Plateforme de Gestion d'Événements

Phase 3 : Sécurité - Authentification et Autorisation

Sécurisons notre application →

Objectifs de la Phase 3

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)
- 🔑 **Créer** un système JWT pour l'authentification stateless

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)
- 🔑 **Créer** un système JWT pour l'authentification stateless
- 🌐 **Intégrer** OAuth2 (Google/GitHub)

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)
- 🔑 **Créer** un système JWT pour l'authentification stateless
- 🌐 **Intégrer** OAuth2 (Google/GitHub)
- 🛡️ **Sécuriser** tous les endpoints de l'API

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)
- 🔑 **Créer** un système JWT pour l'authentification stateless
- 🌐 **Intégrer** OAuth2 (Google/GitHub)
- 🛡️ **Sécuriser** tous les endpoints de l'API
- 🚪 **Contrôler** l'accès aux ressources selon les rôles

Objectifs de la Phase 3

- 🔐 **Implémenter** Spring Security pour l'authentification
- 👥 **Gérer** les rôles et permissions (RBAC)
- 🔑 **Créer** un système JWT pour l'authentification stateless
- 🌐 **Intégrer** OAuth2 (Google/GitHub)
- 🛡️ **Sécuriser** tous les endpoints de l'API
- 🚪 **Contrôler** l'accès aux ressources selon les rôles

À la fin de cette phase, votre API sera complètement sécurisée avec plusieurs modes d'authentification !

Spring Security : Les Concepts Clés

Samuel MICHAUX

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

Samuel MICHAUX

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

2. Authorization (Que pouvez-vous faire ?)

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

2. Authorization (Que pouvez-vous faire ?)

- Vérifier les permissions
- Rôles, authorities, méthodes

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

2. Authorization (Que pouvez-vous faire ?)

- Vérifier les permissions
- Rôles, authorities, méthodes

3. Protection

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

2. Authorization (Que pouvez-vous faire ?)

- Vérifier les permissions
- Rôles, authorities, méthodes

3. Protection

- CSRF, XSS, Session Fixation
- CORS, Headers de sécurité

Samuel MICHAUX

Spring Security : Les Concepts Clés

1. Authentication (Qui êtes-vous ?)

- Vérifier l'identité de l'utilisateur
- Login/Password, Token, OAuth2, etc.

2. Authorization (Que pouvez-vous faire ?)

- Vérifier les permissions
- Rôles, authorities, méthodes

3. Protection

- CSRF, XSS, Session Fixation
- CORS, Headers de sécurité

```
// La chaîne de sécurité Spring
Client → Filter → Authentication → Authorization → Resource
```

Samuel MICHAUX

layout: two-cols zoom: 0.8

Architecture de Sécurité

Samuel MICHAUX

Architecture de Sécurité

Composants Spring Security

Samuel MICHAUX

Architecture de Sécurité

Composants Spring Security

1. SecurityFilterChain
    - Configure la sécurité HTTP
2. UserDetailsService
    - Charge les utilisateurs
3. PasswordEncoder
    - Hash des mots de passe
4. AuthenticationManager
    - Gère l'authentification

pom.xml

Ajout

```xml
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>

<!-- JWT -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-jackson</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
```

```xml
<!-- OAuth2 Client (Optionnel) -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>

<!-- Spring Security Test -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
```

Configuration de Base Spring Security

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

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

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .anyRequest().authenticated()
            );

        return http.build();
    }
}
```

Configuration de Base Spring Security

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

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

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .anyRequest().authenticated()
            );

        return http.build();
    }
}
```

Configuration de Base Spring Security

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

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

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .anyRequest().authenticated()
            );

        return http.build();
    }
}
```

Configuration de Base Spring Security

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

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

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .anyRequest().authenticated()
            );

        return http.build();
    }
}
```

Configuration de Base Spring Security

```java
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;

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

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

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .anyRequest().authenticated()
            );

        return http.build();
    }
}
```

