



Complément Technique PoC V1.x

Table des matières :

1. Spring security / Jwt et token CSRF	02
1. Intégration au projet maven	02
2. Création d'une classe de configuration pour la sécurité	02
3. Implémentation des token JWT et d'un filtre d'authentification avec les classes	04
4. Adaptation des tests d'intégrations	07
5. Extension de l'utilisation des tokens au reste de l'application	09
6. Correction et validation de la collection Postman	18
2. Correction CI/CD	20
3. Multithreading	21
4. Log des communications API	22
5. SOLID – interfaçage	25
6. Modification du document de reporting	26
7. Event-Driven Architecture (EDA)	26

1. Spring security / Jwt et token CSRF

Cette section a pour objectif de présenter l'intégration d'une logique de sécurité reposant sur les tokens JWT (JSON Web Tokens) pour la gestion des sessions, ainsi que sur les tokens CSRF (Cross-Site Request Forgery) pour la protection contre les attaques CSRF. Bien que cette intégration soit avant tout démonstrative, elle illustre les principes fondamentaux de sécurisation des communications avec les endpoints de l'application. C'est une implémentation avant tout backend.

1) Intégration au projet maven

Installation dans le pom.xml :

```
<!-- Spring security -->
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.11.5</version>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-test</artifactId>
  <scope>test</scope>
</dependency>
```

1. Extrait de pom.xml

Activation de spring security et ajout des dépendance pour jwt.

2) Création d'une classe de configuration pour la sécurité :

```
package com.medHead.poc.config;

import com.medHead.poc.security.JwtAuthenticationFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```

import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.security.web.csrf.CookieCsrfTokenRepository;

@Configuration
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;

    /**
     * Constructeur pour injecter le filtre d'authentification JWT.
     * @param jwtAuthenticationFilter le filtre JWT personnalisé pour gérer
     l'authentification.
     */
    public SecurityConfig(JwtAuthenticationFilter jwtAuthenticationFilter) {
        this.jwtAuthenticationFilter = jwtAuthenticationFilter;
    }

    /**
     * Configure la chaîne de filtres de sécurité pour l'application.
     *
     * @param http l'objet HttpSecurity utilisé pour configurer les options de
     sécurité.
     * @return SecurityFilterChain configuré.
     * @throws Exception en cas de problème avec la configuration de sécurité.
     */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
        http
            .csrf(csrf -> csrf
                .csrfTokenRepository(CookieCsrfTokenRepository.withHttp
OnlyFalse())
                // Stockage dans un cookie accessible par JS
            )
            // Désactiver csrf en théorie
            .authorizeHttpRequests(auth -> auth
                .requestMatchers( "/swagger-ui/**", "/v3/api-docs/**",
"/", "/reserve", "/index.html", "/test/csrf", "/assets/**", "/auth/**",
"/resources/**", "/error", "/styles/*.css", "/styles/**", "/polyfills-*.js",
"/main-*.js", "/runtime-*.js", "/favicon.ico", "/scripts/**").permitAll() //
Autoriser l'accès à l'endpoint CSRF
                .requestMatchers("/api/**").hasRole("QUALIFIED_USER")
            )
            // Restrictions habituelles pour /api/**
            .anyRequest().authenticated()
            .addFilterBefore(jwtAuthenticationFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

```

2. SecurityConfig.java

Cette classe permet de distinguer les zones sécurisées et non sécurisées de l'application, tout en définissant les rôles nécessaires pour accéder aux pages protégées. Dans le cadre de la PoC, certains accès sont laissés libres, car la gestion des connexions

utilisateur n'est pas incluse dans le périmètre de la démonstration.
(src/main/java/com/medHead/poc/config/SecurityConfig)

3) Implémentation des token JWT et d'un filtre d'authentification avec les classes :

src/main/java/com.medHead.poc/security/JwtUtil
et src/main/java/com.medHead.poc/security/JwtAuthenticationFilter

```
package com.medHead.poc.security;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets;
import java.security.Key;
import java.util.Date;

/**
 * Classe utilitaire pour la gestion des tokens JWT (JSON Web Tokens).
 * Cette classe permet de générer, signer et valider des tokens JWT
 * en utilisant une clé secrète définie dans les propriétés de l'application.
 */
@Component
public class JwtUtil {
    /**
     * Clé secrète utilisée pour signer les tokens JWT.
     * La valeur est injectée depuis le fichier `application.properties`.
     * pour simplifier la logique dans la poc du fait de manque de contexte de
     login
     */
    @Value("${jwt.secret}")
    private String secret;

    private Key key;

    /**
     * Méthode appelée après l'injection des propriétés pour initialiser la clé
     * HMAC (Hash-based Message Authentication Code).
     */
    @PostConstruct
    public void init() {
        this.key = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));
    }

    /**
     * Génère un token JWT pour un utilisateur donné.
     *
     * @param username le nom d'utilisateur ou identifiant unique.
     * @param role le rôle de l'utilisateur (ex. "ROLE_QUALIFIED_USER").
     * @return un token JWT signé contenant le sujet, les rôles, et une date
     d'expiration.
     */
}
```

```

public String generateToken(String username, String role) {
    return Jwts.builder()
        .setSubject(username)
        .claim("role", role)
        .setExpiration(new Date(System.currentTimeMillis() + 86400000))
// 1 jour
        .signWith(key)
        .compact();
}

/**
 * Valide un token JWT et retourne les claims contenus dans le token.
 *
 * @param token le token JWT à valider.
 * @return un objet Claims contenant les informations décryptées du token.
 * @throws io.jsonwebtoken.JwtException si le token est invalide ou expiré.
 */
public Claims validateToken(String token) {
    return Jwts.parserBuilder()
        .setSigningKey(key)
        .build()
        .parseClaimsJws(token)
        .getBody();
}

/**
 * Méthode de test pour générer un token JWT en local.
 * Utile pour créer des tokens mockés dans le cadre de la PoC.
 * Décommenter pour utiliser au besoin
 * @param args les arguments de la ligne de commande (non utilisés).
 */
/*
public static void main(String[] args) {
    String secret =
"kpPb9R2v7NcGxJ5mQuZnR7q6k7Z3NdMv8XkM4A7C6aPp9W2q3RyR7NzV9QqR3A8x";
//clé de application.properties jwt.secret pour généré jwt.fixed-token
    Key key = Keys.hmacShaKeyFor(secret.getBytes(StandardCharsets.UTF_8));

    String token = Jwts.builder()
// pour la poc aucune expiration
        .setSubject("test-user")
        .claim("role", "ROLE_QUALIFIED_USER")
        .signWith(key, SignatureAlgorithm.HS512)
        .compact();

    System.out.println("Generated Token: " + token);
}*/
}

