
    no usages
    private Float solde;
    no usages
    private Float montant;
    no usages
    @ManyToOne
    private Client client;
}
  
```

```

public enum TYPE {
    1 usage
    GSM,
    1 usage
    INTERNET,
    no usages
    TELEPHONE_FIXE
}
  
```

✓ Entité Client :

```
@Entity @Data @NoArgsConstructor @AllArgsConstructor
@Builder
public class Client {
    no usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    @NotEmpty @Size(min = 3,message = "client name should have at least 3 characters")
    private String nom;
    no usages
    @Email
    @Size(max = 50)
    private String email;
    no usages
    @NotEmpty @Size(min = 4,message = "client name should have at least 4 characters")
    private String username;
    no usages
    @OneToMany(mappedBy = "client", fetch = FetchType.LAZY)
    @JsonProperty(access = JsonProperty.Access.READ_WRITE)
    private Collection<Abonnement> abonnements;
}
```

Les interfaces JpaRepository basées sur Spring Data

```
▼ repositories
  ⓘ AbonnementRepository
  ⓘ ClientRepository
```

```
@Repository
public interface AbonnementRepository extends JpaRepository<Abonnement,Long> {
    1 usage   ⓘ AKASMIOU Ouassima
    List<Abonnement> findByClient_Id(Long clientId);
}
```

```
@Repository
public interface ClientRepository extends JpaRepository<Client,Long> {
    1 usage   ⓘ AKASMIOU Ouassima
    Page<Client> findByNomContains(String kw, Pageable pageable);
    no usages ⓘ AKASMIOU Ouassima
    @Query("Select c from Client c where c.nom like :x ")
    Client chercherClient(@Param("x") String nom);
}
```

Test de la couche DAO

La classe `SubscriptionManagementSystemApplication` est une classe principale d'une application Spring Boot avec une méthode `main` qui exécute l'application. Il contient également une méthode `Bean CommandLineRunner` qui s'exécute après le démarrage de l'application et qui ajoute des données initiales à la base de données en utilisant les classes `Client` et `Abonnement` et leurs référentiels correspondants.

```
@SpringBootApplication
public class SubscriptionManagementSystemApplication {

    no usages  AKASMIOU Ouassima
    public static void main(String[] args) {
        SpringApplication.run(SubscriptionManagementSystemApplication.class, args);
    }

    no usages  AKASMIOU Ouassima *
    @Bean
    CommandLineRunner start(ClientRepository clientRepository, AbonnementRepository abonnementRepository){
        return args -> {
            Stream.of(...values: "ouassima", "Hanane", "Mohamed").forEach(name ->{
                Client client=new Client();
                client.setNom(name);
                client.setEmail(name+"@gmail.com");
                client.setUsername(Math.random()>0.5?"admin":"client");
                clientRepository.save(client);
            });

            Client client1=clientRepository.findById(1L).orElse( other: null);
            Client client2=clientRepository.findById(2L).orElse( other: null);

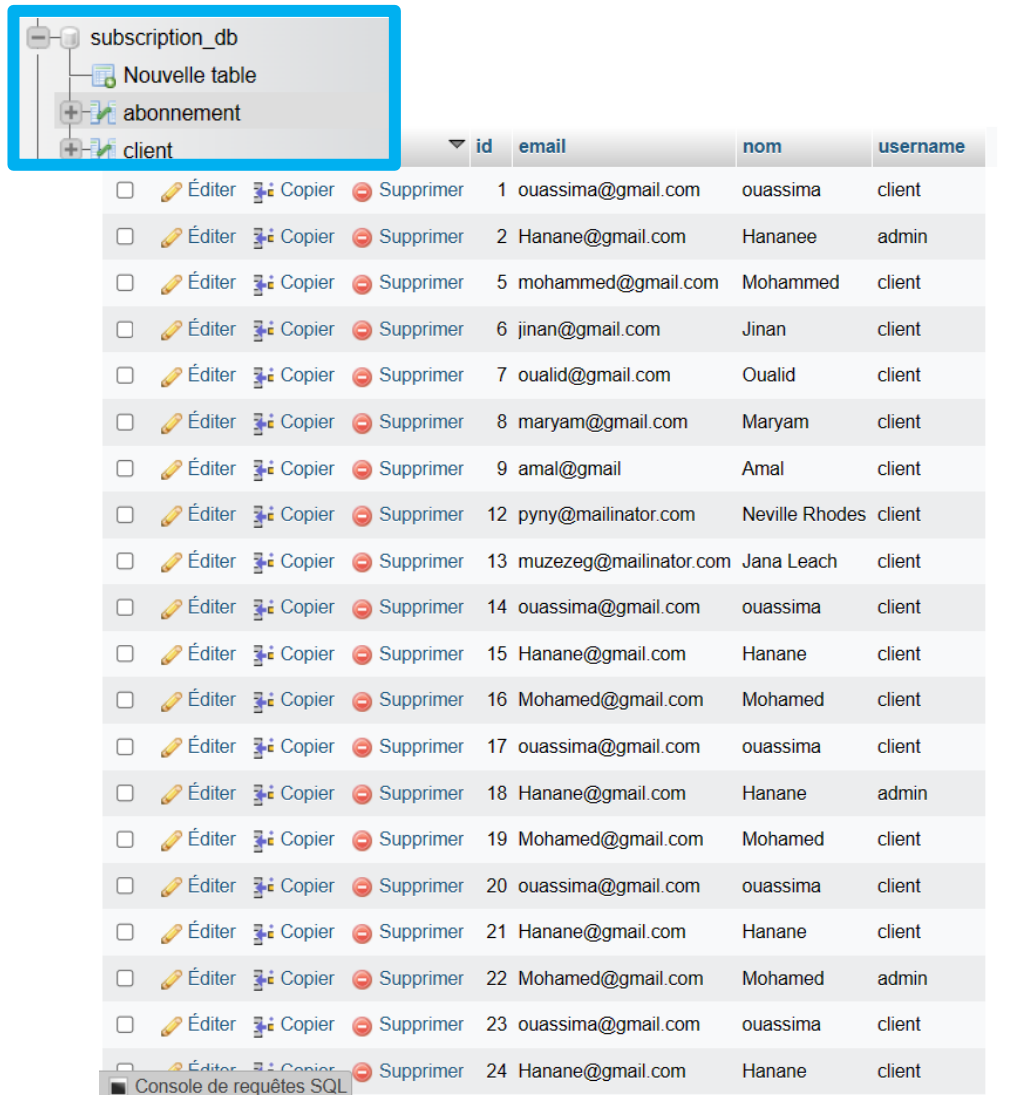
            Client client1=clientRepository.findById(1L).orElse( other: null);
            Client client2=clientRepository.findById(2L).orElse( other: null);

