

**Examen Architecture Distribuée et Middleware**Programmation Orientée Objet en JAVA

|  |
| --- |
| **Réalisé par :** |
| **Youssef Faik** |

**Année Professionnelle : 2024/2025**

# Introduction :

Ce rapport documente la conception et le développement d'une application Web JEE pour la gestion des crédits bancaires, réalisée dans le cadre d'une évaluation. L'application, bâtie sur les frameworks Spring et Angular, vise à gérer les clients, les différents types de crédits (Personnel, Immobilier, Professionnel) ainsi que leurs remboursements. Le document expose l'architecture technique, la modélisation des données, l'implémentation des couches métier et d'accès aux données avec Spring Boot, la création des services web REST, et la sécurisation des accès via Spring Security et JWT.

**Conception :**

## Architecture Technique du Projet :

L'architecture du projet est une application Web multicouche. Un Frontend Angular interagit avec un Backend JEE (Spring Boot) via des API REST.

Le Backend comprend :

* Couche Sécurité (Spring Security, JWT) pour l'authentification et l'autorisation.
* Couche Web (Spring MVC REST Controllers) pour exposer les API.
* Couche Service pour la logique métier.
* Couche DAO (Spring Data, JPA, Hibernate) pour l'accès aux données.

Les données sont persistées dans une Base de Données MySQL (potentiellement conteneurisée avec Docker).

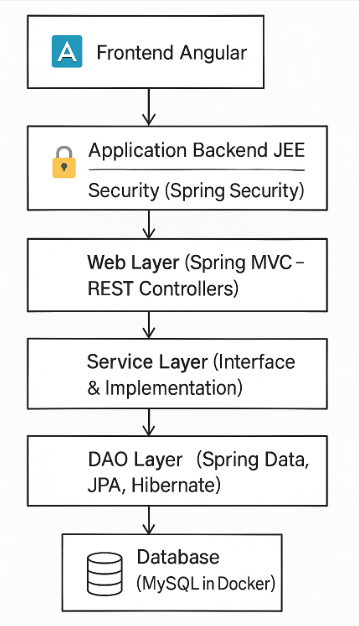


Figure 1 Architecture Technique de l'Application Web JEE et Angular

## Diagramme de Classes des Entités :

Le diagramme de classes ci-dessous (Figure 2) représente les entités JPA du domaine de l'application de gestion de crédits bancaires, en se concentrant sur leurs attributs. Il illustre les relations entre les entités principales : Client, Credit (et ses spécialisations CreditPersonnel, CreditImmobilier, CreditProfessionnel), et Remboursement, ainsi que les énumérations associées (StatutCredit, TypeBienImmobilier, TypeRemboursement). La classe Credit est définie comme abstraite et sert de base pour les différents types de crédits.

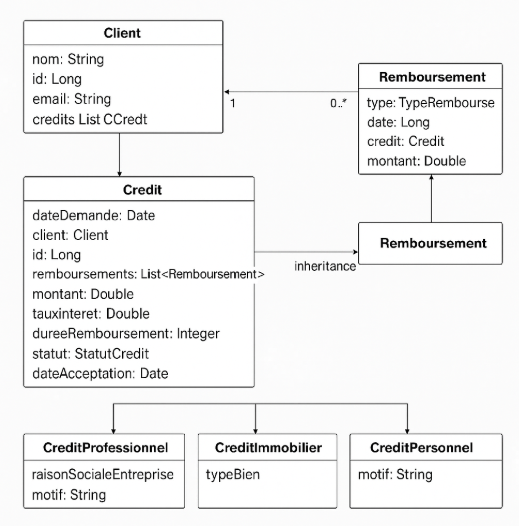


Figure 2 Diagramme de Classes des Entités JPA

# Implémentation :

## Couche DAO :

Nous avons défini plusieurs entités JPA représentant les concepts métiers de l'application de gestion de crédits bancaires. Voici un aperçu des principales entités :

Client

A screen shot of a computer program

AI-generated content may be incorrect.

Credit (classe abstraite):

A screen shot of a computer program

AI-generated content may be incorrect.

CréditPersonnel:

A screen shot of a computer program

AI-generated content may be incorrect.

CreditImmobilier

A screen shot of a computer program

AI-generated content may be incorrect.

CreditProfessionnel:

A screen shot of a computer program

AI-generated content may be incorrect.