```

3. JwtUtil

JwtUtil permet de créer et de gérer les tokens JWT dans le projet de manière simple. Il offre notamment la possibilité de générer des tokens JWT signés et valides pour l'authentification et l'autorisation des utilisateurs. De plus, grâce à la méthode en commentaire dans la classe, il est possible de générer un token permanent, utile pour des tests ou dans le cadre d'une PoC.

```

package com.medHead.poc.security;

import io.jsonwebtoken.Claims;

```

```

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken
;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
import java.util.List;

/**
 * Filtre d'authentification basé sur JWT (JSON Web Token).
 * Ce filtre est exécuté pour chaque requête HTTP afin de valider le token JWT
 * présent dans l'en-tête d'autorisation et d'authentifier l'utilisateur.
 */
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;

    @Autowired
    public JwtAuthenticationFilter(JwtUtil jwtUtil) {
        this.jwtUtil = jwtUtil;
    }

    /**
     * Méthode principale du filtre pour traiter la requête et valider le token
     JWT.
     *
     * @param request la requête HTTP.
     * @param response la réponse HTTP.
     * @param chain la chaîne de filtres permettant de passer à la requête
     suivante.
     * @throws IOException en cas de problème d'entrée/sortie.
     * @throws ServletException en cas d'erreur au niveau du traitement de la
     requête.
     */
    @Override
    protected void doFilterInternal(HttpServletRequest request,
    HttpServletResponse response, FilterChain chain)
        throws IOException, ServletException {

        String header = request.getHeader("Authorization");
        System.out.println("Authorization Header: " + header);

        if (header == null || !header.startsWith("Bearer ")) {
            System.out.println("No valid Authorization header found.");
            chain.doFilter(request, response);
            return;
        }

        String token = header.substring(7);
        try {
            Claims claims = jwtUtil.validateToken(token);
            String username = claims.getSubject();

```

```

        String role = claims.get("role", String.class);

        System.out.println("Token Validated - Username: " + username + ",
Role: " + role);

        if (username != null && role != null) {
            UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(username, null,
List.of(new SimpleGrantedAuthority(role)));
SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    } catch (Exception e) {
        System.out.println("Token validation failed: " + e.getMessage());
        response.setStatus(HttpServletResponse.SC_FORBIDDEN);
        return;
    }

    chain.doFilter(request, response);
}
}

```

4. JwtAuthenticationFilter

Ce filtre vérifie le jeton JWT dans l'en-tête Authorization, s'assurant qu'il est au format "Bearer xxxxx". Il valide ensuite le contenu du jeton. En cas d'absence ou de jeton invalide, il gère les erreurs en continuant sans authentification ou en renvoyant une réponse HTTP 403 (Forbidden).

La logique de sécurité est implémentée. Il faut maintenant adapter les tests et le backend à son utilisation. Commençons par des tests mockés :

4) Adaptation des tests d'intégrations

Nous mettons en place, dans un premier temps, un endpoint dédié permettant de récupérer des tokens CSRF via le fichier `src/main/java/com/medHead/poc/controller/TestController` pour le chemin `/test/csrf`.

```

package com.medHead.poc.controller;

import jakarta.servlet.http.HttpServletRequest;
import org.springframework.http.ResponseEntity;
import org.springframework.security.web.csrf.CsrfToken;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * Contrôleur pour les tests afin de valider les fonctionnalités liées à la
 sécurité CSRF.
 * Cette classe fournit un endpoint permettant de récupérer le token CSRF
 * généré et géré par Spring Security.
 */
@RestController
@RequestMapping("/test")
public class TestController {

```

```

/**
 * Endpoint pour récupérer le token CSRF.
 * Ce token est géré par Spring Security et est automatiquement stocké dans
un cookie.
 * @param request l'objet {@link HttpServletRequest} permettant d'accéder
aux attributs de la requête.
 * @return une réponse contenant le token CSRF en tant que chaîne de
caractères.
 */
@GetMapping("/csrf")
public ResponseEntity<String> getCsrfToken(HttpServletRequest request) {
    CsrfToken csrfToken = (CsrfToken)
request.getAttribute(CsrfToken.class.getName());
    // Le token est maintenant disponible via csrfToken.getToken()
    // Inutile de remettre un cookie vous-même, Spring Security le fait
déjà.
    return ResponseEntity.ok(csrfToken.getToken());
}
}

```

5. TestController

Il faut à présent adapter les différents tests d'intégration à ce nouveau fonctionnement :

```

src/main/test/java/com.medHead.poc/testIntegration/AdviceControllerITest
src/main/test/java/com.medHead.poc/testIntegration/GPSControllerITest
src/main/test/java/com.medHead.poc/testIntegration/PatientControllerITest
src/main/test/java/com.medHead.poc/testIntegration/PopulateHopitalControllerITest
src/main/test/java/com.medHead.poc/testIntegration/ReserveControllerITest

```

L'ensemble des tests inclut une récupération de token dans la méthode @BeforeEach, si nécessaire, en particulier pour les requêtes autres que GET.

```

@BeforeEach
void setUp() throws Exception {
    patientService.clearPatients();
    // Réinitialise les patients
    mockServer = MockRestServiceServer.createServer(restTemplate);
    Locale.setDefault(Locale.US);

    // Étape 1 : Récupérer le cookie CSRF
    result = mockMvc.perform(get("/test/csrf"))
        .andExpect(status().isOk())
        .andReturn();

    // Récupérer le cookie CSRF de la réponse
    csrfCookie = result.getResponse().getCookie("XSRF-TOKEN");
    if (csrfCookie != null) {
        csrfToken = csrfCookie.getValue(); // Stocke le token CSRF
    }

    csrfCookie = result.getResponse().getCookie("XSRF-TOKEN");
    csrfToken = result.getResponse().getContentAsString();
}

```

6. Exemple @BeforeEach

Les requêtes sont donc complétées grâce à ces éléments

```
mockMvc.perform(post("/api/patients")
                .cookie(csrfCookie) // Réinjecte
le cookie CSRF
                .header("X-XSRF-TOKEN", csrfToken) // Transmet
le token dans le header
                .header("Authorization", "Bearer " + fixedToken)
```