            Abonnement abonnement1=new Abonnement();
            abonnement1.setDateAb(new Date());
            abonnement1.setTypeAb(TYPE.GSM);
            abonnement1.setSolde(1200F);
            abonnement1.setMontant(2900F);
            abonnement1.setClient(client1);
            abonnementRepository.save(abonnement1);

            Abonnement abonnement2=new Abonnement();
            abonnement2.setDateAb(new Date());
            abonnement2.setTypeAb(TYPE.INTERNET);
            abonnement2.setSolde(5000F);
            abonnement2.setMontant(100F);
            abonnement2.setClient(client2);
            abonnementRepository.save(abonnement2);
        };
    }
}
```





























































• Dans phpMyAdmin

Maintenant nous allons vérifier que les données ont bien été ajoutées à la base de données.



	id	email	nom	username
<input type="checkbox"/> Éditer Copier Supprimer	1	ouassima@gmail.com	ouassima	client
<input type="checkbox"/> Éditer Copier Supprimer	2	Hanane@gmail.com	Hanane	admin
<input type="checkbox"/> Éditer Copier Supprimer	5	mohammed@gmail.com	Mohammed	client
<input type="checkbox"/> Éditer Copier Supprimer	6	jinan@gmail.com	Jinan	client
<input type="checkbox"/> Éditer Copier Supprimer	7	oualid@gmail.com	Oualid	client
<input type="checkbox"/> Éditer Copier Supprimer	8	maryam@gmail.com	Maryam	client
<input type="checkbox"/> Éditer Copier Supprimer	9	amal@gmail	Amal	client
<input type="checkbox"/> Éditer Copier Supprimer	12	pyny@mailinator.com	Neville Rhodes	client
<input type="checkbox"/> Éditer Copier Supprimer	13	muzezeg@mailinator.com	Jana Leach	client
<input type="checkbox"/> Éditer Copier Supprimer	14	ouassima@gmail.com	ouassima	client
<input type="checkbox"/> Éditer Copier Supprimer	15	Hanane@gmail.com	Hanane	client
<input type="checkbox"/> Éditer Copier Supprimer	16	Mohamed@gmail.com	Mohamed	client
<input type="checkbox"/> Éditer Copier Supprimer	17	ouassima@gmail.com	ouassima	client
<input type="checkbox"/> Éditer Copier Supprimer	18	Hanane@gmail.com	Hanane	admin
<input type="checkbox"/> Éditer Copier Supprimer	19	Mohamed@gmail.com	Mohamed	client
<input type="checkbox"/> Éditer Copier Supprimer	20	ouassima@gmail.com	ouassima	client
<input type="checkbox"/> Éditer Copier Supprimer	21	Hanane@gmail.com	Hanane	client
<input type="checkbox"/> Éditer Copier Supprimer	22	Mohamed@gmail.com	Mohamed	admin
<input type="checkbox"/> Éditer Copier Supprimer	23	ouassima@gmail.com	ouassima	client
<input type="checkbox"/> Éditer Copier Supprimer	24	Hanane@gmail.com	Hanane	client

Console de requêtes SQL

← T →			id	date_ab	montant	solde	type_ab	client_id	
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	6	2023-04-06	3000	20000	TELEPHONE_FIXE	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	21	2010-10-03	48316	63418	INTERNET	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	29	2012-07-13	23498	62761	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	30	2004-05-03	23498	39500	TELEPHONE_FIXE	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	31	2023-04-23	23498	39500	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	32	2023-04-23	23498	39500	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	33	2023-04-23	48316	62761	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	34	2023-04-19	48316	39500	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	35	1976-12-12	1280220	217862	TELEPHONE_FIXE	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	38	2023-05-03	2900	1200	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	39	2023-05-03	100	5000	INTERNET	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	40	2023-05-03	2900	1200	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	41	2023-05-03	100	5000	INTERNET	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	42	2023-05-03	2900	1200	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	43	2023-05-03	100	5000	INTERNET	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	44	2023-05-03	2900	1200	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	45	2023-05-03	100	5000	INTERNET	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	46	2023-05-03	2900	1200	GSM	1
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	47	2023-05-03	100	5000	INTERNET	2
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	48	2023-05-03	2900	1200	GSM	1

Console de requêtes SQL

Console de requêtes SQL

Couche Web

- **Gérer les clients (Chercher, Pagination, Ajout, Edition et Suppression)**

Le contrôleur **ClientController** qui gère les requêtes liées aux clients. Il contient des méthodes pour afficher la liste des clients, ajouter, modifier et supprimer des clients. Les méthodes sont annotées avec les annotations Spring `@GetMapping` et `@PostMapping` pour indiquer les requêtes HTTP qu'elles gèrent. Les méthodes sont également annotées avec `@PreAuthorize` pour limiter l'accès à certaines fonctions aux utilisateurs ayant des rôles spécifiques. Le contrôleur utilise des services et des repositories pour interagir avec la base de données et afficher les données aux utilisateurs.


```

@Controller
public class ClientController {

    6 usages
    private IAbonnementService clientService;

    2 usages
    private ClientRepository clientRepository;

    no usages AKASMIIOU Ouassima
    public ClientController(IAbonnementService clientService, ClientRepository clientRepository) {
        this.clientService = clientService;
        this.clientRepository = clientRepository;
    }

    // handler method to handle list clients and return mode and view
    no usages AKASMIIOU Ouassima
    @GetMapping("/user/clients")
    public String listClients(Model model, @RequestParam(name = "page", defaultValue = "0") int page,
                               @RequestParam(name = "size", defaultValue = "8") int size,
                               @RequestParam(name = "keyword", defaultValue = "") String kw) {
        Page<Client> pageClients = clientRepository.findByNomContains(kw, PageRequest.of(page, size));
        model.addAttribute( attributeName: "clients", pageClients.getContent());
        model.addAttribute( attributeName: "pages", new int[pageClients.getTotalPages()]);
        model.addAttribute( attributeName: "currentPage", page);
        model.addAttribute( attributeName: "keyword", kw);
        return "clients";
    }

    @GetMapping("/admin/clients/new")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String createClientForm(Model model) {

        // create client object to hold client form data
        Client client = new Client();
        model.addAttribute( attributeName: "client", client);
        return "create_client";
    }

    no usages AKASMIIOU Ouassima
    @PostMapping("/admin/clients")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String saveClient(@Valid Client client, Model model) {
        model.addAttribute( attributeName: "client", client);
        clientService.saveClient(client);
        return "redirect:/user/clients";
    }
}

```