UserDetailsService Personnalisé

```java
@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
            .orElseThrow(() -> new UsernameNotFoundException(
                "Utilisateur non trouvé : " + email
            ));

        return CustomUserPrincipal.create(user);
    }

    // Méthode utilisée par JWT
    @Transactional
    public UserDetails loadUserById(Long id) {
        User user = userRepository.findById(id)
            .orElseThrow(() -> new ResourceNotFoundException("User", id));

        return CustomUserPrincipal.create(user);
    }
}
```

```java
// CustomUserPrincipal.java
@Data
@AllArgsConstructor
public class CustomUserPrincipal implements UserDetails {
    private Long id;
    private String email;
    private String password;
    private Collection<? extends GrantedAuthority> authorities

    public static CustomUserPrincipal create(User user) {
        List<GrantedAuthority> authorities = Arrays.asList(
            new SimpleGrantedAuthority("ROLE_" + user.getRole(
        );

        return new CustomUserPrincipal(
            user.getId(),
            user.getEmail(),
            user.getPassword(),
            authorities
        );
    }

    // Implémenter les méthodes UserDetails...
}
```

JWT (JSON Web Tokens)

🔑

Authentification stateless pour les APIs REST

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

1. Header : Type et algorithme

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

1. Header : Type et algorithme

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

1. Header : Type et algorithme

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

2. Payload : Les claims (données)

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

1. **Header** : Type et algorithme

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

2. **Payload** : Les claims (données)

```
{
  "sub": "1",
  "email": "user@example.com",
  "roles": ["ROLE_USER"],
  "iat": 1704067200,
  "exp": 1704931200
}
```

Comprendre JWT

Structure d'un JWT

```
eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiIxIiwiaWF0IjoxNzA0MDY3MjAwLCJleHAiOjE3MDQ5MzEyMDB9.signature
```

Les 3 parties (séparées par des `.`)

1. **Header** : Type et algorithme

```
{
  "alg": "HS512",
  "typ": "JWT"
}
```

2. **Payload** : Les claims (données)

```
{
  "sub": "1",
  "email": "user@example.com",
  "roles": ["ROLE_USER"],
  "iat": 1704067200,
  "exp": 1704931200
}
```

3. **Signature** : Vérification de l'intégrité

JwtTokenProvider

```java
@Component
@Slf4j
public class JwtTokenProvider {

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

    @Value("${app.jwt.expiration}")
    private int jwtExpirationInMs;

    public String generateToken(Authentication authentication) {
        CustomUserPrincipal userPrincipal =
            (CustomUserPrincipal) authentication.getPrincipal();

        Date expiryDate = new Date(
            System.currentTimeMillis() + jwtExpirationInMs
        );

        return Jwts.builder()
            .setSubject(Long.toString(userPrincipal.getId()))
            .claim("email", userPrincipal.getEmail())
            .claim("roles", userPrincipal.getAuthorities())
            .setIssuedAt(new Date())
            .setExpiration(expiryDate)
            .signWith(SignatureAlgorithm.HS512, jwtSecret)
            .compact();
    }
...
```

```java
...
    public Long getUserIdFromJWT(String token) {
        Claims claims = Jwts.parser()
            .setSigningKey(jwtSecret)
            .parseClaimsJws(token)
            .getBody();

        return Long.parseLong(claims.getSubject());
    }

    public boolean validateToken(String authToken) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(authToken);
            return true;
        } catch (SignatureException ex) {
            log.error("Signature JWT invalide");
        } catch (MalformedJwtException ex) {
            log.error("Token JWT invalide");
        } catch (ExpiredJwtException ex) {
            log.error("Token JWT expiré");
        } catch (UnsupportedJwtException ex) {
            log.error("Token JWT non supporté");
        } catch (IllegalArgumentException ex) {
            log.error("JWT claims string est vide");
        }
        return false;
    }
}
```

