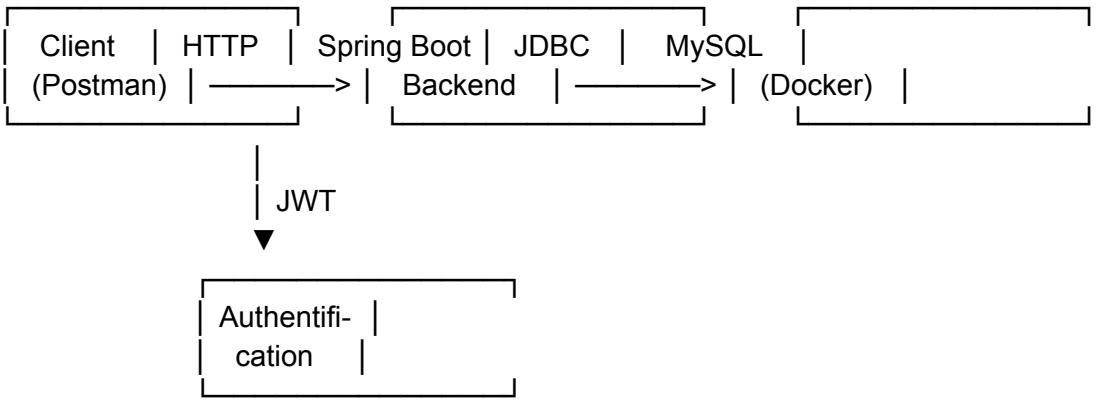
Documentation Complète - Système d'Authentification JWT avec Spring Boot

🎯 Vue d'ensemble du projet

Ce document détaille l'implémentation complète d'un système d'authentification JWT (JSON Web Token) pour l'application **MSSP HealthCheck** utilisant Spring Boot, MySQL et Docker.

📋 Table des matières

🏗️ Architecture du projet



Flux d'authentification

1. **Register** : Création d'un compte utilisateur

2. **Login** : Authentification et génération du token JWT
3. **Protected Routes** : Accès aux ressources protégées avec le token

---

Technologies utilisées

Technologie	Version	Utilisation
Java	24.0.1	Langage de programmation
Spring Boot	3.4.11	Framework backend
Spring Security	6.4.12	Sécurité et authentification
Spring Data JPA	3.4.11	Accès aux données
Hibernate	6.6.33	ORM (Object-Relational Mapping)
MySQL	8.0.44	Base de données
Docker	Latest	Conteneurisation MySQL
JWT (JJWT)	0.12.3	Génération et validation des tokens
Lombok	1.18.42	Réduction du code boilerplate
Maven	3.x	Gestion des dépendances

⚙️ Configuration de l'environnement

Prérequis

- Java JDK 17 ou supérieur
- IntelliJ IDEA (ou tout autre IDE Java)
- Docker Desktop
- Postman (pour les tests)
- Maven

Installation de Docker et MySQL

1. Télécharger et installer Docker Desktop

Windows :

- Téléchargez depuis : https://www.docker.com/products/docker-desktop/
- Installez et démarrez Docker Desktop

Vérification :

```
docker --version
```

2. Créer le conteneur MySQL

```
docker run -d \
  --name mssp-mysql \
  -e MYSQL_ROOT_PASSWORD=root123 \
  -e MYSQL_DATABASE=mssp_healthcheck \
  -e MYSQL_USER=mssp_user \
  -e MYSQL_PASSWORD=mssp_pass123 \
  -p 3307:3306 \
  mysql:8.0.44
```

Explication des paramètres :

- `-d` : Mode détaché (background)
- `--name mssp-mysql` : Nom du conteneur
- `-e` : Variables d'environnement
- `-p 3307:3306` : Map le port 3306 du conteneur vers le port 3307 de l'hôte
- `mysql:8.0.44` : Image MySQL version 8.0.44

3. Vérifier que MySQL fonctionne

```
docker ps
```

Vous devriez voir le conteneur `mssp-mysql` avec le statut "Up".

4. Se connecter à MySQL

docker exec -it mssp-mysql mysql -u root -proot123

Une fois connecté :

SHOW DATABASES;
USE mssp_healthcheck;
SHOW TABLES;
EXIT;