```

@GetMapping("/admin/clients/edit/{id}")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String editClientForm(@PathVariable Long id, Model model) {
    model.addAttribute("client", clientService.getClientById(id));
    return "edit_client";
}

// handler method to handle delete client request

no usages AKASMIOU Ouassima
@GetMapping("/admin/clients/{id}")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteClient(@PathVariable Long id, String keyword, int page) {
    clientService.deleteClientById(id);
    return "redirect:/user/clients?page="+page+"&keyword="+keyword;
}

no usages AKASMIOU Ouassima
@GetMapping("/")
public String home() { return "redirect:/user/clients"; }
}

@PostMapping("/admin/clients/{id}")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String updateClient(@PathVariable Long id,
                           @Valid Client client,
                           Model model) {
    model.addAttribute("client", client);
    // get client from database by id
    @Valid
    Client existingClient = clientService.getClientById(id);
    existingClient.setId(id);
    existingClient.setNom(client.getNom());
    existingClient.setEmail(client.getEmail());
    existingClient.setUsername(client.getUsername());

    // save updated client object
    clientService.updateClient(existingClient);
    return "redirect:/user/clients";
}

```

• Gérer les abonnements

Le contrôleur **AbonnementController** qui gère les requêtes HTTP pour la gestion des abonnements. Il a des méthodes qui prennent en charge les requêtes GET et POST pour l'affichage et la manipulation des abonnements. Il utilise une instance de la classe **IAbonnementService** et les classes **AbonnementRepository** et **ClientRepository** pour accéder aux données d'abonnement et de client. Il y a également des annotations de sécurité qui spécifient les autorisations pour certaines des méthodes.

```

@Controller
public class AbonnementController {
    7 usages
    private IAbonnementService abonnementService;
    1 usage
    private AbonnementRepository abonnementRepository;
    2 usages
    private ClientRepository clientRepository;

    no usages AKASMIIOU Ouassima
    public AbonnementController(IAbonnementService abonnementService, AbonnementRepository abonnementRepository, ClientRepository clientRepository) {
        this.abonnementService = abonnementService;
        this.abonnementRepository = abonnementRepository;
        this.clientRepository = clientRepository;
    }

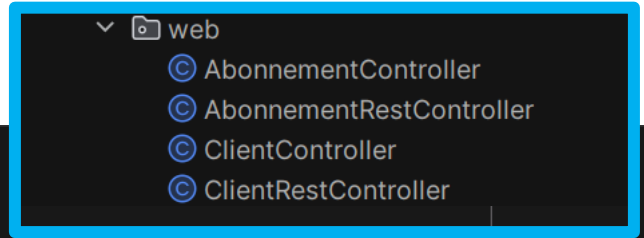
    @GetMapping("/user/abonnements/{id}")
    public String listAbonnements(Model model, @PathVariable Long id) {
        List<Abonnement> abonnements = abonnementService.getByClient_Id(id);
        model.addAttribute( attributeName: "abonnements", abonnements);
        model.addAttribute( attributeName: "id", id);
        return "abonnements";
    }

    @GetMapping("/user/abonnements/new/{id}")
    public String createAbonnementForm(Model model, @PathVariable long id) {
        Abonnement abonnement = new Abonnement();
        model.addAttribute( attributeName: "abonnement", abonnement);
        model.addAttribute( attributeName: "id", id);
        return "create_abonnement";
    }

    @PostMapping("/user/abonnements")
    public String saveAbonnement(@Valid Abonnement abonnement, Model model, @RequestParam Long clientId) {
        Client client = clientRepository.findById(clientId).orElseThrow(() -> new IllegalArgumentException("Invalid client Id:" + clientId));
        abonnement.setClient(client);
        model.addAttribute( attributeName: "abonnement", abonnement);
        abonnementService.saveAbonnement(abonnement);
        return "redirect:/user/abonnements/"+clientId;
    }

    no usages AKASMIIOU Ouassima
    @GetMapping("/admin/abonnements/edit/{id}")
    @PreAuthorize("hasRole('ROLE_ADMIN')")
    public String editAbonnementForm(@PathVariable Long id, Model model, @RequestParam(value = "clientId") long clientId) {
        model.addAttribute( attributeName: "abonnement", abonnementService.getAbonnementById(id));
        model.addAttribute( attributeName: "clientId", clientId);
        return "edit_abonnement";
    }
}

```



```

@PostMapping(⌚"/admin/abonnements/{id}")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String updateAbonnement(@PathVariable Long id,
                               @Valid Abonnement abonnement,
                               Model model, @RequestParam Long clientId) {
    model.addAttribute( attributeName: "abonnement", abonnement);
    Abonnement existingabonnement = abonnementService.getAbonnementById(id);
    existingabonnement.setId(id);
    existingabonnement.setTypeAb(abonnement.getTypeAb());
    existingabonnement.setDateAb(abonnement.getDateAb());
    existingabonnement.setSolde(abonnement.getSolde());
    existingabonnement.setMontant(abonnement.getMontant());
    abonnementService.updateAbonnement(existingabonnement);
    return "redirect:/user/abonnements/"+clientId;
}
no usages  AKASMIOU Ouassima
@GetMapping(⌚"/admin/deleteAbonnement/{id}")
@PreAuthorize("hasRole('ROLE_ADMIN')")
public String deleteAbonnement(@PathVariable Long id, @RequestParam(value = "clientId") Long clientId) {
    abonnementService.deleteAbonnementById(id);
    return "redirect:/user/abonnements/"+clientId;
}

```

- Création un web service RESTful qui permet de gérer les clients et les abonnements

✓ ClientRestController :

```

@org.springframework.web.bind.annotation.RestController
@RequestMapping(⌚"/api")
public class ClientRestController {
    7 usages
    private ClientRepository clientRepository;

    no usages  new *
    public ClientRestController(ClientRepository clientRepository) {
        this.clientRepository = clientRepository;
    }

    no usages  new *
    @GetMapping(⌚"/clients")
    public List<Client> getClients() {
        List<Client> clients = clientRepository.findAll();
        for (Client client : clients) {
            client.getAbonnements().size(); // chargement paresseux des abonnements pour éviter la boucle infinie
        }
        return clients;
    }
}

```