Remboursement:

A screen shot of a computer program

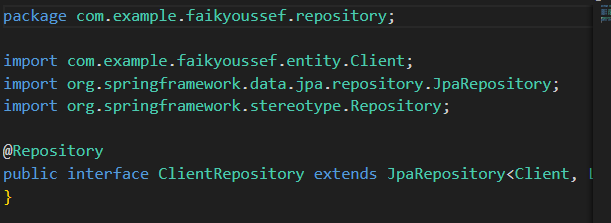
AI-generated content may be incorrect.

## Création des interfaces JPA Repository :

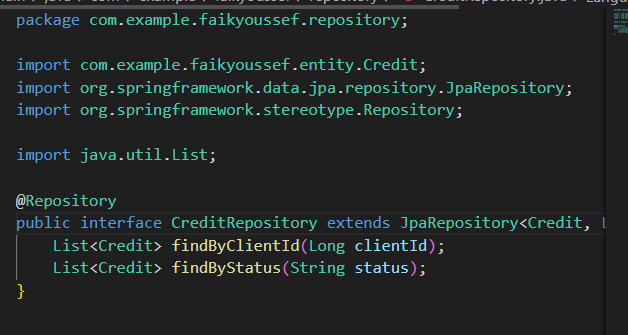
Les interfaces ci-dessous héritent de JpaRepository, ce qui permet de profiter des fonctionnalités de Spring Data JPA pour effectuer des opérations CRUD sans implémentation manuelle.

**Liste des interfaces créées :**

**ClientRepository :**

****

**CreditRepository**

****

RemboursementRepository :

A screen shot of a computer program

AI-generated content may be incorrect.

## Couche Service (DTOs, Mappers, Services) :

La couche service constitue la couche métier de l'application. Elle permet de séparer la logique métier des autres couches (DAO, Web) et facilite la maintenance, la réutilisation du code, ainsi que la sécurité et les tests unitaires.

Elle est composée de trois éléments clés :

* Les DTOs : pour transporter des données entre les couches.
* Les Mappers : pour convertir entre les entités JPA et les DTOs.
* Les Services : pour implémenter la logique métier via des interfaces et des classes concrètes.

### Création des DTOs :

Nous avons créé les DTOs suivants :

* ClientDTO, ClientRequestDTO, ClientSummaryDTO
* CreditDTO, CreditRequestDTO, CreditSummaryDTO
* CreditPersonnelDTO, CreditImmobilierDTO, CreditProfessionnelDTO
* CreditPersonnelRequestDTO, CreditImmobilierRequestDTO, CreditProfessionnelRequestDTO
* RemboursementDTO, RemboursementRequestDTO

### Exemple :

ClientDto :

A computer screen shot of a program code

AI-generated content may be incorrect.

CreditDTO:



LoginRequestDTO  
A computer screen with text on it

AI-generated content may be incorrect.

### Création des Mappers :

Les mappers permettent de transformer automatiquement les entités JPA en DTOs et vice versa.

Les classes de mapping suivantes ont été créées :

* ClientMapper.java
* CreditMapper.java
* RemboursementMapper.java

### Exemple :

ClientMapper :

A screen shot of a computer program

AI-generated content may be incorrect.

CreditMapper :

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

### Création des Services:

**Services principaux :**

* ClientService & ClientServiceImpl
* CreditService & CreditServiceImpl
* RemboursementService & RemboursementServiceImpl
* ReportingService & ReportingServiceImpl (pour fonctionnalités avancées)

### Exemple :

ClientService :

A screen shot of a computer code

AI-generated content may be incorrect.

ClientServiceImpl :

A screen shot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

## Couche Web : REST Controllers + Swagger (OpenAPI):

La couche Web est responsable de l'exposition des services via des API RESTful, permettant à des clients (comme Angular) d'interagir avec l’application Spring Boot. Elle est construite à l’aide de Spring MVC.

Les fonctionnalités exposées sont documentées grâce à Swagger (OpenAPI), ce qui facilite le test et l'intégration des API.