<ul style="list-style-type: none"> ✓ PatientControllerTest (com.medHead.poc.test) ✓ testProcessPatient() ✓ testCreatePatient() ✓ testGetAllPatients() ✓ testDeletePatient() ✓ checkDefaultLocale() ✓ testCreatePatient_InvalidSpecialty() ✓ testProcessPatient_NoHospitalFound() ✓ testGetPatientById_NotFound() 	<ul style="list-style-type: none"> ✓ GPSControllerTest (com.medHead.poc.test) ✓ testValidCoordinates() ✓ testInvalidCoordinates() ✓ testGetTravelDelay() 	<ul style="list-style-type: none"> ✓ PatientControllerTest (com.medHead.poc.test) ✓ testProcessPatient() ✓ testCreatePatient() ✓ testGetAllPatients() ✓ testDeletePatient() ✓ checkDefaultLocale() ✓ testCreatePatient_InvalidSpecialty() ✓ testProcessPatient_NoHospitalFound() ✓ testGetPatientById_NotFound() 	<ul style="list-style-type: none"> ✓ PopulateHospitalControllerTest (com.medHead.poc.test) ✓ testClearCache() ✓ testGetHospitalsCache() ✓ testGetHospitals() 	<ul style="list-style-type: none"> ✓ ReserveControllerTest (com.medHead.poc.test) ✓ testReserveBed_Failure() ✓ testReserveBed_InvalidRequest() ✓ testGenerateCsrfToken() ✓ testReserveBed_Success()
---	--	---	--	--

```
[INFO] Results:
[INFO]
[INFO] Tests run: 90, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.698 s
[INFO] Finished at: 2024-12-11T19:25:50+01:00
[INFO] -----
```

7. Validation de l'implémentation

L'ensemble de l'intégrité des fonctions et rétablis pour les tests grâce à cette adaptation, mais elle n'est pas encore effective.

5) Extension de l'utilisation des tokens au reste de l'application

Afin de préserver la logique des tokens à travers les requêtes séquentielles, comme dans le processus principal de gestion d'un patient, les tokens spécifiques à cet usage seront injectés à l'aide de

src/main/java/com/medHead/poc/config/RestTemplateConfig.

```
package com.medHead.poc.config;

import com.medHead.poc.controller.TokenController;
import jakarta.annotation.PreDestroy;
import org.apache.http.impl.conn.PoolingHttpClientConnectionManager;
import org.slf4j.Logger;
```

```

import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.client.SimpleClientHttpRequestFactory;
import org.springframework.web.client.RestTemplate;

import javax.net.ssl.*;
import java.io.IOException;
import java.net.HttpURLConnection;
import java.security.cert.X509Certificate;
import java.util.concurrent.TimeUnit;

/**
 * Configuration globale pour le RestTemplate utilisé dans l'application.
 * Cette classe :
 * - Configure un pool de connexions pour optimiser les appels HTTP.
 * - Configure les timeouts des requêtes HTTP.
 * - Désactive temporairement la validation SSL (uniquement en développement ou PoC).
 */
@Configuration
public class RestTemplateConfig {

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

    @Value("${jwt.fixed-token}") // Injecte le token JWT depuis
application.properties
    private String fixedToken;

    /**
     * Contrôleur utilisé pour ajouter le token CSRF aux en-têtes des requêtes.
     */
    @Autowired
    private TokenController tokenController; // Utilisation du TokenController
pour le CSRF

    /**
     * Gestionnaire de pool de connexions pour optimiser les performances des
appels HTTP.
     */
    private PoolingHttpClientConnectionManager connectionManager;

    /**
     * Factory HTTP personnalisée pour gérer les timeouts et la configuration
des connexions.
     */
    private SimpleClientHttpRequestFactory requestFactory;

    /**
     * Crée et configure une instance de RestTemplate.
     *
     * @return Un RestTemplate configuré avec un pool de connexions et des
timeouts adaptés.
     */
    @Bean

```

```

public RestTemplate restTemplate() {
    try {
        // Désactivation de la validation SSL
        disableSslValidation();

        // Configuration du gestionnaire de pool de connexions
        configureConnectionManager();

        // Configuration de la factory HTTP avec gestion des timeouts
        SimpleClientHttpRequestFactory requestFactory = new
SimpleClientHttpRequestFactory() {
            @Override
            protected void prepareConnection(HttpURLConnection connection,
String httpMethod) throws IOException {
                super.prepareConnection(connection, httpMethod);
                connection.setRequestProperty("SO_REUSEADDR", "true");
                connection.setConnectTimeout(600);
// Timeout de connexion
                connection.setReadTimeout(400);
// Timeout de lecture
            }
        };

        // Création du RestTemplate avec la factory configurée
        RestTemplate restTemplate = new RestTemplate(requestFactory);

        // Ajout d'un intercepteur pour injecter le JWT et le CSRF
        restTemplate.getInterceptors().add((request, body, execution) -> {
            // Ajouter l'en-tête Authorization avec un JWT fixe
            request.getHeaders().set("Authorization", "Bearer " +
fixedToken);
            // Ajouter le token CSRF
            tokenController.addCsrfHeader(request.getHeaders());

            return execution.execute(request, body);
        });

        logger.info("RestTemplate configuré avec succès.");
        return restTemplate;
    } catch (Exception e) {
        logger.error("Erreur lors de la configuration du RestTemplate :
{}", e.getMessage(), e);
        throw new RuntimeException("Erreur lors de la configuration du
RestTemplate", e);
    }
}

/**
 * Configure le gestionnaire de pool de connexions HTTP.
 * Ce gestionnaire optimise les performances en réutilisant les connexions
existantes.
 */
private void configureConnectionManager() {
    connectionManager = new PoolingHttpClientConnectionManager(10,
TimeUnit.SECONDS);
    connectionManager.setMaxTotal(100);
// Nombre total maximum de connexions
    connectionManager.setDefaultMaxPerRoute(50);
// Nombre maximum de connexions par route
    logger.info("PoolingHttpClientConnectionManager configuré avec

```

```

maxTotal={} et maxPerRoute={},
        connectionManager.getMaxTotal(),
connectionManager.getDefaultMaxPerRoute());
    }

    /**
     * Désactive la validation SSL pour accepter tous les certificats.
     */
    private void disableSslValidation() throws Exception {
        TrustManager[] trustAllCerts = new TrustManager[]{
            new X509TrustManager() {
                public void checkClientTrusted(X509Certificate[] certs,
String authType) {}
                public void checkServerTrusted(X509Certificate[] certs,
String authType) {}
                public X509Certificate[] getAcceptedIssuers() { return
null; }
            }
        };

        SSLContext sc = SSLContext.getInstance("TLS");
        sc.init(null, trustAllCerts, new java.security.SecureRandom());
        HTTPSURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
        HTTPSURLConnection.setDefaultHostnameVerifier((hostname, session) ->
true);
        logger.info("Validation SSL désactivée.");
    }

    /**
     * Libère les ressources du gestionnaire de pool de connexions lors de la
destruction du bean.
     * Cette méthode est appelée automatiquement par le conteneur Spring à la
fin de l'application.
     */
    @PreDestroy
    public void cleanup() {
        if (connectionManager != null) {
            logger.info("Nettoyage du PoolingHttpClientConnectionManager.");
            connectionManager.close();
        }
    }
}

```

8. RestTemplateConfig.Java

Pour permettre son intégration à l'interface frontend, tout en conservant son usage actuel, un contrôleur dédié a été ajouté pour gérer la création de tokens CSRF destinés au frontend, via `src/main/java/com/medHead/poc/controller/TokenController` :

```

package com.medHead.poc.controller;

import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.http.HttpHeaders;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;

@Component