```

@GetMapping(Ⓜ"/clients/{id}")
public Client getClientById(@PathVariable Long id) {
    return clientRepository.findById(id)
        .orElseThrow(() -> new RuntimeException(HttpStatus.NOT_FOUND, String.format("Client %d not found", id)));
}

no usages new *
@PostMapping(Ⓜ"/clients")
public Client createClient(@RequestBody Client client) {
    return clientRepository.save(client);
}

no usages
@PutMapping(Ⓜ"/clients/{id}")
public Client updateClient(@PathVariable Long id, @RequestBody Client client) {
    Client existingClient = clientRepository.findById(id)
        .orElseThrow(() -> new RuntimeException(HttpStatus.NOT_FOUND, String.format("Client %d not found", id)));
    existingClient.setNom(client.getNom());
    existingClient.setEmail(client.getEmail());
    existingClient.setUsername(client.getUsername());
    existingClient.setAbonnements(client.getAbonnements());
    return clientRepository.save(existingClient);
}

no usages
@DeleteMapping(Ⓜ"/clients/{id}")
public void deleteClient(@PathVariable Long id) {
    clientRepository.deleteById(id);
}
}

```

✓ AbonnementRestController :

```

@org.springframework.web.bind.annotation.RestController
@RequestMapping(Ⓜ"/api")
public class AbonnementRestController {
    7 usages
    private AbonnementRepository abonnementRepository;
    3 usages
    private ClientRepository clientRepository;
    no usages AKASMIOU Ouassima
    public AbonnementRestController(AbonnementRepository abonnementRepository, ClientRepository clientRepository) {
        this.abonnementRepository = abonnementRepository;
        this.clientRepository = clientRepository;
    }

    no usages AKASMIOU Ouassima
    @GetMapping(Ⓜ"/abonnements")
    public List<Abonnement> getAbonnements() {
        List<Abonnement> abonnements = abonnementRepository.findAll();
        for (Abonnement abonnement : abonnements) {
            abonnement.getClient().getId(); // chargement paresseux du client pour éviter la boucle infinie
        }
        return abonnements;
    }
}

```

```

@GetMapping(Ⓜ"/abonnements/{id}")
public Abonnement getAbonnementById(@PathVariable Long id) {
    return abonnementRepository.findById(id)
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, String.format("Subscription %d not found", id)));
}

no usages  AKASMIIOU Ouassima

@PostMapping(Ⓜ"/abonnements")
public Abonnement createAbonnement(@RequestBody Abonnement abonnement) {
    // Vérifier si le client existe
    Client client = clientRepository.findById(abonnement.getClient().getId())
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, String.format("Client %d not found", abonnement.ge

    // Associer le client à l'abonnement
    abonnement.setClient(client);

    return abonnementRepository.save(abonnement);
}

no usages  AKASMIIOU Ouassima

@PutMapping(Ⓜ"/abonnements/{id}")
public Abonnement updateAbonnement(@PathVariable Long id, @RequestBody Abonnement abonnement) {
    // Vérifier si l'abonnement existe
    Abonnement existingAbonnement = abonnementRepository.findById(id)
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, String.format("Subscription %d not found", id)));

    // Vérifier si le client existe
    Client client = clientRepository.findById(abonnement.getClient().getId())
        .orElseThrow(() -> new ResponseStatusException(HttpStatus.NOT_FOUND, String.format("Client %d not found", abonnement.ge

    // Mettre à jour les informations de l'abonnement
    existingAbonnement.setDateAb(abonnement.getDateAb());
    existingAbonnement.setSolde(abonnement.getSolde());
    existingAbonnement.setMontant(abonnement.getMontant());
    existingAbonnement.setTypeAb(abonnement.getTypeAb());
    existingAbonnement.setClient(client);

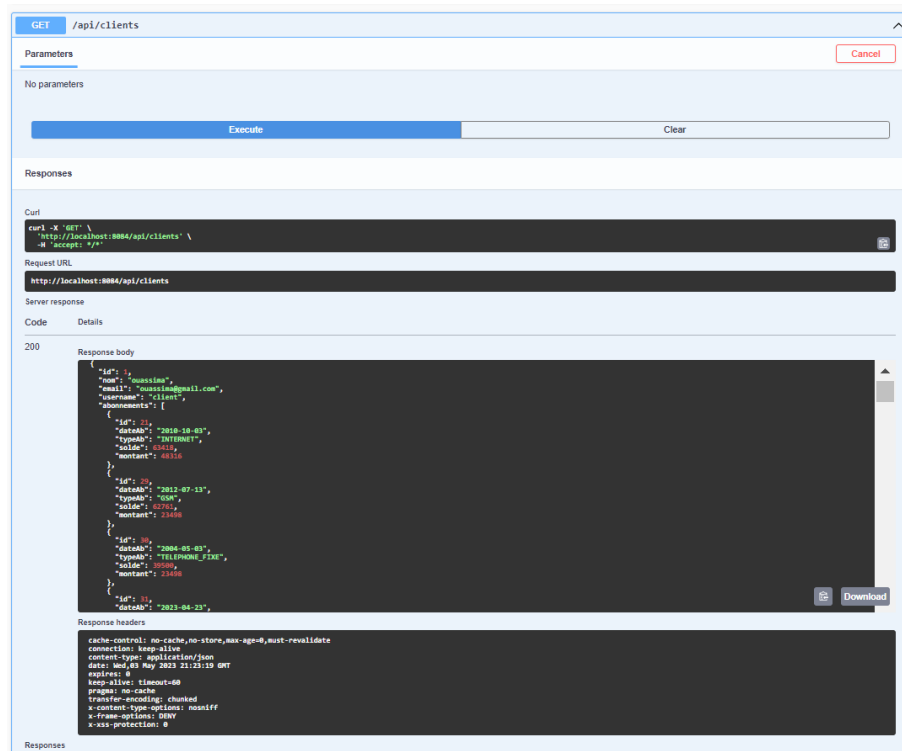
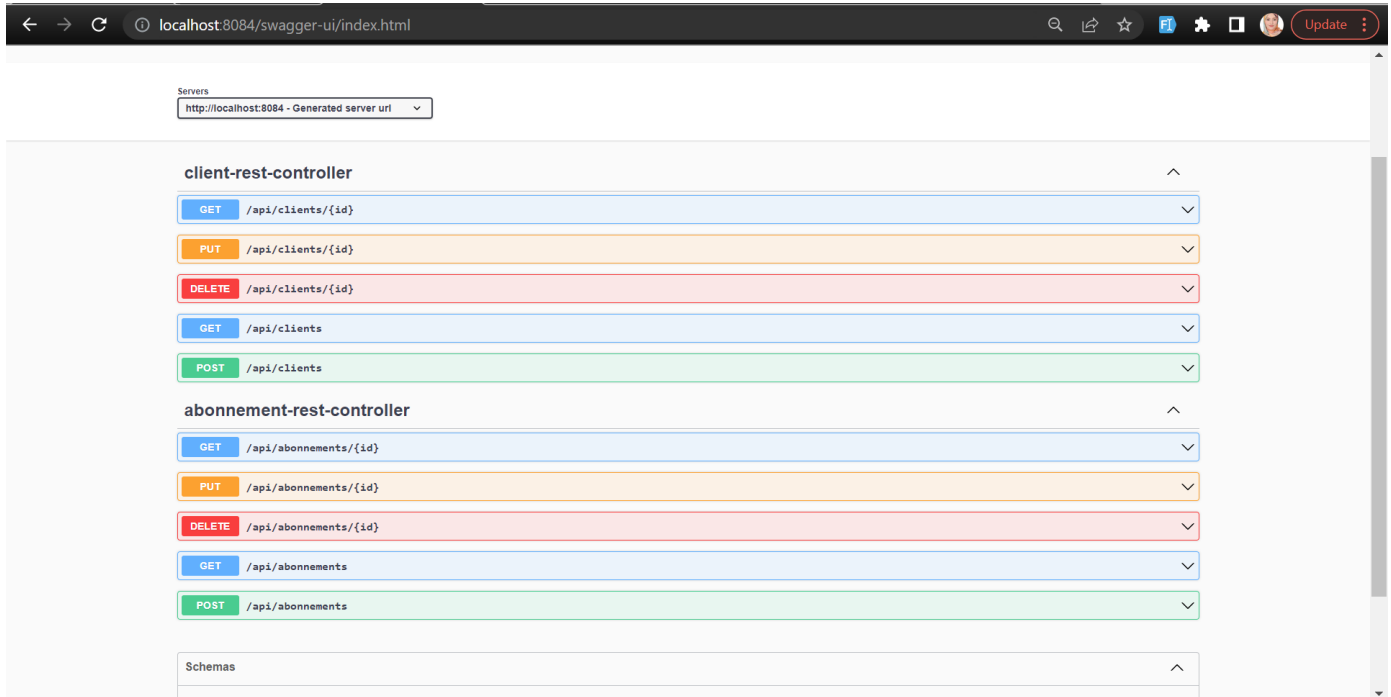
    return abonnementRepository.save(existingAbonnement);
}