### Contrôleurs REST créés :

Les contrôleurs suivants ont été implémentés pour chaque entité principale :

|  |  |
| --- | --- |
| ClientController.java | Gérer les clients |
| CreditController.java | Gérer les crédits (tous types) |
| RemboursementController.java | Gérer les remboursements |
| HomeController.java | Page d’accueil ou test simple |

### Exemple :

ClientController :

|  |
| --- |
| package com.example.faikyoussef.controller;  import com.example.faikyoussef.dto.ClientDto;  import com.example.faikyoussef.service.ClientService;  import io.swagger.v3.oas.annotations.Operation;  import io.swagger.v3.oas.annotations.media.Content;  import io.swagger.v3.oas.annotations.media.ExampleObject;  import io.swagger.v3.oas.annotations.media.Schema;  import io.swagger.v3.oas.annotations.parameters.RequestBody;  import io.swagger.v3.oas.annotations.responses.ApiResponse;  import io.swagger.v3.oas.annotations.security.SecurityRequirement;  import io.swagger.v3.oas.annotations.tags.Tag;  import jakarta.validation.Valid;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.security.access.prepost.PreAuthorize;  import org.springframework.web.bind.annotation.\*;  import java.util.List;  @RestController  @RequestMapping("/api/v1/clients")  @Tag(name = "Client Management", description = "APIs for managing clients")  public class ClientController {      private final ClientService clientService;      public ClientController(ClientService clientService) {          this.clientService = clientService;      }    @PostMapping      @PreAuthorize("hasRole('EMPLOYE') or hasRole('ADMIN')")      @Operation(summary = "Create a new client",              description = "Creates a new client with the provided details.",              requestBody = @RequestBody(description = "Client data to create",                      content = @Content(mediaType = "application/json",                              schema = @Schema(implementation = ClientDto.class),                              examples = @ExampleObject(value = "{\"name\": \"John Doe\", \"email\": \"john.doe@example.com\"}"))),              responses = {                      @ApiResponse(responseCode = "201", description = "Client created successfully"),                      @ApiResponse(responseCode = "400", description = "Invalid input"),                      @ApiResponse(responseCode = "403", description = "Access denied")              },              security = @SecurityRequirement(name = "bearerAuth"))      public ResponseEntity<ClientDto> createClient(@Valid @RequestBody ClientDto clientDto) {          ClientDto savedClient = clientService.saveClient(clientDto);          return new ResponseEntity<>(savedClient, HttpStatus.CREATED);      }    @GetMapping("/{id}")      @Operation(summary = "Get a client by ID",              description = "Retrieves a specific client by their ID.",              responses = {                      @ApiResponse(responseCode = "200", description = "Client found"),                      @ApiResponse(responseCode = "404", description = "Client not found"),                      @ApiResponse(responseCode = "403", description = "Access denied")              },              security = @SecurityRequirement(name = "bearerAuth"))      public ResponseEntity<ClientDto> getClientById(@PathVariable Long id) {          ClientDto clientDto = clientService.getClientById(id);          return ResponseEntity.ok(clientDto);      }    @GetMapping      @PreAuthorize("hasRole('EMPLOYE') or hasRole('ADMIN')")      @Operation(summary = "Get all clients",              description = "Retrieves a list of all clients.",              responses = {                      @ApiResponse(responseCode = "200", description = "Successfully retrieved list"),                      @ApiResponse(responseCode = "403", description = "Access denied")              },              security = @SecurityRequirement(name = "bearerAuth"))      public ResponseEntity<List<ClientDto>> getAllClients() {          List<ClientDto> clients = clientService.getAllClients();          return ResponseEntity.ok(clients);      }    @PutMapping("/{id}")      @PreAuthorize("hasRole('EMPLOYE') or hasRole('ADMIN')")      @Operation(summary = "Update an existing client",              description = "Updates an existing client with the provided details.",              requestBody = @RequestBody(description = "Client data to update",                      content = @Content(mediaType = "application/json",                              schema = @Schema(implementation = ClientDto.class),                              examples = @ExampleObject(value = "{\"name\": \"Jane Doe\", \"email\": \"jane.doe@example.com\"}"))),              responses = {                      @ApiResponse(responseCode = "200", description = "Client updated successfully"),                      @ApiResponse(responseCode = "400", description = "Invalid input"),                      @ApiResponse(responseCode = "404", description = "Client not found"),                      @ApiResponse(responseCode = "403", description = "Access denied")              },              security = @SecurityRequirement(name = "bearerAuth"))      public ResponseEntity<ClientDto> updateClient(@PathVariable Long id, @Valid @RequestBody ClientDto clientDto) {          ClientDto updatedClient = clientService.updateClient(id, clientDto);          return ResponseEntity.ok(updatedClient);      }    @DeleteMapping("/{id}")      @PreAuthorize("hasRole('ADMIN')")      @Operation(summary = "Delete a client by ID",              description = "Deletes a specific client by their ID.",              responses = {                      @ApiResponse(responseCode = "204", description = "Client deleted successfully"),                      @ApiResponse(responseCode = "404", description = "Client not found"),                      @ApiResponse(responseCode = "403", description = "Access denied")              },              security = @SecurityRequirement(name = "bearerAuth"))      public ResponseEntity<Void> deleteClient(@PathVariable Long id) {          clientService.deleteClient(id);          return ResponseEntity.noContent().build();      }  } |