```

```

public class TokenController {

    private final RestTemplate restTemplate;

    private String csrfToken;

    // Injection via le constructeur pour éviter le cycle
    public TokenController(RestTemplateBuilder restTemplateBuilder) {
        this.restTemplate = restTemplateBuilder.build(); // Utilise un
RestTemplate standard
    }

    /**
     * Récupère un nouveau token CSRF avant chaque requête.
     */
    public String getCsrfToken() {
        try {
            // Appel au backend pour récupérer le token
            ResponseEntity<String> response =
restTemplate.getForEntity("http://localhost:8080/test/csrf", String.class);

            // Retourner la réponse brute du body comme token CSRF
            return response.getBody();
        } catch (Exception e) {
            // Loguer les erreurs pour mieux diagnostiquer les problèmes
            System.err.println("Erreur lors de la récupération du token CSRF :
" + e.getMessage());
            return null;
        }
    }

    /**
     * Ajoute le token CSRF à l'en-tête de la requête.
     *
     * @param headers Les en-têtes où le token doit être ajouté
     */
    public void addCsrfHeader(HttpHeaders headers) {
        String token = getCsrfToken();
        if (token != null) {
            headers.set("X-XSRF-TOKEN", token);
        } else {
            System.err.println("Impossible d'ajouter le token CSRF : token
null.");
        }
    }
}

```

8. TokenController.Java

Pour intégrer les tokens par injection dans la page unique de la PoC, qui ne comporte pas de logique de connexion, les éléments nécessaires à Angular ont été préparés via

/src/main/angular/src/app/service/ApiService.ts

```

import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';

```

```

@Injectable({
  providedIn: 'root',
})
export class ApiService {
  private readonly apiUrl = 'https://localhost:8443';

  constructor(private http: HttpClient) {}

  /**
   * Récupérer un token CSRF.
   */
  getCsrfToken(): Observable<string> {
    return this.http.get(`${this.apiUrl}/test/csrf`, { responseType: 'text' });
  }

  /**
   * Effectuer une requête POST avec JWT et CSRF.
   */
  postWithTokens(endpoint: string, payload: any, jwtToken: string, csrfToken:
string): Observable<any> {
    const headers = {
      'Content-Type': 'application/json',
      Authorization: `Bearer ${jwtToken}`,
      'X-XSRF-TOKEN': csrfToken, // Utilise le token brut ici
    };

    return this.http.post(`${this.apiUrl}${endpoint}`, payload, { headers });
  }
}

```

9. ApiService.ts

Ce service Angular gère l'interaction avec l'API en permettant de récupérer un token CSRF et d'effectuer des requêtes POST sécurisées avec des tokens JWT et CSRF.

Les tokens peuvent ensuite être ajoutés automatiquement aux headers des requêtes du service de réservation principal, comme défini dans :

/src/main/angular/src/app/reservation/reservation.component.ts

```

import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { ApiService } from '../service/ApiService';

// Importation directe de specialities.json
import specialitiesData from '../../assets/specialities.json';

@Component({
  selector: 'app-reservation',
  standalone: true,
  imports: [CommonModule, FormsModule],
  templateUrl: './reservation.component.html',
  styleUrls: ['./reservation.component.scss'],
})
export class ReservationComponent {
  qualite: string = 'Dr.';
}

```

```

nom: string = 'Frank Estein';
latitude: number = 48.8566;
longitude: number = 2.3522;
specialite: string = ''; // Spécialité
entrée par l'utilisateur
reservationResponse: any = null; // Réponse de l'API
specialities: string[] = []; // Liste des
spécialités disponibles
filteredSpecialities: string[] = [];
showUrgencyPopup: boolean = false; // Indique si le
popup d'urgence est affiché

jwtToken: string =
'eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXN0LXVzZXIiLCJyb2xlijoiek9MRV9RVUFMSUZJRURFV
VNFUiJ9.UD006gvM-0WabxCZ2UWEYrPkoxQrXu1pxQNnw3avnjhVCsrQ25KHT0zGkNa8uIFfxWqb-53
twgndJUbPMI8hCA'; // Pour stocker le JWT
csrfToken: string = ''; // Pour stocker le token CSRF

constructor(private apiService: ApiService) { // Injection correcte de
ApiService
    // Charger les spécialités au moment de l'initialisation
    this.loadSpecialities();
    // Charger les tokens au démarrage
    this.loadTokens();
}

// Charger les tokens au démarrage
loadTokens(): void {
    this.apiService.getCsrfToken().subscribe({
        next: (csrfToken: string) => {
            this.csrfToken = csrfToken;
            console.log('CSRF Token récupéré: ', this.csrfToken);
        },
        error: (err: any) => console.error('Erreur lors de la récupération du
token CSRF', err),
    });
}

// Charger les spécialités à partir du fichier JSON importé
loadSpecialities(): void {
    this.specialities = Object.keys(specialitiesData).filter((key) => key !==
'');
}

// Gérer la saisie de spécialité
onSpecialityInput(value: string): void {
    const inputValue = value.toLowerCase().trim();
    if (inputValue) {
        this.filteredSpecialities = this.specialities.filter((speciality) =>
speciality.toLowerCase().startsWith(inputValue)
        );
    } else {
        this.filteredSpecialities = []; // Vider les suggestions si le champ est
vide
    }
}

// Sélectionner une spécialité dans la liste des suggestions
selectSpeciality(speciality: string): void {
    this.specialite = speciality;
    this.filteredSpecialities = [];
}

```

```

}

// Générer un UUID
generateUUID(): string {
  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, (c) => {
    const r = (Math.random() * 16) | 0;
    const v = c === 'x' ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  });
}

// Méthode pour soumettre le formulaire
submitForm(): void {
  // Capitaliser la première lettre de la spécialité
  this.specialite = this.capitalizeFirstLetter(this.specialite);

  // Empêche la soumission si une réservation est en cours
  if (event) {
    event.preventDefault();
  }

  // Empêche la soumission si une réservation est en cours
  if (this.reservationResponse) {
    return;
  }

  if (!this.specialite || !this.specialities.includes(this.specialite)) {
    // Afficher le popup d'urgence pour spécialité inconnue
    this.showUrgencyPopup = true;
    return;
  }

  this.processReservation(this.specialite); // Réserver normalement si la
spécialité est valide
}

// Processus de réservation
processReservation(specialite: string): void {
  const reservationData = {
    id: this.generateUUID(),
    specialite: this.specialite,
    responsable: this.nom,
    qualite: this.qualite,
    latitude: this.latitude,
    longitude: this.longitude,
  };

  // Envoi des données avec fetch
  fetch('/api/patients/process', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
      'Authorization': `Bearer ${this.jwtToken}`, // Ajouter le JWT
      'X-XSRF-TOKEN': this.csrfToken, // Ajouter le
CSRF Token
    },
    body: JSON.stringify(reservationData),
    credentials: 'include'
  })
  .then((response) => response.json())