📁 Structure du projet

```
healthcheck/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── mssp/
│   │   │           └── healthcheck/
│   │   │               ├── MsspHealthCheckApplication.java
│   │   │               ├── controller/
│   │   │               │   ├── AuthController.java
│   │   │               │   └── TestController.java
│   │   │               ├── dto/
│   │   │               │   ├── ApiResponse.java
│   │   │               │   ├── AuthResponse.java
│   │   │               │   ├── LoginRequest.java
│   │   │               │   └── RegisterRequest.java
│   │   │               ├── entity/
│   │   │               │   ├── Role.java
│   │   │               │   └── User.java
│   │   │               ├── repository/
│   │   │               │   └── UserRepository.java
│   │   │               ├── security/
│   │   │               │   ├── CustomUserDetailsService.java
│   │   │               │   ├── JwtAuthenticationFilter.java
│   │   │               │   ├── JwtUtil.java
│   │   │               │   └── SecurityConfig.java
│   │   │               └── service/
│   │   │                   └── AuthService.java
│   │   └── resources/
│   │       └── application.properties
│   └── test/
├── pom.xml
└── README.md
```

🗄️ Configuration de la base de données

Fichier : `application.properties`

```
# ===================================
# Application Configuration
# ===================================
spring.application.name=MSSP-HealthCheck
server.port=8080

# ===================================
# MySQL Database Configuration
# ===================================
spring.datasource.url=jdbc:mysql://localhost:3307/mssp_healthcheck?createDatabaseIfNotExist=true
spring.datasource.username=mssp_user
spring.datasource.password=mssp_pass123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# ===================================
# JPA / Hibernate Configuration
# ===================================
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.properties.hibernate.jdbc.time_zone=UTC

# ===================================
# JWT Configuration
# ===================================
jwt.secret=404E635266556A586E3272357538782F413F4428472B4B6250645367566B5970
jwt.expiration=86400000

# ===================================
# Logging Configuration
# ===================================
logging.level.org.springframework.security=DEBUG
logging.level.com.mssp.healthcheck=DEBUG
```

Explication des propriétés clés :

- **spring.datasource.url** : URL de connexion MySQL (port 3307)
- **spring.jpa.hibernate.ddl-auto=update** : Crée/met à jour automatiquement les tables
- **spring.jpa.show-sql=true** : Affiche les requêtes SQL dans les logs

- **jwt.secret** : Clé secrète pour signer les tokens JWT (256 bits en hexadécimal)
- **jwt.expiration** : Durée de validité du token (24h = 86400000 ms)

🔧 Implémentation détaillée

1. Entités (Entity Layer)

Role.java - Énumération des rôles

```
package com.mssp.healthcheck.entity;

public enum Role {
    ADMIN,      // Administrateur système
    TECHNICIAN, // Technicien MSSP
    MANAGER     // Manager/Chef de projet
}
```

User.java - Entité utilisateur

```
package com.mssp.healthcheck.entity;

import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.time.LocalDateTime;
import java.util.Collection;
import java.util.List;

@Entity
@Table(name = "users", uniqueConstraints = {
    @UniqueConstraint(columnNames = "username"),
    @UniqueConstraint(columnNames = "email")
})
@EntityListeners(AuditingEntityListener.class)
public class User implements UserDetails {

    @Id
```

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;

@NotBlank(message = "Le nom d'utilisateur est obligatoire")
@Size(min = 3, max = 50)
@Column(nullable = false, unique = true, length = 50)
private String username;

@NotBlank(message = "L'email est obligatoire")
@Email(message = "Format d'email invalide")
@Column(nullable = false, unique = true, length = 100)
private String email;

@NotBlank(message = "Le mot de passe est obligatoire")
@Column(nullable = false, length = 255)
private String password;

@NotBlank(message = "Le prénom est obligatoire")
@Column(name = "first_name", nullable = false, length = 50)
private String firstName;

@NotBlank(message = "Le nom est obligatoire")
@Column(name = "last_name", nullable = false, length = 50)
private String lastName;

@Enumerated(EnumType.STRING)
@Column(nullable = false, length = 20)
private Role role;

@Column(name = "phone_number", length = 20)
private String phoneNumber;

@Column(name = "is_active", nullable = false)
private Boolean isActive = true;

@Column(name = "last_login")
private LocalDateTime lastLogin;

@CreatedDate
@Column(name = "created_at", nullable = false, updatable = false)
private LocalDateTime createdAt;

@LastModifiedDate
@Column(name = "updated_at")
private LocalDateTime updatedAt;

// Constructeurs
public User() {
```

```
}

public User(Long id, String username, String email, String password, String firstName,
        String lastName, Role role, String phoneNumber, Boolean isActive,
        LocalDateTime lastLogin, LocalDateTime createdAt, LocalDateTime updatedAt) {
    this.id = id;
    this.username = username;
    this.email = email;
    this.password = password;
    this.firstName = firstName;
    this.lastName = lastName;
    this.role = role;
    this.phoneNumber = phoneNumber;
    this.isActive = isActive;
    this.lastLogin = lastLogin;
    this.createdAt = createdAt;
    this.updatedAt = updatedAt;
}