CreditController :

|  |
| --- |
| package com.example.faikyoussef.controller;  import com.example.faikyoussef.dto.CreditDto;  import com.example.faikyoussef.service.CreditService;  import io.swagger.v3.oas.annotations.Operation;  import io.swagger.v3.oas.annotations.media.Content;  import io.swagger.v3.oas.annotations.media.ExampleObject;  import io.swagger.v3.oas.annotations.media.Schema;  import io.swagger.v3.oas.annotations.parameters.RequestBody;  import io.swagger.v3.oas.annotations.responses.ApiResponse;  import io.swagger.v3.oas.annotations.tags.Tag;  import jakarta.validation.Valid;  import org.springframework.http.HttpStatus;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.\*;  import java.util.List;  @RestController  @RequestMapping("/api/v1/credits")  @Tag(name = "Credit Management", description = "APIs for managing credits")  public class CreditController {      private final CreditService creditService;      public CreditController(CreditService creditService) {          this.creditService = creditService;      }      @PostMapping("/clients/{clientId}")      @Operation(summary = "Create a new credit for a client",              description = "Creates a new credit for a specific client with the provided details.",            requestBody = @RequestBody(description = "Credit data to create",                      content = @Content(mediaType = "application/json",                              schema = @Schema(implementation = CreditDto.class),                              examples = @ExampleObject(value = "{\"amount\": 10000.0, \"status\": \"PENDING\", \"typeCredit\": \"PERSONNEL\"}"))),              responses = {                      @ApiResponse(responseCode = "201", description = "Credit created successfully"),                      @ApiResponse(responseCode = "400", description = "Invalid input"),                      @ApiResponse(responseCode = "404", description = "Client not found")              })      public ResponseEntity<CreditDto> createCredit(@PathVariable Long clientId, @Valid @org.springframework.web.bind.annotation.RequestBody CreditDto creditDto) {          CreditDto savedCredit = creditService.saveCredit(creditDto, clientId);          return new ResponseEntity<>(savedCredit, HttpStatus.CREATED);      }      @GetMapping("/{id}")      @Operation(summary = "Get a credit by ID",              description = "Retrieves a specific credit by its ID.",              responses = {                      @ApiResponse(responseCode = "200", description = "Credit found"),                      @ApiResponse(responseCode = "404", description = "Credit not found")              })      public ResponseEntity<CreditDto> getCreditById(@PathVariable Long id) {          CreditDto creditDto = creditService.getCreditById(id);          return ResponseEntity.ok(creditDto);      }      @GetMapping      @Operation(summary = "Get all credits or filter by status",              description = "Retrieves a list of all credits, optionally filtered by status (e.g., EN\_ATTENTE, APPROUVE, REJETE).",              responses = {                      @ApiResponse(responseCode = "200", description = "Successfully retrieved list")              })      public ResponseEntity<List<CreditDto>> getAllCredits(@RequestParam(required = false) String status) {          List<CreditDto> credits;          if (status != null && !status.isEmpty()) {              credits = creditService.getCreditsByStatus(status);          } else {              credits = creditService.getAllCredits();          }          return ResponseEntity.ok(credits);      }      @GetMapping("/clients/{clientId}")      @Operation(summary = "Get all credits for a specific client",              description = "Retrieves a list of all credits associated with a specific client ID.",              responses = {                      @ApiResponse(responseCode = "200", description = "Successfully retrieved list"),                      @ApiResponse(responseCode = "404", description = "Client not found")              })      public ResponseEntity<List<CreditDto>> getCreditsByClientId(@PathVariable Long clientId) {          List<CreditDto> credits = creditService.getCreditsByClientId(clientId);          return ResponseEntity.ok(credits);      }      @PutMapping("/{id}")      @Operation(summary = "Update an existing credit",              description = "Updates an existing credit with the provided details. Note: Client ID cannot be changed via this endpoint.",            requestBody = @RequestBody(description = "Credit data to update",                      content = @Content(mediaType = "application/json",                              schema = @Schema(implementation = CreditDto.class),                              examples = @ExampleObject(value = "{\"amount\": 12000.0, \"status\": \"APPROVED\", \"typeCredit\": \"PERSONNEL\"}"))),              responses = {                      @ApiResponse(responseCode = "200", description = "Credit updated successfully"),                      @ApiResponse(responseCode = "400", description = "Invalid input"),                      @ApiResponse(responseCode = "404", description = "Credit not found")              })      public ResponseEntity<CreditDto> updateCredit(@PathVariable Long id, @Valid @org.springframework.web.bind.annotation.RequestBody CreditDto creditDto) {          CreditDto updatedCredit = creditService.updateCredit(id, creditDto);          return ResponseEntity.ok(updatedCredit);      }      @DeleteMapping("/{id}")      @Operation(summary = "Delete a credit by ID",              description = "Deletes a specific credit by its ID.",              responses = {                      @ApiResponse(responseCode = "204", description = "Credit deleted successfully"),                      @ApiResponse(responseCode = "404", description = "Credit not found")              })      public ResponseEntity<Void> deleteCredit(@PathVariable Long id) {          creditService.deleteCredit(id);          return ResponseEntity.noContent().build();      }  } |