```



```

        .then((data) => {
            this.reservationResponse = data; // Affiche la
réponse dans la pop-up
        })
        .catch((error) => {
            console.error('Erreur lors de la réservation :', error);
            alert('Une erreur est survenue lors de la réservation.');
```

```
    });
```

```
    // Fermer la pop-up et réinitialiser la page
```

```
    closePopup() {
```

```
        console.log('Popup fermée');
```

```
        this.reservationResponse = null;
```

```
        // Cacher la
```

```
pop-up
```

```
        this.specialite = '';
```

```
        //
```

```
Réinitialiser le champ de spécialités
```

```
        this.filteredSpecialities = []; // Vider les suggestions
```

```
    }
```

```
    capitalizeFirstLetter(value: string): string {
```

```
        return value.charAt(0).toUpperCase() + value.slice(1).toLowerCase();
```

```
    }
```

```
    // Réserver en urgence
```

```
    reserveInUrgency(): void {
```

```
        this.specialite = "Médecine d'urgence"; // Remplit automatiquement avec
```

```
"Médecine d'urgence"
```

```
        this.showUrgencyPopup = false; // Ferme le popup
```

```
    // Soumettre les données de réservation
```

```
    const reservationData = {
```

```
        id: this.generateUUID(),
```

```
        specialite: this.specialite,
```

```
        responsable: this.nom,
```

```
        qualite: this.qualite,
```

```
        latitude: this.latitude,
```

```
        longitude: this.longitude,
```

```
    };
```

```
    // Envoi des données avec fetch
```

```
    fetch('/api/patients/process', {
```

```
        method: 'POST',
```

```
        headers: {
```

```
            'Content-Type': 'application/json',
```

```
            'Authorization': `Bearer ${this.jwtToken}`, // Ajouter le JWT
```

```
            'X-XSRF-TOKEN': this.csrfToken, // Ajouter le CSRF Token
```

```
        },
```

```
        body: JSON.stringify(reservationData),
```

```
        credentials: 'include'
```

```
    })
```

```
    .then((response) => response.json())
```

```
    .then((data) => {
```

```
        this.reservationResponse = data; // Affiche la réponse dans la pop-up
```

```
    })
```

```
    .catch((error) => {
```

```
        console.error('Erreur lors de la réservation :', error);
```

```
        alert('Une erreur est survenue lors de la réservation.');
```

```
    });
```

```
}
```

```

// Fermer le popup d'urgence
closeUrgencyPopup(): void {
  console.log('Popup fermée');
  this.reservationResponse = null; // Cacher la
pop-up
  this.specialite = ''; //
Réinitialiser le champ de spécialités
  this.showUrgencyPopup = false;
  this.filteredSpecialities = []; // Vider les suggestions
}
}

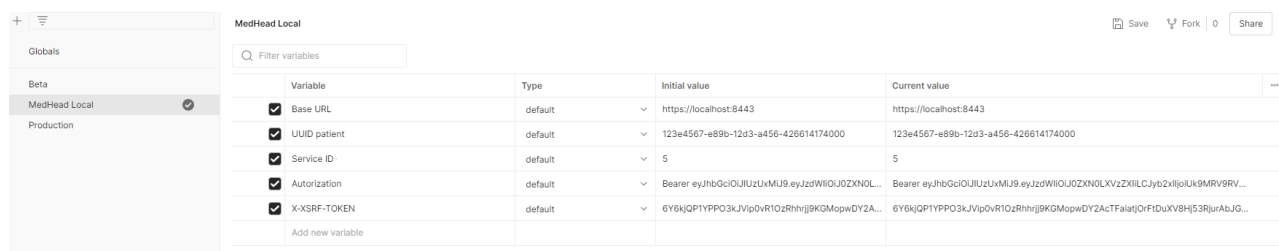
```

10. reservation.component.ts

Remarque : En raison de l'utilisation de deux origines distinctes (http://localhost:4200 pour le frontend Angular et https://localhost:8443 pour le backend Spring), il sera nécessaire de gérer les problèmes de CORS (Cross-Origin Resource Sharing) afin de finaliser l'intégration du frontend. Cela inclut une configuration appropriée des en-têtes CORS sur le backend pour autoriser les requêtes provenant de l'origine Angular.

6) Correction et validation de la collection Postman

Pour rendre Postman fonctionnel, il sera nécessaire d'ajouter deux variables d'environnement pour contenir les tokens JWT et CSRF.



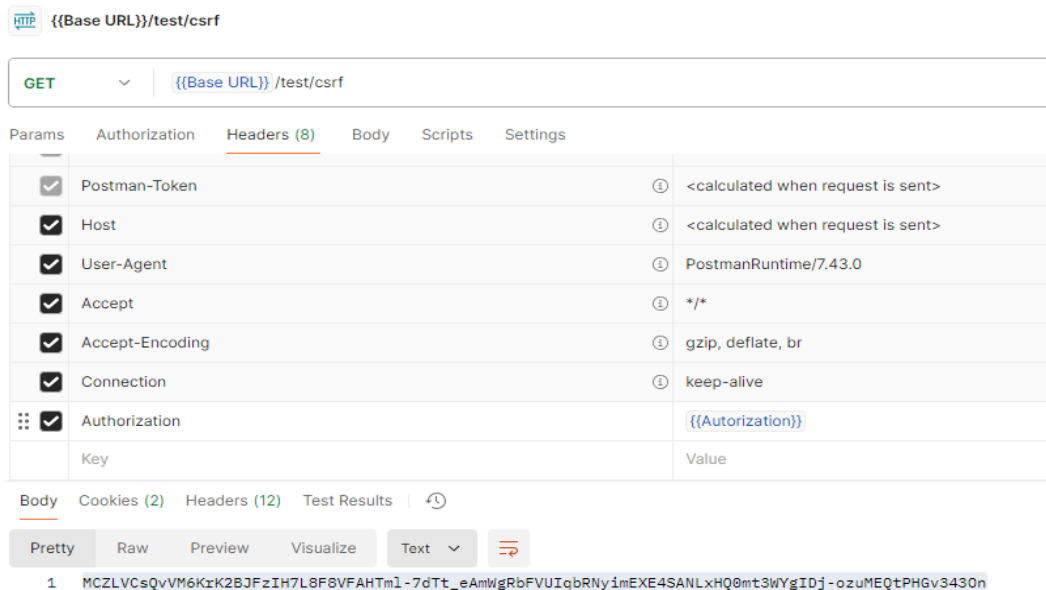
Variable	Type	Initial value	Current value
<input checked="" type="checkbox"/> Base URL	default	https://localhost:8443	https://localhost:8443
<input checked="" type="checkbox"/> UUID patient	default	123e4567-e89b-12d3-a456-426614174000	123e4567-e89b-12d3-a456-426614174000
<input checked="" type="checkbox"/> Service ID	default	5	5
<input checked="" type="checkbox"/> Authorization	default	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXNOLXVzZXliLCJyY2xiOiUk9MRV9RVUFMSUZJRURfVVNFUiJ9.UD006gvM-OWabxCZ2UWEYrPkoxQrXu1pxQNnw3avnjhVCsrQ25KHTOzGkNa8ulFfxWqb-53twgndJUbpMI8hCA	Bearer eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXNOLXVzZXliLCJyY2xiOiUk9MRV9RVUFMSUZJRURfVVNFUiJ9.UD006gvM-OWabxCZ2UWEYrPkoxQrXu1pxQNnw3avnjhVCsrQ25KHTOzGkNa8ulFfxWqb-53twgndJUbpMI8hCA
<input checked="" type="checkbox"/> X-XSRF-TOKEN	default	6Y6kJP1YPP03kJVip0vR10zRhrrjBKGmopwDY2A...	6Y6kJP1YPP03kJVip0vR10zRhrrjBKGmopwDY2A...
Add new variable			