// Builder Pattern
public static Builder builder() {
    return new Builder();
}

public static class Builder {
    private Long id;
    private String username;
    private String email;
    private String password;
    private String firstName;
    private String lastName;
    private Role role;
    private String phoneNumber;
    private Boolean isActive = true;
    private LocalDateTime lastLogin;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;

    public Builder id(Long id) {
        this.id = id;
        return this;
    }

    public Builder username(String username) {
        this.username = username;
        return this;
    }
```

```
public Builder email(String email) {
    this.email = email;
    return this;
}

public Builder password(String password) {
    this.password = password;
    return this;
}

public Builder firstName(String firstName) {
    this.firstName = firstName;
    return this;
}

public Builder lastName(String lastName) {
    this.lastName = lastName;
    return this;
}

public Builder role(Role role) {
    this.role = role;
    return this;
}

public Builder phoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
    return this;
}

public Builder isActive(Boolean isActive) {
    this.isActive = isActive;
    return this;
}

public Builder lastLogin(LocalDateTime lastLogin) {
    this.lastLogin = lastLogin;
    return this;
}

public Builder createdAt(LocalDateTime createdAt) {
    this.createdAt = createdAt;
    return this;
}

public Builder updatedAt(LocalDateTime updatedAt) {
    this.updatedAt = updatedAt;
    return this;
```

```
    }

    public User build() {
        return new User(id, username, email, password, firstName, lastName,
                role, phoneNumber, isActive, lastLogin, createdAt, updatedAt);
    }
}

// Getters et Setters
public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

@Override
public String getUsername() {
    return username;
}

public void setUsername(String username) {
    this.username = username;
}

public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

@Override
public String getPassword() {
    return password;
}

public void setPassword(String password) {
    this.password = password;
}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
```

```
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public Role getRole() {
    return role;
}

public void setRole(Role role) {
    this.role = role;
}

public String getPhoneNumber() {
    return phoneNumber;
}

public void setPhoneNumber(String phoneNumber) {
    this.phoneNumber = phoneNumber;
}

public Boolean getIsActive() {
    return isActive;
}

public void setIsActive(Boolean isActive) {
    this.isActive = isActive;
}

public LocalDateTime getLastLogin() {
    return lastLogin;
}

public void setLastLogin(LocalDateTime lastLogin) {
    this.lastLogin = lastLogin;
}

public LocalDateTime getCreatedAt() {
    return createdAt;
}

public void setCreatedAt(LocalDateTime createdAt) {
```

```
        this.createdAt = createdAt;
    }

    public LocalDateTime getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(LocalDateTime updatedAt) {
        this.updatedAt = updatedAt;
    }

    // UserDetails Implementation
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return List.of(new SimpleGrantedAuthority("ROLE_" + role.name()));
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return isActive;
    }

    // Méthode utilitaire
    public String getFullName() {
        return firstName + " " + lastName;
    }
}
```

Points clés :

- Implémente `UserDetails` pour l'intégration avec Spring Security
- Utilise JPA pour le mapping objet-relationnel

- Annotations de validation (`@NotBlank`, `@Email`, etc.)
- Gestion automatique des dates de création/modification avec `@EntityListeners`

---

2. Repository Layer

UserRepository.java

```
package com.mssp.healthcheck.repository;

import com.mssp.healthcheck.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByUsername(String username);
    Optional<User> findByEmail(String email);
    boolean existsByUsername(String username);
    boolean existsByEmail(String email);
}
```

Méthodes fournies automatiquement par JpaRepository :

- `save(User user)` : Créer ou mettre à jour un utilisateur
- `findById(Long id)` : Trouver un utilisateur par ID
- `findAll()` : Récupérer tous les utilisateurs
- `delete(User user)` : Supprimer un utilisateur
- `count()` : Compter le nombre d'utilisateurs

---

3. DTOs (Data Transfer Objects)

LoginRequest.java

```
package com.mssp.healthcheck.dto;

import jakarta.validation.constraints.NotBlank;

public class LoginRequest {

    @NotBlank(message = "Le nom d'utilisateur est obligatoire")
    private String username;
```

```
    @NotBlank(message = "Le mot de passe est obligatoire")
    private String password;

    public LoginRequest() {
    }

    public LoginRequest(String username, String password) {
        this.username = username;
        this.password = password;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