### Documentation Swagger:

Pour documenter les API, le projet utilise **Springdoc OpenAPI** (Swagger v3).

|  |
| --- |
| @Configuration  public class OpenApiConfig {      @Bean      public OpenAPI customOpenAPI() {          return new OpenAPI()                  .info(new Info()                          .title("Banking System API")                          .description("RESTful API for managing clients, credits, and repayments in a banking system")                          .version("1.0.0")                          .contact(new Contact()                                  .name("Banking System Team")                                  .email("contact@bank-example.com")                                  .url("https://www.bank-example.com"))                          .license(new License()                                  .name("MIT License")                                  .url("https://opensource.org/licenses/MIT")))                  .externalDocs(new ExternalDocumentation()                          .description("Banking System Documentation")                          .url("https://bank-example.com/docs"))                  .tags(Arrays.asList(                          new Tag().name("Client Management").description("Operations about clients"),                          new Tag().name("Credit Management").description("Operations about credits"),                          new Tag().name("Remboursement Management").description("Operations about repayments"),                          new Tag().name("Authentication").description("Operations for authentication and user management")                  ))                  .addSecurityItem(new SecurityRequirement().addList("bearerAuth"))                  .components(new Components()                          .addSecuritySchemes("bearerAuth", new SecurityScheme()                                  .name("bearerAuth")                                  .type(SecurityScheme.Type.HTTP)                                  .scheme("bearer")                                  .bearerFormat("JWT")                                  .description("JWT Authorization header using the Bearer scheme. Example: \"Authorization: Bearer {token}\"")));      }  } |

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Couche Sécurité (Spring Security + JWT):

La sécurité de l’application est assurée via Spring Security combinée avec JWT (JSON Web Token). Cela permet d’authentifier les utilisateurs et de gérer les autorisations d’accès aux différentes ressources exposées par les APIs.

### Gestion des utilisateurs et rôles :

entities/User.java

|  |
| --- |
| @Entity @Table(name = "users",  uniqueConstraints = {  @UniqueConstraint(columnNames = "username"),  @UniqueConstraint(columnNames = "email")  }) @Getter @Setter @NoArgsConstructor @AllArgsConstructor @ToString(exclude = {"client"}) @EqualsAndHashCode(exclude = {"client"}) public class User {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Long id;   @NotBlank  @Size(max = 20)  private String username;   @NotBlank  @Size(max = 50)  @Email  private String email;   @NotBlank  @Size(max = 120)  private String password;   @ManyToMany(fetch = FetchType.EAGER)  @JoinTable(name = "user\_roles",  joinColumns = @JoinColumn(name = "user\_id"),  inverseJoinColumns = @JoinColumn(name = "role\_id"))  private Set<Role> roles = new HashSet<>();   // Optional: Link to Client entity  @OneToOne(mappedBy = "user")  private Client client;   public User(String username, String email, String password) {  this.username = username;  this.email = email;  this.password = password;  } } |

entities/Role.java :

|  |
| --- |
| @Entity @Table(name = "roles") @Data @NoArgsConstructor @AllArgsConstructor public class Role {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Integer id;   @Enumerated(EnumType.STRING)  @Column(length = 20)  private ERole name;   public Role(ERole name) {  this.name = name;  } } |