11. Postman – variables

Le jeton JWT permanent utilisable pour la PoC est disponible dans le fichier application.properties :

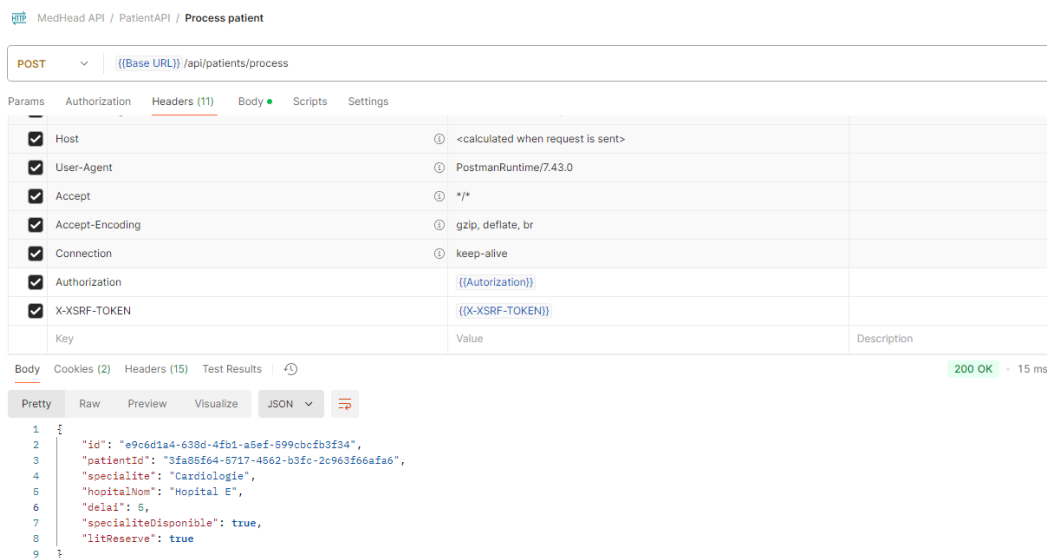
```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ0ZXNOLXVzZXliLCJyY2xiOiUk9MRV9RVUFMSUZJRURfVVNFUiJ9.UD006gvM-OWabxCZ2UWEYrPkoxQrXu1pxQNnw3avnjhVCsrQ25KHTOzGkNa8ulFfxWqb-53twgndJUbpMI8hCA
```

Il permet, grâce à une première requête GET vers `{{Base URL}}/test/csrf`, de récupérer un token pour remplir l'en-tête X-XSRF-TOKEN, en spécifiant également le jeton JWT dans l'en-tête Authorization.



11. Postman – Générateur de token CSRF

Ensuite, en utilisant les deux tokens, il permet de retrouver l'ensemble des fonctionnalités du backend :



12. Postman – POST -process

La collection est placée dans le dépôt `src/main/resources/MedHead API/token.postman_collection`.

2. Correction CI/CD

Dans le cadre de l'optimisation de notre processus d'intégration continue et de déploiement (CI/CD), la configuration du pipeline a été mise à jour pour rendre les tests et les builds plus flexibles et mieux séparés. Désormais, l'exécution des tests est distincte des étapes de build et peut être lancée manuellement via l'interface GitHub. Cela permet un contrôle plus précis des actions à effectuer tout en évitant l'exécution systématique des tests à chaque commit, ce qui améliore l'efficacité globale du pipeline.

Pour implémenter cette fonctionnalité, une branche a été créée avec la commande suivante : `git checkout -b OnDemandCICD`. Dans cette branche, le dossier `.github` contenant la configuration nécessaire pour GitHub Actions a été ajouté. Cette configuration a ensuite été poussée sur le dépôt. Une fois la branche mise à jour et poussée, elle a été fusionnée avec la branche `main`, rendant ainsi le pipeline CI/CD directement disponible dans GitHub Actions.

```
name: On demand CI/CD Pipeline

# Utilisation sur githubaction à la demande
on:
  workflow_dispatch:
    inputs:
      environnement:
        description: ''
        required: true
        default: 'dev'
        options:
          - main
          - test

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '20.11.1' # Node.js version compatible avec Angular

      - name: Install and Build Angular
        run: |
          cd src/main/angular
          npm install
          npm run build

  test:
    runs-on: ubuntu-latest
    needs: build
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
```

```

- name: Setup Java
  uses: actions/setup-java@v3
  with:
    distribution: 'temurin'
    java-version: '22'

- name: Run Backend Tests
  run: mvn clean test # Exécution des tests Java

- name: Generate JaCoCo Test Coverage Report
  run: mvn verify # Génération du rapport de couverture des tests

```

13. CiCd.yml

Disponibilité des tests à la demande :

14. Github action

3. Multithreading

Le multithreading consiste à mettre en place un pool de threads pour exécuter des tâches en concurrence, notamment pour soumettre les informations au service GPS et obtenir les temps de trajet entre l'emplacement du patient et les différents hôpitaux d'un district.