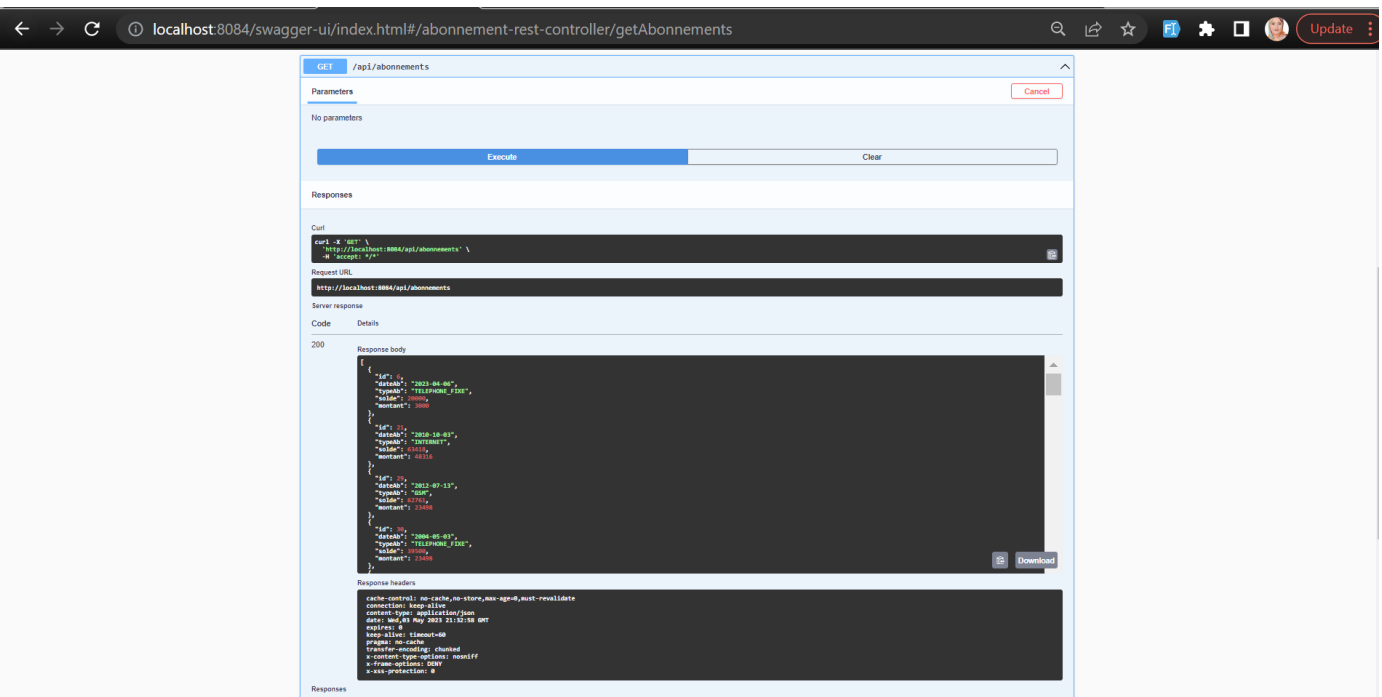
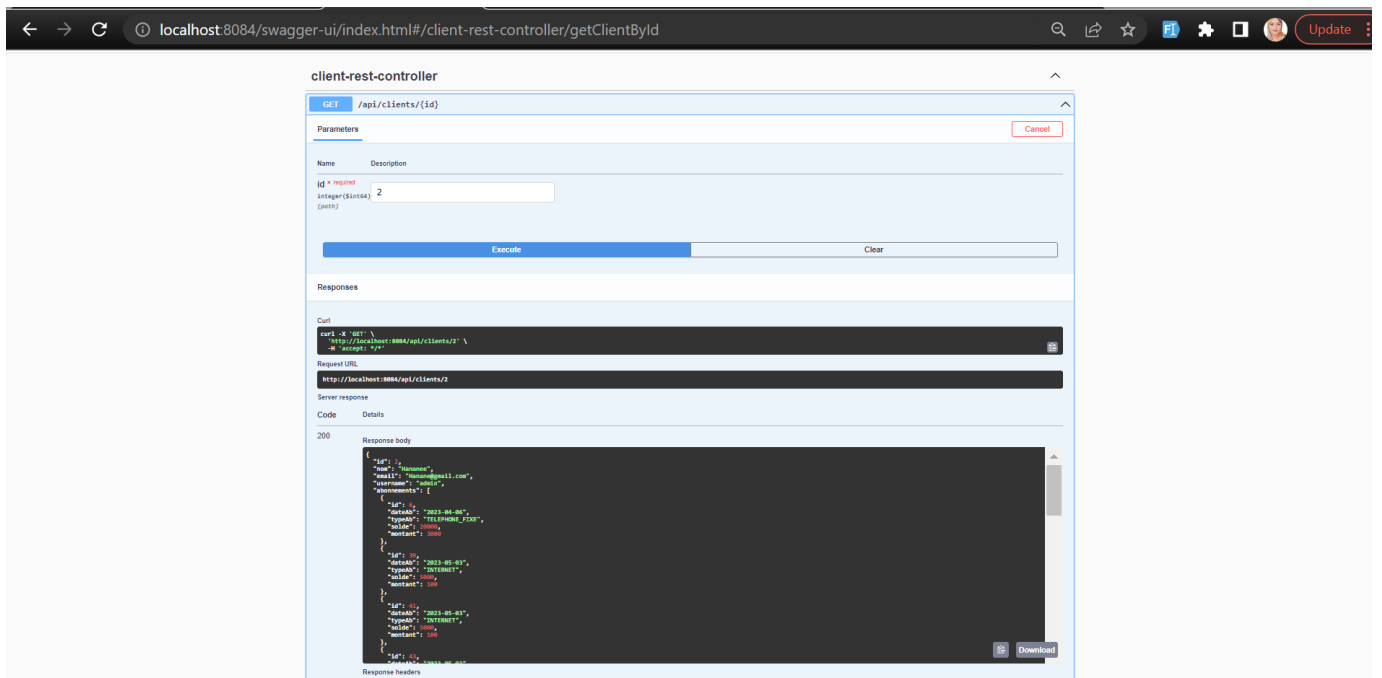
no usages  AKASMIIOU Ouassima