### Authentification avec JWT:

**Login / Register**

* **Contrôleur :** AuthController.java
* **Endpoints :**
  + POST /api/auth/login : Authentification
  + POST /api/auth/register : Enregistrement utilisateur
* **Payloads :**
  + LoginRequestDTO : contient email et mot de passe
  + RegisterRequestDTO : contient nom, email, mot de passe, rôle(s)

|  |
| --- |
| * @CrossOrigin(origins = "\*", maxAge = 3600) @RestController @RequestMapping("/api/v1/auth") @Tag(name = "Authentication", description = "API for user authentication and registration") public class AuthController {  private final AuthenticationManager authenticationManager;  private final UserRepository userRepository;  private final RoleRepository roleRepository;  private final ClientRepository clientRepository;  private final PasswordEncoder encoder;  private final JwtUtils jwtUtils;   public AuthController(AuthenticationManager authenticationManager,   UserRepository userRepository,   RoleRepository roleRepository,  ClientRepository clientRepository,  PasswordEncoder encoder,   JwtUtils jwtUtils) {  this.authenticationManager = authenticationManager;  this.userRepository = userRepository;  this.roleRepository = roleRepository;  this.clientRepository = clientRepository;  this.encoder = encoder;  this.jwtUtils = jwtUtils;  }   @PostMapping("/signin")  @Operation(  summary = "Authenticate user",  description = "Authenticates a user with username and password, returns JWT token", requestBody = @io.swagger.v3.oas.annotations.parameters.RequestBody(  description = "Login credentials",  content = @Content(schema = @Schema(implementation = LoginRequest.class))  ),   responses = {  @ApiResponse(responseCode = "200", description = "Authentication successful",  content = @Content(schema = @Schema(implementation = JwtResponse.class))),  @ApiResponse(responseCode = "401", description = "Authentication failed")  },  security = {} // No security required for this endpoint  )  public ResponseEntity<?> authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {  Authentication authentication = authenticationManager.authenticate(  new UsernamePasswordAuthenticationToken(loginRequest.getUsername(), loginRequest.getPassword()));   SecurityContextHolder.getContext().setAuthentication(authentication);  String jwt = jwtUtils.generateJwtToken(authentication);    UserDetailsImpl userDetails = (UserDetailsImpl) authentication.getPrincipal();  List<String> roles = userDetails.getAuthorities().stream()  .map(item -> item.getAuthority())  .collect(Collectors.toList());   return ResponseEntity.ok(new JwtResponse(jwt,  userDetails.getId(),   userDetails.getUsername(),   userDetails.getEmail(),   roles));  }   @PostMapping("/signup")  @Operation(  summary = "Register new user",  description = "Registers a new user with username, email, password and roles",  requestBody = @io.swagger.v3.oas.annotations.parameters.RequestBody(  description = "Registration details",  content = @Content(schema = @Schema(implementation = SignupRequest.class))  ), responses = {  @ApiResponse(responseCode = "200", description = "Registration successful",  content = @Content(schema = @Schema(implementation = MessageResponse.class))),  @ApiResponse(responseCode = "400", description = "Invalid input")  },  security = {} // No security required for this endpoint  )  public ResponseEntity<?> registerUser(@Valid @RequestBody SignupRequest signUpRequest) {  if (userRepository.existsByUsername(signUpRequest.getUsername())) {  return ResponseEntity  .badRequest()  .body(new MessageResponse("Error: Username is already taken!"));  }   if (userRepository.existsByEmail(signUpRequest.getEmail())) {  return ResponseEntity  .badRequest()  .body(new MessageResponse("Error: Email is already in use!"));  }   // Create new user's account  User user = new User(signUpRequest.getUsername(),  signUpRequest.getEmail(),  encoder.encode(signUpRequest.getPassword()));   Set<String> strRoles = signUpRequest.getRoles();  Set<Role> roles = new HashSet<>();   if (strRoles == null || strRoles.isEmpty()) {  Role clientRole = roleRepository.findByName(ERole.ROLE\_CLIENT)  .orElseThrow(() -> new RuntimeException("Error: Role is not found."));  roles.add(clientRole);  } else {  strRoles.forEach(role -> {  switch (role) {  case "admin":  Role adminRole = roleRepository.findByName(ERole.ROLE\_ADMIN)  .orElseThrow(() -> new RuntimeException("Error: Role is not found."));  roles.add(adminRole);  break;  case "employe":  Role modRole = roleRepository.findByName(ERole.ROLE\_EMPLOYE)  .orElseThrow(() -> new RuntimeException("Error: Role is not found."));  roles.add(modRole);  break;  default:  Role userRole = roleRepository.findByName(ERole.ROLE\_CLIENT)  .orElseThrow(() -> new RuntimeException("Error: Role is not found."));  roles.add(userRole);  }  });  }   user.setRoles(roles);  User savedUser = userRepository.save(user);    // Create client profile if required  if (signUpRequest.getClientInfo() != null && roles.stream().anyMatch(r -> r.getName() == ERole.ROLE\_CLIENT)) {  Client client = new Client();  client.setName(signUpRequest.getClientInfo().getName() != null ?   signUpRequest.getClientInfo().getName() :   signUpRequest.getUsername());  client.setEmail(signUpRequest.getEmail());  client.setUser(savedUser);  clientRepository.save(client);  }   return ResponseEntity.ok(new MessageResponse("User registered successfully!"));  } } |