La situation initiale est un parcours séquentiel d'une collection, ici la liste des hôpitaux : (`src/main/java/com/medHead/poc/controller/PatientController`)

```

import java.util.concurrent.*;

// 3. Calculer les delais pour chaque hopital
for (Hopital hopital : hopitaux) {
    int delai = gpsService.getTravelDelay(
        patient.getLatitude(),
        patient.getLongitude(),

```

```

        hopital.getLatitude(),
        hopital.getLongitude()
    );
    hopital.setDelai(delai);
}

```

15. Traitement séquentiel des appels GPS

Maintenant, il est possible de déclarer un pool d'exécuteurs qui feront le même travail de manière simultanée et non synchronisée afin de maximiser le temps d'exécution, surtout ici si les requêtes à un service externe peuvent prendre un certain temps.

```

// 3. Calculer les delais pour chaque hopital
// Créer un thread pool avec un nombre fixe de threads
int nThreads = 10;

try (ExecutorService executorService = Executors.newFixedThreadPool(nThreads))
{
    List<Future<Void>> futures = new ArrayList<>();
    // Soumettre chaque tâche de calcul dans le pool
    for (Hopital hopital : hopitaux) {
        final Patient localPatient = patient;
        futures.add(executorService.submit(() -> {
            try {
                int delai = gpsService.getTravelDelay(
                    localPatient.getLatitude(),
                    localPatient.getLongitude(),
                    hopital.getLatitude(),
                    hopital.getLongitude()
                );
                hopital.setDelai(delai);
            } catch (Exception e) {
                e.printStackTrace(); // Afficher l'erreur dans les logs
            }
            return null; // Nous retournons null car il s'agit d'une tâche void
        }));
    }
    // Attendre que toutes les tâches soient terminées
    for (Future<Void> future : futures) {
        future.get(); // Ceci bloque jusqu'à ce que la tâche soit terminée
    }
    // Fermer le pool de threads
    executorService.shutdown();
}

```

16. Traitement concurrent des appels GPS

Il faut toutefois attendre la fin de la dernière réponse pour terminer véritablement le parcours. Enfin, le pool de threads peut être libéré après usage.

4. Log des communications API

Pour respecter les exigences du Règlement Général sur la Protection des Données (RGPD) tout en conservant des logs sur les échanges d'API, il est essentiel de limiter les données sensibles enregistrées, tout en disposant d'informations suffisantes pour

surveiller l'activité et diagnostiquer les problèmes de manière efficace (anonymisation des données personnelles, exclusion des données sensibles, respect de la confidentialité).

On configure un fichier *logback-spring.xml* afin de filtrer les logs et de créer un historique applicatif pour la PoC (enregistré dans */logs/ledHead.log*) ainsi qu'un historique de communication pour l'application définitive (enregistré dans */logs/http_logs.log*). (*src/main/resources/logback-spring.xml*)

```
<configuration>

  <!-- Appender pour les logs HTTP -->
  <appender name="HTTP_FILE" class="ch.qos.logback.core.FileAppender">
    <file>logs/http_logs.log</file>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- Appender pour les logs applicatifs -->
  <appender name="APP_FILE" class="ch.qos.logback.core.FileAppender">
    <file>logs/medHead.log</file>
    <encoder>
      <pattern>%d{yyyy-MM-dd HH:mm:ss} - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- Logger qui filtre les logs par marqueur HTTP -->
  <logger name="com.medHead.poc" level="DEBUG">
    <appender-ref ref="HTTP_FILE" />
  </logger>

  <!-- Logger qui filtre les logs applicatifs -->
  <logger name="com.medHead.app" level="DEBUG">
    <appender-ref ref="APP_FILE" />
  </logger>

  <root level="INFO">
    <appender-ref ref="CONSOLE" />
  </root>

</configuration>
```

17. logback-spring.xml

Les fichiers font l'objet d'implémentation pour pouvoir loggé les informations :

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.Marker;
import org.slf4j.MarkerFactory;

private static final Logger logger =
LoggerFactory.getLogger(GPSController.class);
private static final Marker HTTP_MARKER = MarkerFactory.getMarker("HTTP_FILE");
private static final Marker AP_MARKER =
MarkerFactory.getMarker("APPLICATION_FILE");

logger.info(HTTP_MARKER, "Log de communication");
logger.info(AP_MARKER, "Autre log");
```

18. Exemple d'import et configuration

Exemple de résultat du fichier http_logs.log pour des log RGPD friendly :

```
2024-12-12 06:26:44 - Nom de l'hôpital défini : Hopital I
2024-12-12 06:26:44 - Services disponibles définis : [1, 2, 3, 4, 5, 6, 9, 10, 11]
2024-12-12 06:26:44 - Latitude de l'hôpital définie : 48.859
2024-12-12 06:26:44 - Longitude de l'hôpital définie : 2.354
2024-12-12 06:26:44 - Nombre de lits invalide : 25
2024-12-12 06:26:44 - Délai de l'hôpital défini : 9999
2024-12-12 06:26:44 - ID de l'hôpital défini : null
2024-12-12 06:26:44 - Nom de l'hôpital défini : Hopital J
2024-12-12 06:26:44 - Services disponibles définis : [1, 5, 6, 7, 8, 10]
2024-12-12 06:26:44 - Latitude de l'hôpital définie : 48.8534
2024-12-12 06:26:44 - Longitude de l'hôpital définie : 2.3488
2024-12-12 06:26:44 - Nombre de lits invalide : 9
2024-12-12 06:26:44 - Délai de l'hôpital défini : 9999
2024-12-12 06:26:44 - Réponse reçue de l'API : [Hopital{id=null, nom='Hopital A', serviceIdsDisponibles=[1, 2, 4, 5, 6, 7, 8, 11], latitude=48.8566, longitude=2.3522, nombreLitDisponible=15, delai=9999}, Hopital{id=null, nom='Hopital B', serviceIdsDisponibles=[2, 3, 4, 5, 7, 9, 10, 11, 12], latitude=48.8648, longitude=2.3499, nombreLitDisponible=8, delai=9999}, Hopital{id=null, nom='Hopital C', serviceIdsDisponibles=[1, 2, 3, 5, 6, 7, 12], latitude=48.8584, longitude=2.2945, nombreLitDisponible=20, delai=9999}, Hopital{id=null, nom='Hopital D', serviceIdsDisponibles=[2, 3, 4, 5, 9, 10, 11], latitude=48.86, longitude=2.327, nombreLitDisponible=12, delai=9999}, Hopital{id=null, nom='Hopital E', serviceIdsDisponibles=[1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12], latitude=48.8675, longitude=2.33, nombreLitDisponible=5, delai=9999}, Hopital{id=null, nom='Hopital F', serviceIdsDisponibles=[1, 2, 5, 6, 7, 9, 10], latitude=48.8619, longitude=2.3364, nombreLitDisponible=18, delai=9999}, Hopital{id=null, nom='Hopital G', serviceIdsDisponibles=[1, 3, 4, 5, 8, 9, 11], latitude=48.8545, longitude=2.3478, nombreLitDisponible=10, delai=9999}, Hopital{id=null, nom='Hopital H', serviceIdsDisponibles=[1, 2, 3, 4, 5, 6, 7, 8, 12], latitude=48.855, longitude=2.3419, nombreLitDisponible=7, delai=9999}, Hopital{id=null, nom='Hopital I', serviceIdsDisponibles=[1, 2, 3, 4, 5, 6, 9, 10, 11], latitude=48.859, longitude=2.354, nombreLitDisponible=25, delai=9999}, Hopital{id=null, nom='Hopital J', serviceIdsDisponibles=[1, 5, 6, 7, 8, 10], latitude=48.8534, longitude=2.3488, nombreLitDisponible=9, delai=9999}]
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital. Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>, hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 5 minutes.
2024-12-12 06:26:44 - Délai de l'hôpital défini : 5
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital. Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>, hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 11 minutes.
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital. Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>, hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital. Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>, hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 16 minutes.
2024-12-12 06:26:44 - Délai de l'hôpital défini : 16
2024-12-12 06:26:44 - Délai de trajet calculé: 11 minutes.
2024-12-12 06:26:44 - Délai de l'hôpital défini : 11
2024-12-12 06:26:44 - Délai de l'hôpital défini : 11
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital. Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>, hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
```