RegisterRequest.java

```
package com.mssp.healthcheck.dto;

import com.mssp.healthcheck.entity.Role;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;

public class RegisterRequest {

    @NotBlank(message = "Le nom d'utilisateur est obligatoire")
    @Size(min = 3, max = 50)
    private String username;

    @NotBlank(message = "L'email est obligatoire")
    @Email(message = "Format d'email invalide")
    private String email;
```

```
@NotBlank(message = "Le mot de passe est obligatoire")
@Size(min = 6, message = "Le mot de passe doit contenir au moins 6 caractères")
private String password;

@NotBlank(message = "Le prénom est obligatoire")
private String firstName;

@NotBlank(message = "Le nom est obligatoire")
private String lastName;

@NotNull(message = "Le rôle est obligatoire")
private Role role;

private String phoneNumber;

// Constructeurs
public RegisterRequest() {
}

public RegisterRequest(String username, String email, String password, String firstName,
                       String lastName, Role role, String phoneNumber) {
    this.username = username;
    this.email = email;
    this.password = password;
    this.firstName = firstName;
    this.lastName = lastName;
    this.role = role;
    this.phoneNumber = phoneNumber;
}

// Getters
public String getUsername() {
    return username;
}

public String getEmail() {
    return email;
}

public String getPassword() {
    return password;
}

public String getFirstName() {
    return firstName;
}

public String getLastName() {
```

```
        return lastName;
    }

    public Role getRole() {
        return role;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    // Setters
    public void setUsername(String username) {
        this.username = username;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public void setRole(Role role) {
        this.role = role;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }
}
```

AuthResponse.java

```
package com.mssp.healthcheck.dto;

import com.mssp.healthcheck.entity.Role;

public class AuthResponse {
```

```
private String token;
private String type;
private Long id;
private String username;
private String email;
private String fullName;
private Role role;

public AuthResponse() {
}

public AuthResponse(String token, String type, Long id, String username,
                    String email, String fullName, Role role) {
    this.token = token;
    this.type = type;
    this.id = id;
    this.username = username;
    this.email = email;
    this.fullName = fullName;
    this.role = role;
}

public static Builder builder() {
    return new Builder();
}

public static class Builder {
    private String token;
    private String type;
    private Long id;
    private String username;
    private String email;
    private String fullName;
    private Role role;

    public Builder token(String token) {
        this.token = token;
        return this;
    }

    public Builder type(String type) {
        this.type = type;
        return this;
    }

    public Builder id(Long id) {
        this.id = id;
        return this;
```

```
    }

    public Builder username(String username) {
        this.username = username;
        return this;
    }

    public Builder email(String email) {
        this.email = email;
        return this;
    }

    public Builder fullName(String fullName) {
        this.fullName = fullName;
        return this;
    }

    public Builder role(Role role) {
        this.role = role;
        return this;
    }

    public AuthResponse build() {
        return new AuthResponse(token, type, id, username, email, fullName, role);
    }
}

// Getters et Setters
public String getToken() {
    return token;
}

public void setToken(String token) {
    this.token = token;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public Long getId() {
    return id;
}
```

```
    public void setId(Long id) {
        this.id = id;
    }

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public Role getRole() {
        return role;
    }

    public void setRole(Role role) {
        this.role = role;
    }
}
```

ApiResponse.java

```
package com.mssp.healthcheck.dto;

public class ApiResponse {
    private boolean success;
    private String message;

    public ApiResponse() {
    }
```

```
    public ApiResponse(boolean success, String message) {
        this.success = success;
        this.message = message;
    }

    public static ApiResponse success(String message) {
        return new ApiResponse(true, message);
    }

    public static ApiResponse error(String message) {
        return new ApiResponse(false, message);
    }

    public boolean isSuccess() {
        return success;
    }

    public void setSuccess(boolean success) {
        this.success = success;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

---

4. Security Layer

JwtUtil.java - Utilitaire JWT

```
package com.mssp.healthcheck.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.io.Decoders;
import io.jsonwebtoken.security.Keys;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.security.Key;
import java.util.Date;
```

```
import java.util.HashMap;
import java.util.Map;
import java.util.function.Function;

@Component
public class JwtUtil {

    @Value("${jwt.secret}")
    private String secret;

    @Value("${jwt.expiration}")
    private Long expiration;

    public String extractUsername(String token) {
        return extractClaim(token, Claims::getSubject);
    }

    public Date extractExpiration(String token) {
        return extractClaim(token, Claims::getExpiration);
    }

    public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {
        final Claims claims = extractAllClaims(token);
        return claimsResolver.apply(claims);
    }

    private Claims extractAllClaims(String token) {
        return Jwts
                .parserBuilder()
                .setSigningKey(getSignKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
    }

    private Boolean isTokenExpired(String token) {
        return extractExpiration(token).before(new Date());
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }

    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return createToken(claims, userDetails.getUsername());
    }
```

```
    private String createToken(Map<String, Object> claims, String subject) {
        return Jwts.builder()
                .setClaims(claims)
                .setSubject(subject)
                .setIssuedAt(new Date(System.currentTimeMillis()))
                .setExpiration(new Date(System.currentTimeMillis() + expiration))
                .signWith(getSignKey(), SignatureAlgorithm.HS256)
                .compact();
    }

    private Key getSignKey() {
        byte[] keyBytes = Decoders.BASE64.decode(secret);
        return Keys.hmacShaKeyFor(keyBytes);
    }
}
```

Fonctionnalités :

- Génération de tokens JWT
- Extraction des informations (username, expiration)
- Validation des tokens

CustomUserDetailsService.java