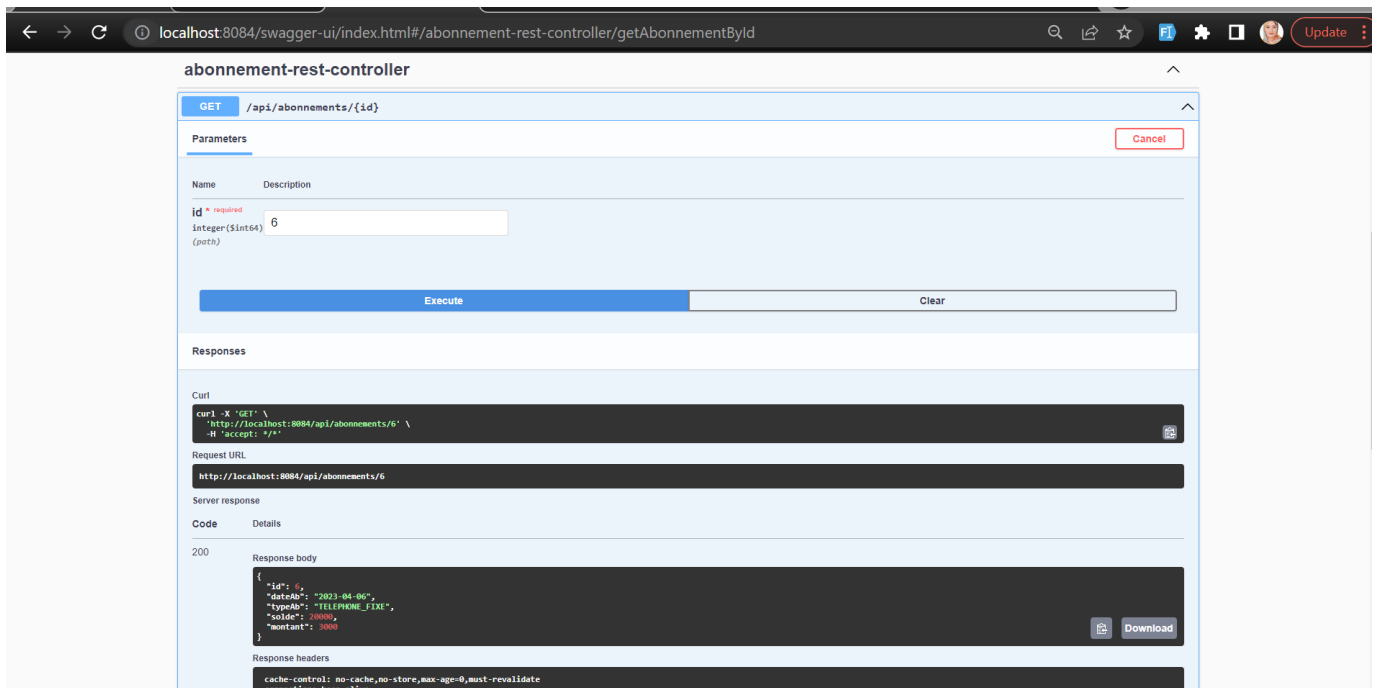
@DeleteMapping(Ⓜ"/abonnements/{id}")
public void deleteAbonnement(@PathVariable Long id) {
    abonnementRepository.deleteById(id);
}

```

Nous allons utiliser Swagger pour tester la méthode GET de notre API REST. Nous allons envoyer une requête à l'URL correspondante et vérifier que les données retournées sont bien celles attendues. En utilisant Swagger, nous pourrions également visualiser la documentation de notre API et découvrir les différentes méthodes disponibles.







- **Sécurité : Sécuriser l'accès à l'application en développant un système d'authentification statefull basé sur Spring Security avec deux rôles USER et ADMIN.**

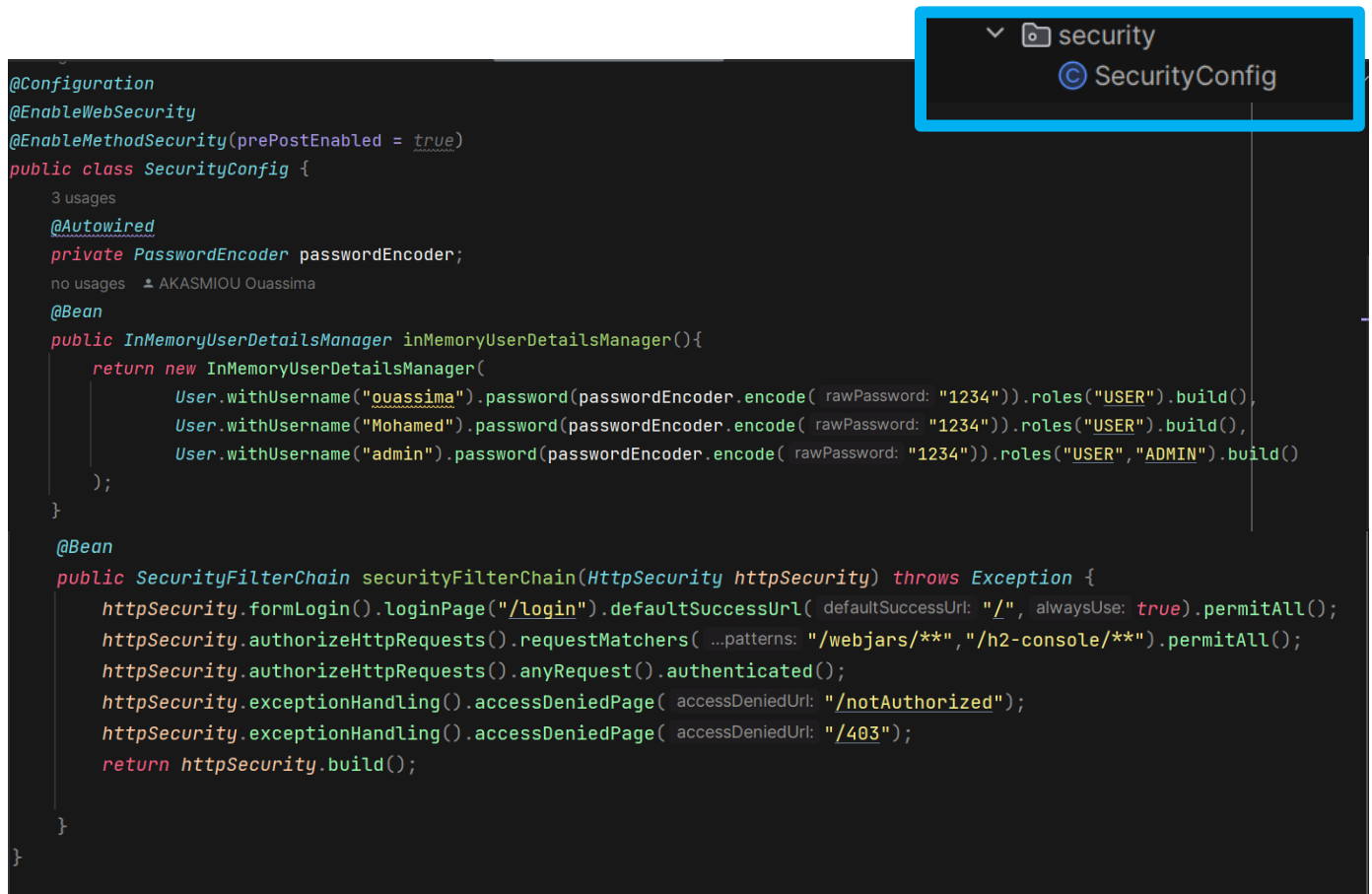
Pour la sécurité de l'application en intégrant une fonctionnalité d'authentification et de gestion de rôles. Les utilisateurs qui auront le rôle ADMIN pourront créer, lire, mettre à jour et supprimer des clients et ses abonnements, tandis que les utilisateurs ayant le rôle USER seront limités à la consultation des données des clients et de leurs abonnements. Cependant, ils auront tout de même la possibilité de charger les abonnements d'un client.

Ajouter la dépendance de Sécurité dans le fichier maven (pom.xml).

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- https://mvnrepository.com/artifact/org.thymeleaf.extras/thymeleaf-extras-springsecurity6 -->
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
  <version>3.1.0.M1</version>
</dependency>
```