```

2024-12-12 06:26:44 - Délai de trajet calculé: 11 minutes.
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital.
Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>,
hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital.
Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>,
hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 16 minutes.
2024-12-12 06:26:44 - Délai de l'hôpital défini : 16
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital.
Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>,
hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 13 minutes.
2024-12-12 06:26:44 - Délai de l'hôpital défini : 13
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital.
Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>,
hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de trajet calculé: 6 minutes.
2024-12-12 06:26:44 - Demande de délai de trajet entre patient et hôpital.
Paramètres: patientLatitude=<masqué>, patientLongitude=<masqué>,
hospitalLatitude=<masqué>, hospitalLongitude=<masqué>
2024-12-12 06:26:44 - Délai de l'hôpital défini : 6
2024-12-12 06:26:44 - Délai de l'hôpital défini : 11

```

19. *http_logs.log*

5. SOLID – interfaçage

Afin de respecter le principe de non-ségrégation des interfaces, des interfaces seront ajoutées à différentes classes du programme. Cette approche garantit une flexibilité accrue, permettant une évolution plus aisée de l'architecture sans compromettre l'intégrité du code existant.

Les classes concernées par cette refactorisation incluent principalement les services métiers, tels que **PatientService**, **GPSService**, **ReserveService**, et **PopulateHopitalService**, qui gèrent des logiques distinctes pouvant être adaptées ou remplacées par de nouvelles implémentations. De plus, des interfaces seront ajoutées aux classes de configuration, telles que **JwtUtil** et **JwtAuthenticationFilter**, pour faciliter la personnalisation de la gestion des tokens et de l'authentification. Cette démarche sera également étendue, dans une moindre mesure, aux contrôleurs **TestController** et **TokenController**.

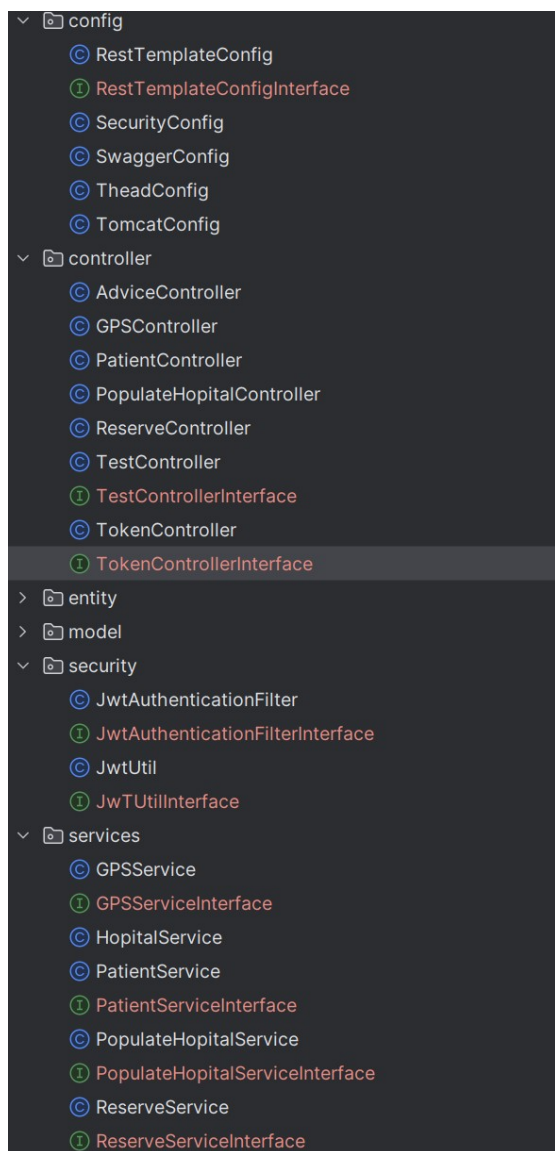
Cette approche assure une meilleure testabilité, une modularité accrue et une séparation claire des responsabilités.

```

public interface PatientServiceInterface {
    Patient initializePatient(Patient patient);
    List<Patient> getAllPatients();
    Optional<Patient> getPatientById(UUID id);
    Patient savePatient(Patient patient);
    boolean deletePatient(UUID id);
    Map<String, Integer> getSpecialityDictionary();
    void clearPatients();
}

```

20. Exemple pour *PatientServiceInterface*



Dans la classe de l'interface :

```
@Service
public class PatientService implements
PatientServiceInterface {
```

Toutes les autres classes qui utilisent PatientService doivent désormais passer par PatientServiceInterface.

```
@Service 8 usages 1 MickaelDP *
public class PatientService implements PatientServiceInterface {
```

L'objectif est donc d'avoir zéro usage direct.

```
@Service 1 usage 1 MickaelDP *
public class PatientService implements PatientServiceInterface {
    private static final Logger logger = LoggerFactory.getLogger(PatientService.class);
    private static final Marker HTTP_MARKER = MarkerFactory.getMarker("HTTP_FILE");
```

Au final, PatientServiceInterface est implémentée par PatientService, et le seul usage direct de PatientService est fait par cette même classe.

Et la même procédure est appliquée à chaque interface.

21. Implémenter les interfaces

6. Modification du document de reporting

Des corrections et des précisions ont été apportées à différentes parties du document de reporting, prenant en compte les éventuels manques et changements, tels que ceux relatifs au CI/CD.

7. Event-Driven Architecture (EDA)

Dans le cadre de la gestion d'événements, tels que des alertes signalant qu'aucun lit n'est disponible pour un service à l'échelle d'un district ou même au niveau national, l'utilisation d'une architecture pilotée par événements (EDA) peut s'avérer particulièrement efficace. Pour mettre en œuvre ce type de solution, des outils comme

Apache Kafka peuvent être utilisés afin de faciliter la gestion, la transmission et le traitement des événements en temps réel.