```
package com.mssp.healthcheck.security;

import com.mssp.healthcheck.repository.UserRepository;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    public CustomUserDetailsService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        return userRepository.findByUsername(username)
                .orElseThrow(() -> new UsernameNotFoundException("Utilisateur non trouvé : " +
username));
```

```
    }
}
```

JwtAuthenticationFilter.java

```
package com.mssp.healthcheck.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;
    private final UserDetailsService userDetailsService;

    public JwtAuthenticationFilter(JwtUtil jwtUtil, UserDetailsService userDetailsService) {
        this.jwtUtil = jwtUtil;
        this.userDetailsService = userDetailsService;
    }

    @Override
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain filterChain
    ) throws ServletException, IOException {

        final String authHeader = request.getHeader("Authorization");
        final String jwt;
        final String username;

        // Vérifier si le header Authorization existe et commence par "Bearer "
        if (authHeader == null || !authHeader.startsWith("Bearer ")) {
            filterChain.doFilter(request, response);
            return;
        }
```

```
        // Extraire le token
        jwt = authHeader.substring(7);
        username = jwtUtil.extractUsername(jwt);

        // Authentifier si l'utilisateur n'est pas déjà authentifié
        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null)
{
            UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

            if (jwtUtil.validateToken(jwt, userDetails)) {
                UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                        userDetails,
                        null,
                        userDetails.getAuthorities()
                );
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

SecurityConfig.java

```
package com.mssp.healthcheck.security;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationC
onfiguration;
import
org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
```

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    private final UserDetailsService userDetailsService;

    public SecurityConfig(UserDetailsService userDetailsService) {
        this.userDetailsService = userDetailsService;
    }

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        return http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .anyRequest().permitAll()  // TEMPORAIREMENT : Tout est autorisé
                )
                .sessionManagement(session -> session
                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                )
                .build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config)
throws Exception {
        return config.getAuthenticationManager();
    }
}
```

Note : La sécurité est temporairement désactivée (`.anyRequest().permitAll()`) pour faciliter les tests initiaux.

---

5. Service Layer

AuthService.java

```
package com.mssp.healthcheck.service;

import com.mssp.healthcheck.dto.AuthResponse;
import com.mssp.healthcheck.dto.LoginRequest;
import com.mssp.healthcheck.dto.RegisterRequest;
import com.mssp.healthcheck.entity.User;
import com.mssp.healthcheck.repository.UserRepository;
import com.mssp.healthcheck.security.JwtUtil;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDateTime;

@Service
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtUtil jwtUtil;
    private final AuthenticationManager authenticationManager;

    public AuthService(UserRepository userRepository, PasswordEncoder passwordEncoder,
                       JwtUtil jwtUtil, AuthenticationManager authenticationManager) {
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.jwtUtil = jwtUtil;
        this.authenticationManager = authenticationManager;
    }

    @Transactional
    public AuthResponse register(RegisterRequest request) {
        // Vérifier si le username existe déjà
        if (userRepository.existsByUsername(request.getUsername())) {
            throw new RuntimeException("Le nom d'utilisateur existe déjà");
        }

        // Vérifier si l'email existe déjà
        if (userRepository.existsByEmail(request.getEmail())) {
            throw new RuntimeException("L'email existe déjà");
        }
```

```
    // Créer le nouvel utilisateur
    User user = User.builder()
            .username(request.getUsername())
            .email(request.getEmail())
            .password(passwordEncoder.encode(request.getPassword()))
            .firstName(request.getFirstName())
            .lastName(request.getLastName())
            .role(request.getRole())
            .phoneNumber(request.getPhoneNumber())
            .isActive(true)
            .build();

    userRepository.save(user);

    // Générer le token JWT
    String token = jwtUtil.generateToken(user);

    // Retourner la réponse
    return AuthResponse.builder()
            .token(token)
            .type("Bearer")
            .id(user.getId())
            .username(user.getUsername())
            .email(user.getEmail())
            .fullName(user.getFullName())
            .role(user.getRole())
            .build();
}

@Transactional
public AuthResponse login(LoginRequest request) {
    // Authentifier l'utilisateur
    authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                    request.getUsername(),
                    request.getPassword()
            )
    );

    // Récupérer l'utilisateur
    User user = userRepository.findByUsername(request.getUsername())
            .orElseThrow(() -> new RuntimeException("Utilisateur non trouvé"));

    // Vérifier si l'utilisateur est actif
    if (!user.getIsActive()) {
        throw new RuntimeException("Compte désactivé");
    }
```