Configuration la sécurité d'une application web en utilisant Spring Security. Il crée un objet `InMemoryUserDetailsManager` pour stocker des informations de connexion en mémoire, définit des règles de sécurité pour les requêtes HTTP et gère les pages d'erreur. La méthode `securityFilterChain()` retourne un objet `SecurityFilterChain` qui est utilisé pour protéger les ressources.



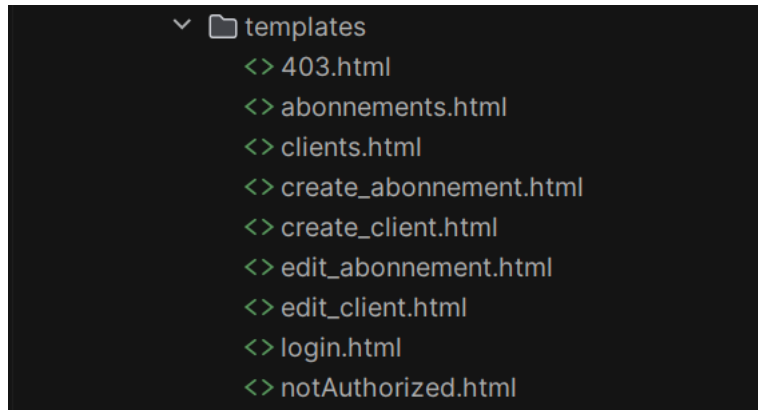
Le rôle de ce contrôleur est de gérer les demandes de connexion et de gestion de sécurité dans une application Web. Il fournit des pages de connexion et de refus d'autorisation pour les utilisateurs non autorisés et redirige les utilisateurs authentifiés vers la page d'accueil de l'application.

```

@Controller
public class SecurityController {
    no usages AKASMIQU Ouassima
    @GetMapping(🌐"/notAuthorized")
    public String notAuthorized() { return "notAuthorized"; }

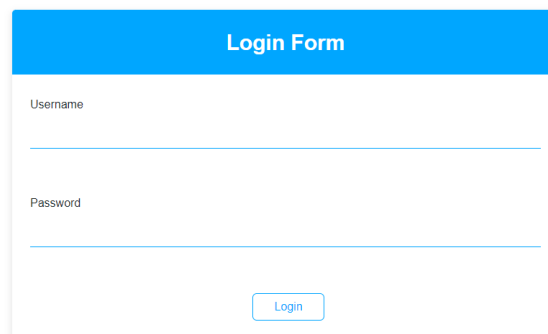
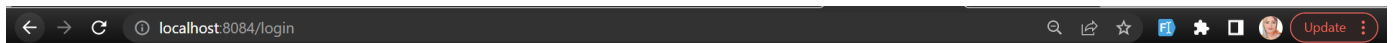
    no usages AKASMIQU Ouassima
    @GetMapping(🌐"/login")
    public String login() { return "login"; }
}
    
```

Sous le dossier ressource on crée un dossier template dans lequel on crée nos pages html.



- Les interfaces (démonstration)

- ✓ Page Login

A screenshot of a web application's login form. The form has a blue header with the text 'Login Form'. Below the header, there are two input fields: 'Username' and 'Password'. Each field has a blue underline. At the bottom of the form, there is a blue button labeled 'Login'.