### Configuration de sécurité:

**Classe principale : SecurityConfig.java**

* Configuration des endpoints publics/privés
* Intégration du filtre JWT (JwtAuthenticationFilter)
* Gestion des exceptions
* Activation du CORS, désactivation du CSRF pour REST

|  |
| --- |
| * @Configuration @EnableMethodSecurity public class WebSecurityConfig {    private final UserDetailsServiceImpl userDetailsService;  private final AuthEntryPointJwt unauthorizedHandler;    public WebSecurityConfig(UserDetailsServiceImpl userDetailsService, AuthEntryPointJwt unauthorizedHandler) {  this.userDetailsService = userDetailsService;  this.unauthorizedHandler = unauthorizedHandler;  }    @Bean  public AuthTokenFilter authenticationJwtTokenFilter() {  return new AuthTokenFilter();  }    @Bean  public DaoAuthenticationProvider authenticationProvider() {  DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();    authProvider.setUserDetailsService(userDetailsService);  authProvider.setPasswordEncoder(passwordEncoder());    return authProvider;  }    @Bean  public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {  return authConfig.getAuthenticationManager();  }    @Bean  public PasswordEncoder passwordEncoder() {  return new BCryptPasswordEncoder();  }    @Bean  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  http.csrf(AbstractHttpConfigurer::disable)  .exceptionHandling(exception -> exception.authenticationEntryPoint(unauthorizedHandler))  .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))  .authorizeHttpRequests(auth ->   auth.requestMatchers("/api/v1/auth/\*\*").permitAll()  .requestMatchers("/v3/api-docs/\*\*").permitAll()  .requestMatchers("/swagger-ui/\*\*").permitAll()  .requestMatchers("/swagger-ui.html").permitAll()  .anyRequest().authenticated()  );    http.authenticationProvider(authenticationProvider());    http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);    return http.build();  } } |

### Services de sécurité

Les services de sécurité assurent la gestion des utilisateurs, la vérification des identifiants, la génération et la validation des tokens JWT. Voici les éléments principaux :

JwtService :

Service responsable de la génération, la validation et l'extraction des informations depuis les tokens JWT.