```
        // Mettre à jour la date de dernière connexion
        user.setLastLogin(LocalDateTime.now());
        userRepository.save(user);

        // Générer le token JWT
        String token = jwtUtil.generateToken(user);

        // Retourner la réponse
        return AuthResponse.builder()
                .token(token)
                .type("Bearer")
                .id(user.getId())
                .username(user.getUsername())
                .email(user.getEmail())
                .fullName(user.getFullName())
                .role(user.getRole())
                .build();
    }
}
```

Logique métier :

- **Register** : Vérifie l'unicité, hash le mot de passe, crée l'utilisateur et génère un token
- **Login** : Authentifie, met à jour la dernière connexion et génère un token

---

6. Controller Layer

AuthController.java

```
package com.mssp.healthcheck.controller;

import com.mssp.healthcheck.dto.ApiResponse;
import com.mssp.healthcheck.dto.AuthResponse;
import com.mssp.healthcheck.dto.LoginRequest;
import com.mssp.healthcheck.dto.RegisterRequest;
import com.mssp.healthcheck.service.AuthService;
import jakarta.validation.Valid;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
public class AuthController {
```

```
    private final AuthService authService;

    public AuthController(AuthService authService) {
        this.authService = authService;
    }

    /**
     * Endpoint d'inscription
     * POST /api/auth/register
     */
    @PostMapping("/register")
    public ResponseEntity<?> register(@Valid @RequestBody RegisterRequest request) {
        try {
            AuthResponse response = authService.register(request);
            return ResponseEntity.status(HttpStatus.CREATED).body(response);
        } catch (RuntimeException e) {
            return ResponseEntity.badRequest().body(ApiResponse.error(e.getMessage()));
        }
    }

    /**
     * Endpoint de connexion
     * POST /api/auth/login
     */
    @PostMapping("/login")
    public ResponseEntity<?> login(@Valid @RequestBody LoginRequest request) {
        try {
            AuthResponse response = authService.login(request);
            return ResponseEntity.ok(response);
        } catch (RuntimeException e) {
            return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                    .body(ApiResponse.error("Identifiants invalides"));
        }
    }

    /**
     * Endpoint de test (accessible sans authentification)
     * GET /api/auth/test
     */
    @GetMapping("/test")
    public ResponseEntity<ApiResponse> test() {
        return ResponseEntity.ok(ApiResponse.success("API d'authentification fonctionne !"));
    }
}
```

TestController.java

```
package com.mssp.healthcheck.controller;
```

```
import com.mssp.healthcheck.dto.ApiResponse;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/test")
public class TestController {

    /**
     * Endpoint accessible à tous les utilisateurs authentifiés
     */
    @GetMapping("/user")
    public ResponseEntity<ApiResponse> testUser() {
        return ResponseEntity.ok(ApiResponse.success("Accès utilisateur autorisé !"));
    }

    /**
     * Endpoint accessible uniquement aux ADMIN
     */
    @GetMapping("/admin")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<ApiResponse> testAdmin() {
        return ResponseEntity.ok(ApiResponse.success("Accès admin autorisé !"));
    }

    /**
     * Endpoint accessible aux ADMIN et TECHNICIAN
     */
    @GetMapping("/tech")
    @PreAuthorize("hasAnyRole('ADMIN', 'TECHNICIAN')")
    public ResponseEntity<ApiResponse> testTech() {
        return ResponseEntity.ok(ApiResponse.success("Accès technicien autorisé !"));
    }
}
```

---

7. Classe principale

MsspHealthCheckApplication.java

```
package com.mssp.healthcheck;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication
@EnableJpaAuditing
@EnableJpaRepositories(basePackages = "com.mssp.healthcheck.repository")
public class MsspHealthCheckApplication {

    public static void main(String[] args) {
        SpringApplication.run(MsspHealthCheckApplication.class, args);
    }
}
```

🧪 Tests avec Postman

Configuration de Postman

1. Téléchargez et installez Postman : https://www.postman.com/downloads/
2. Créez une nouvelle collection "MSSP HealthCheck"
3. Configurez les variables d'environnement :
    - `base_url` : `http://localhost:8080`

Test 1 : Vérification de l'API

Méthode : `GET`
URL : `{{base_url}}/api/auth/test`
Headers : Aucun

Résultat attendu : 200 OK

```
{
    "success": true,
    "message": "API d'authentification fonctionne !"
}
```

Test 2 : Créer un utilisateur (Register)

Méthode : `POST`
URL : `{{base_url}}/api/auth/register`
Headers :

Content-Type: application/json

Body (raw JSON) :

```
{
   "username": "admin",
   "email": "admin@mssp.com",
   "password": "Admin123",
   "firstName": "Admin",
   "lastName": "MSSP",
   "role": "ADMIN",
   "phoneNumber": "+212600000000"
}
```

Résultat attendu : 201 Created