Lorsque nous nous authentifions avec un utilisateur ayant le rôle ADMIN, nous avons tous les droits pour ajouter, modifier ou supprimer des clients et ses abonnements.

Login Form

Username

Password

Interface dédiée à l'affichage des clients.

← → ↻ localhost:8084/user/clients
🔑 🔍 📄 ☆ ⚙️ 🖨️ 👤 Update ⋮

Subscription Management System
Subscription Management admin ▾

List Clients

Add Client

Keyword : 🔍

ID	Name	Email	Username	Subscription	Actions
1	ouassima	ouassima@gmail.com	client	show	🗑️ ✎
2	Hanane	Hanane@gmail.com	admin	show	🗑️ ✎
5	Mohammed	mohammed@gmail.com	client	show	🗑️ ✎
6	Jinan	jinan@gmail.com	client	show	🗑️ ✎
7	Oualid	oualid@gmail.com	client	show	🗑️ ✎
8	Maryam	maryam@gmail.com	client	show	🗑️ ✎
9	Amal	amal@gmail	client	show	🗑️ ✎
12	Neville Rhodes	pyrry@mailinator.com	client	show	🗑️ ✎

0
1
2
3
4
5

Formulaire pour ajouter un client.



















Formulaire pour modifier un client.



Il est possible d'afficher les abonnements d'un client en cliquant sur le bouton « Show », par exemple on affiche les abonnements de client d'Id 1.

List Clients





































[Add Client](#)Keyword : 

ID	Name	Email	Username	Subscription	Actions
1	ouassima	ouassima@gmail.com	client	show	 
2	Hanane	Hanane@gmail.com	admin	show	 
5	Mohammed	mohammed@gmail.com	client	show	 
6	Jinan	jinan@gmail.com	client	show	 
7	Oualid	oualid@gmail.com	client	show	 
8	Maryam	maryam@gmail.com	client	show	 
9	Amal	amal@gmail	client	show	 
12	Neville Rhodes	pyny@mailinator.com	client	show	 

0 1 2 3 4 5

List Abonnements

[Add Subscription](#)

ID	Subscription Date	Subscription Type	Balance	Amount	Actions
21	2010-10-03	INTERNET	63418.0	48316.0	 
29	2012-07-13	GSM	62761.0	23498.0	 
30	2004-05-03	TELEPHONE_FIXE	39500.0	23498.0	 
31	2023-04-23	GSM	39500.0	23498.0	 
32	2023-04-23	GSM	39500.0	23498.0	 
33	2023-04-23	GSM	62761.0	48316.0	 
34	2023-04-19	GSM	39500.0	48316.0	 
35	1976-12-12	TELEPHONE_FIXE	217862.0	1280220.0	 
38	2023-05-03	GSM	1200.0	2900.0	 
40	2023-05-03	GSM	1200.0	2900.0	 
42	2023-05-03	GSM	1200.0	2900.0	 
44	2023-05-03	GSM	1200.0	2900.0	 
46	2023-05-03	GSM	1200.0	2900.0	 
48	2023-05-03	GSM	1200.0	2900.0	 
50	2023-05-03	GSM	1200.0	2900.0	 
52	2023-05-03	GSM	1200.0	2900.0	 
54	2023-05-03	GSM	1200.0	2900.0	 
56	2023-05-03	GSM	1200.0	2900.0	 

On peut ajouter un abonnement d'un client spécifique.

Create New Subscription

Subscription Date

Type

Balance

Amount

[Submit](#)

Il est également possible de modifier l'abonnement d'un client spécifique.

Subscription Management System Subscription Management admin

Update Subscription

Subscription Date
03/10/2010

Type
INTERNET

Balance
63418.0

Amount
48316.0

Submit

Lorsque l'on supprime un abonnement ou un client, une alerte de confirmation s'affiche pour confirmer l'opération.

















Subscription Management System Subscription Management admin

localhost:8084 says
Etes vous sûr?

OK Cancel

List Abonnements

Add Subscription

ID	Subscription Date	Type	Balance	Amount	Actions
21	2010-10-03	INTERNET	63418.0	48316.0	 
29	2012-07-13	GSM	62761.0	23498.0	 
30	2004-05-03	TELEPHONE_FIXE	39500.0	23498.0	 
31	2023-04-23	GSM	39500.0	23498.0	 
32	2023-04-23	GSM	39500.0	23498.0	 
33	2023-04-23	GSM	62761.0	48316.0	 
34	2023-04-19	GSM	39500.0	48316.0	 
35	1976-12-12	TELEPHONE_FIXE	217862.0	1280220.0	 

Nous effectuons la déconnexion puis nous nous authentifions avec un utilisateur ayant le rôle USER.


admin

Logout

Par conséquent, nous n'avons pas l'autorisation d'ajouter, modifier ou supprimer un client.

Subscription Management System Subscription Management ouassima

List Clients

Keyword : 

ID	Name	Email	Username	Subscription
1	ouassima	ouassima@gmail.com	client	show
2	Hanane	Hanane@gmail.com	admin	show
5	Mohammed	mohammed@gmail.com	client	show
6	Jinan	jinan@gmail.com	client	show
7	Oualid	oualid@gmail.com	client	show
8	Maryam	maryam@gmail.com	client	show
9	Amal	amal@gmail	client	show
12	Neville Rhodes	pyny@mailinator.com	client	show

0 1 2 3 4 5

Nous affichons les abonnements du client 1 et comme vous pouvez le constater, nous n'avons pas la possibilité de modifier ou de supprimer un abonnement existant. Cependant, nous avons la possibilité d'ajouter un nouvel abonnement pour ce client.

Subscription Management System Subscription Management ouassima

List Abonnements

[Add Subscription](#)

ID	Subscription Date	Subscription Type	Balance	Amount
21	2010-10-03	INTERNET	63418.0	48316.0
29	2012-07-13	GSM	62761.0	23498.0
30	2004-05-03	TELEPHONE_FIXE	39500.0	23498.0
31	2023-04-23	GSM	39500.0	23498.0
32	2023-04-23	GSM	39500.0	23498.0
33	2023-04-23	GSM	62761.0	48316.0
34	2023-04-19	GSM	39500.0	48316.0
35	1976-12-12	TELEPHONE_FIXE	217862.0	1280220.0
38	2023-05-03	GSM	1200.0	2900.0
40	2023-05-03	GSM	1200.0	2900.0
42	2023-05-03	GSM	1200.0	2900.0
44	2023-05-03	GSM	1200.0	2900.0
46	2023-05-03	GSM	1200.0	2900.0
48	2023-05-03	GSM	1200.0	2900.0
50	2023-05-03	GSM	1200.0	2900.0
52	2023-05-03	GSM	1200.0	2900.0