|  |
| --- |
| @Component public class JwtUtils {  private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);   @Value("${jwt.secret}")  private String jwtSecret;   @Value("${jwt.expirationMs}")  private int jwtExpirationMs;   public String generateJwtToken(Authentication authentication) {  UserDetailsImpl userPrincipal = (UserDetailsImpl) authentication.getPrincipal();   return Jwts.builder()  .setSubject((userPrincipal.getUsername()))  .setIssuedAt(new Date())  .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))  .signWith(key(), SignatureAlgorithm.HS256)  .compact();  }   private Key key() {  return Keys.hmacShaKeyFor(Decoders.BASE64.decode(jwtSecret));  }   public String getUserNameFromJwtToken(String token) {  return Jwts.parserBuilder().setSigningKey(key()).build()  .parseClaimsJws(token).getBody().getSubject();  }   public boolean validateJwtToken(String authToken) {  try {  Jwts.parserBuilder().setSigningKey(key()).build().parse(authToken);  return true;  } catch (MalformedJwtException e) {  logger.error("Invalid JWT token: {}", e.getMessage());  } catch (ExpiredJwtException e) {  logger.error("JWT token is expired: {}", e.getMessage());  } catch (UnsupportedJwtException e) {  logger.error("JWT token is unsupported: {}", e.getMessage());  } catch (IllegalArgumentException e) {  logger.error("JWT claims string is empty: {}", e.getMessage());  }   return false;  } } |

JwtAuthenticationFilter:

Filtre qui intercepte chaque requête HTTP pour extraire le token JWT, le valider, puis configurer le contexte de sécurité Spring.

|  |
| --- |
| public class AuthTokenFilter extends OncePerRequestFilter {  @Autowired  private JwtUtils jwtUtils;   @Autowired  private UserDetailsServiceImpl 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));   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);  }   return null;  } } |

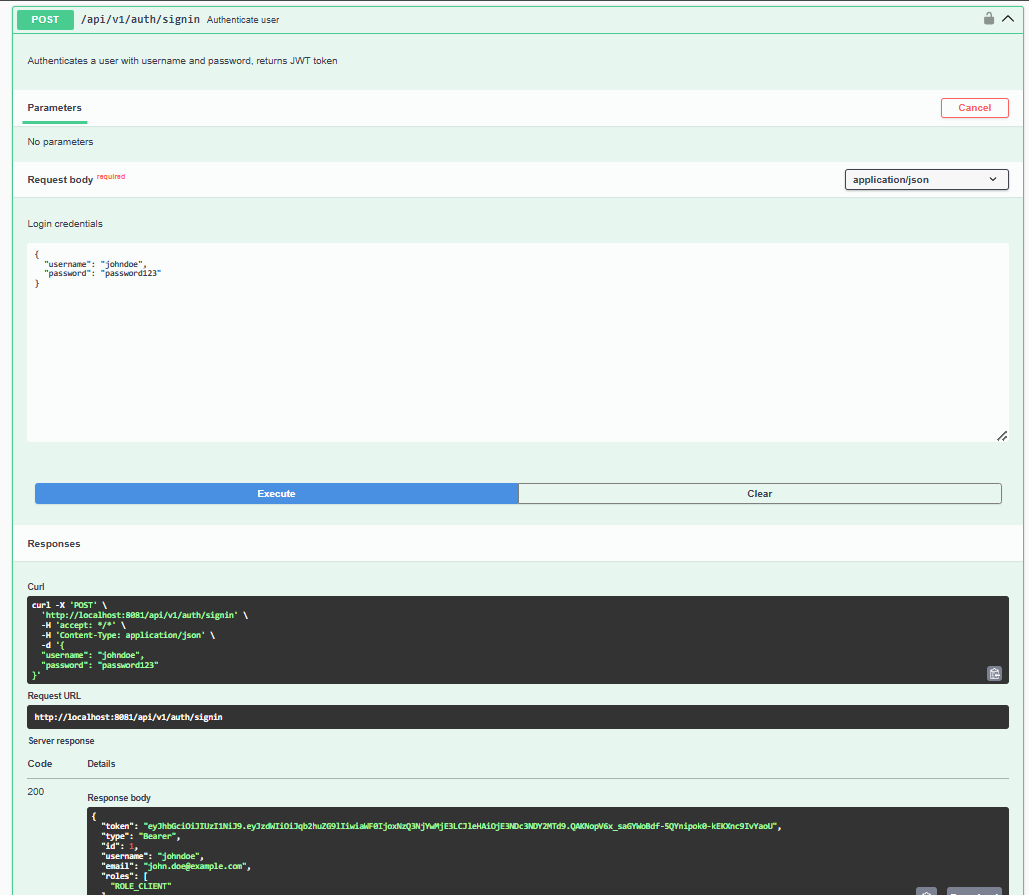


Figure 4 Login swagger impl

Frontend

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