```
{
   "token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImlhdCI6MTcwMDAwMDAwMCwiZXhwIjoxNz
AwMDg2NDAwfQ.xxx",
   "type": "Bearer",
   "id": 1,
   "username": "admin",
   "email": "admin@mssp.com",
   "fullName": "Admin MSSP",
   "role": "ADMIN"
}
```


Enregistrez le token retourné pour les tests suivants !

Test 3 : Se connecter (Login)

Méthode : `POST`
URL : `{{base_url}}/api/auth/login`
Headers :

Content-Type: application/json

Body (raw JSON) :

```
{
   "username": "admin",
   "password": "Admin123"
}
```

Résultat attendu : 200 OK


```
{
   "token":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbiIsImlhdCI6MTcwMDAwMDAwMCwiZXhwIjoxNzAwMDg2NDAwfQ.yyy",
   "type": "Bearer",
   "id": 1,
   "username": "admin",
   "email": "admin@mssp.com",
   "fullName": "Admin MSSP",
   "role": "ADMIN"
}
```

Test 4 : Route protégée (avec token)

Méthode : `GET`
URL : `{{base_url}}/api/test/user`
Headers :

Authorization: Bearer {VOTRE_TOKEN_ICI}

Résultat attendu : 200 OK


```
{
   "success": true,
   "message": "Accès utilisateur autorisé !"
}
```

Test 5 : Route admin uniquement

Méthode : `GET`
URL : `{{base_url}}/api/test/admin`
Headers :

Authorization: Bearer {TOKEN_ADMIN}

Résultat attendu avec ADMIN : 200 OK


```
{
   "success": true,
   "message": "Accès admin autorisé !"
```

```
}
```

Résultat attendu avec TECHNICIAN : 403 Forbidden

Test 6 : Cas d'erreurs

Username déjà existant

Body :

```
{
   "username": "admin",  // Déjà utilisé
   "email": "nouveau@mssp.com",
   "password": "Test123",
   "firstName": "Test",
   "lastName": "User",
   "role": "TECHNICIAN"
}
```

Résultat : 400 Bad Request

```
{
   "success": false,
   "message": "Le nom d'utilisateur existe déjà"
}
```

Identifiants incorrects (Login)

Body :

```
{
   "username": "admin",
   "password": "MauvaisMotDePasse"
}
```

Résultat : 401 Unauthorized

```
{
   "success": false,
   "message": "Identifiants invalides"
}
```

🔍 Vérification dans MySQL

Se connecter à MySQL

docker exec -it mssp-mysql mysql -u root -proot123

Commandes utiles

```
-- Utiliser la base de données
USE mssp_healthcheck;

-- Lister les tables
SHOW TABLES;

-- Voir la structure de la table users
DESCRIBE users;

-- Voir tous les utilisateurs
SELECT * FROM users;

-- Voir les utilisateurs avec des champs spécifiques
SELECT id, username, email, role, is_active, created_at FROM users;

-- Compter le nombre d'utilisateurs
SELECT COUNT(*) FROM users;

-- Trouver un utilisateur par username
SELECT * FROM users WHERE username = 'admin';

-- Désactiver un utilisateur
UPDATE users SET is_active = 0 WHERE username = 'admin';

-- Supprimer un utilisateur
DELETE FROM users WHERE username = 'admin';

-- Quitter MySQL
EXIT;
```

🐛 Dépannage

Problème 1 : Application ne démarre pas

Erreur : `Port 8080 already in use`

Solution :

Windows
netstat -ano | findstr :8080
taskkill /PID [PID_NUMBER] /F

# Ou changez le port dans application.properties
server.port=8081

Problème 2 : Connexion MySQL échoue

Erreur : `Communications link failure`

Solutions :

1. Vérifiez que Docker est démarré

Vérifiez que le conteneur MySQL tourne :
docker ps

2.

Si le conteneur n'existe pas, recréez-le :
docker run -d --name mssp-mysql -e MYSQL_ROOT_PASSWORD=root123 -e MYSQL_DATABASE=mssp_healthcheck -e MYSQL_USER=mssp_user -e MYSQL_PASSWORD=mssp_pass123 -p 3307:3306 mysql:8.0.44

3.

Problème 3 : UserRepository non trouvé

Erreur : `No qualifying bean of type 'UserRepository'`

Solution : Vérifiez que `MsspHealthCheckApplication.java` a l'annotation :

@EnableJpaRepositories(basePackages = "com.mssp.healthcheck.repository")

Problème 4 : 401 Unauthorized sur routes publiques

Solution : Vérifiez dans `SecurityConfig.java` que :

```
.authorizeHttpRequests(auth -> auth
    .anyRequest().permitAll()  // Temporairement
)
```

Problème 5 : Token JWT invalide

Causes possibles :

- Token expiré (validité de 24h)
- Mauvaise copie du token (espaces)
- Secret JWT modifié dans `application.properties`

Solution : Générez un nouveau token en vous reconnectant (POST `/api/auth/login`)

📊 Structure de la base de données

Table : users

Colonne	Type	Contraintes	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Identifiant unique
username	VARCHAR(50)	NOT NULL, UNIQUE	Nom d'utilisateur
email	VARCHAR(100)	NOT NULL, UNIQUE	Adresse email
password	VARCHAR(255)	NOT NULL	Mot de passe hashé (BCrypt)
first_name	VARCHAR(50)	NOT NULL	Prénom
last_name	VARCHAR(50)	NOT NULL	Nom
role	ENUM	NOT NULL	ADMIN, TECHNICIAN, MANAGER
phone_number	VARCHAR(20)	NULL	Numéro de téléphone
is_active	BOOLEAN	NOT NULL, DEFAULT TRUE	Compte actif/inactif

last_login	DATETIME	NULL	Dernière connexion
created_at	DATETIME	NOT NULL	Date de création
updated_at	DATETIME	NULL	Date de dernière modification

📝 Commandes utiles

Docker

```
# Lister les conteneurs en cours d'exécution
docker ps

# Lister tous les conteneurs (même arrêtés)
docker ps -a

# Arrêter un conteneur
docker stop mssp-mysql

# Démarrer un conteneur
docker start mssp-mysql

# Supprimer un conteneur
docker rm mssp-mysql

# Voir les logs d'un conteneur
docker logs mssp-mysql

# Voir les logs en temps réel
docker logs -f mssp-mysql
```

MySQL dans Docker

```
# Se connecter au conteneur MySQL
docker exec -it mssp-mysql mysql -u root -proot123

# Exécuter une commande SQL directement
docker exec -it mssp-mysql mysql -u root -proot123 -e "SELECT * FROM
mssp_healthcheck.users;"

# Exporter la base de données
docker exec mssp-mysql mysqldump -u root -proot123 mssp_healthcheck > backup.sql

# Importer une base de données
```

docker exec -i mssp-mysql mysql -u root -proot123 mssp_healthcheck < backup.sql

Maven

```
# Compiler le projet
mvn clean compile

# Compiler et exécuter les tests
mvn clean test

# Compiler et packager (créer le JAR)
mvn clean package

# Nettoyer le projet
mvn clean

# Exécuter l'application
mvn spring-boot:run
```

🔒 Bonnes pratiques de sécurité

1. Mots de passe

- ✅ Utilisation de BCrypt pour hasher les mots de passe
- ✅ Validation de la longueur minimale (6 caractères)
- ⚠️ À améliorer : Ajouter des règles de complexité (majuscules, chiffres, caractères spéciaux)

2. JWT

- ✅ Tokens signés avec HS256
- ✅ Expiration des tokens (24h)
- ✅ Secret stocké dans application.properties
- ⚠️ À améliorer : Utiliser des variables d'environnement pour les secrets en production

3. Protection CSRF

- ✅ CSRF désactivé (API stateless)
- ✅ Sessions désactivées (STATELESS)

4. Validation des données

- ✅ Annotations de validation sur les DTOs
- ✅ Contraintes d'unicité (username, email)

- ⚠️ À améliorer : Validation côté frontend également

5. Gestion des erreurs

- ✅ Messages d'erreur génériques pour éviter la fuite d'informations
- ⚠️ À améliorer : Logging détaillé des erreurs côté serveur

✅ Checklist de déploiement en production

Avant le déploiement

- [ ] Changer le secret JWT (`jwt.secret`)
- [ ] Utiliser des variables d'environnement pour les secrets
- [ ] Activer HTTPS
- [ ] Configurer CORS correctement
- [ ] Ajouter des limites de taux (rate limiting)
- [ ] Configurer les logs de production
- [ ] Mettre en place un système de monitoring
- [ ] Backup automatique de la base de données
- [ ] Tester tous les endpoints
- [ ] Vérifier les performances
- [ ] Documenter l'API (Swagger)

🎓 Conclusion

Ce système d'authentification JWT fournit une base solide pour l'application MSSP HealthCheck. Il implémente les meilleures pratiques de sécurité et suit l'architecture Spring Boot standard.

Points forts :

- Architecture modulaire et maintenable
- Sécurité robuste avec JWT
- Base de données relationnelle avec MySQL
- Validation des données
- Gestion des rôles et permissions

Améliorations futures :

- Refresh tokens
- Two-factor authentication (2FA)
- Audit logging complet
- Tests automatisés

- CI/CD pipeline

Document créé le 27 octobre 2025 Dernière mise à jour : 27 octobre 2025