JWT Authentication Filter

```java
@Component
@RequiredArgsConstructor
@Slf4j
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtTokenProvider tokenProvider;
    private final CustomUserDetailsService userDetailsService;

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

        String token = getTokenFromRequest(request);

        if (StringUtils.hasText(token) && tokenProvider.validateToken(token)) {
            Long userId = tokenProvider.getUserIdFromJWT(token);
            UserDetails userDetails = userDetailsService.loadUserById(userId);

            UsernamePasswordAuthenticationToken authentication =
                new UsernamePasswordAuthenticationToken(
                    userDetails, null, userDetails.getAuthorities()
                );
            authentication.setDetails(
                new WebAuthenticationDetailsSource().buildDetails(request)
            );

            SecurityContextHolder.getContext().setAuthentication(authentication);
        }

        filterChain.doFilter(request, response);
    }

    private String getTokenFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
```

Controller d'Authentification

```java
@RestController @RequestMapping("/api/auth")
@RequiredArgsConstructor @Slf4j
public class AuthController {

    private final AuthenticationManager authenticationManager;
    private final JwtTokenProvider tokenProvider;
    private final UserService userService;

    @PostMapping("/login")
    public ResponseEntity<JwtAuthenticationResponse> authenticateUser(
            @Valid @RequestBody LoginRequest loginRequest) {

        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                loginRequest.getEmail(),
                loginRequest.getPassword()
            )
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = tokenProvider.generateToken(authentication);

        CustomUserPrincipal userPrincipal =
            (CustomUserPrincipal) authentication.getPrincipal();

        return ResponseEntity.ok(new JwtAuthenticationResponse(
            jwt,
            userPrincipal.getId(),
            userPrincipal.getEmail(),
            userPrincipal.getAuthorities()
        ));
    }
```

```java
...
    @PostMapping("/register")
    public ResponseEntity<ApiResponse> registerUser(
            @Valid @RequestBody SignUpRequest signUpRequest) {

        if (userService.existsByEmail(signUpRequest.getEmail())) {
            return ResponseEntity.badRequest()
                .body(new ApiResponse(false, "Email déjà utilisé!"));
        }

        User user = new User();
        user.setEmail(signUpRequest.getEmail());
        user.setPassword(passwordEncoder.encode(signUpRequest.getPassword()
        user.setFirstName(signUpRequest.getFirstName());
        user.setLastName(signUpRequest.getLastName());
        user.setRole(Role.PARTICIPANT);

        User result = userService.save(user);

        return ResponseEntity.ok(
            new ApiResponse(true, "Utilisateur enregistré avec succès")
        );
    }
}
```

Sécurisation par Rôles (RBAC)

Role-Based Access Control

Annotations de Sécurité

Au niveau méthode

```java
@RestController
@RequestMapping("/api/events")
@RequiredArgsConstructor
public class EventController {

    // Accessible à tous les authentifiés
    @GetMapping
    @PreAuthorize("isAuthenticated()")
    public List<EventDTO> getAllEvents() {
        return eventService.findAll();
    }

    // Seulement les organisateurs et admins
    @PostMapping
    @PreAuthorize("hasAnyRole('ORGANIZER', 'ADMIN')")
    public EventDTO createEvent(@Valid @RequestBody CreateEventDTO dto) {
        return eventService.create(dto);
    }

    // Seulement le propriétaire ou admin
    @PutMapping("/{id}")
    @PreAuthorize("@eventSecurity.isOwnerOrAdmin(#id, authentication)")
    public EventDTO updateEvent(
            @PathVariable Long id,
            @Valid @RequestBody UpdateEventDTO dto) {
        return eventService.update(id, dto);
    }

    // Seulement les admins
    @DeleteMapping("/{id}")
    @PreAuthorize("hasRole('ADMIN')")
    public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
        eventService.delete(id);
        return ResponseEntity.noContent().build();
    }
}
```

Security Expressions Personnalisées

```java
@Component("eventSecurity")
@RequiredArgsConstructor
public class EventSecurity {

    private final EventRepository eventRepository;

    public boolean isOwnerOrAdmin(Long eventId, Authentication authentication) {
        if (authentication == null || !authentication.isAuthenticated()) {
            return false;
        }

        CustomUserPrincipal principal =
            (CustomUserPrincipal) authentication.getPrincipal();

        // Admin a tous les droits
        if (principal.getAuthorities().stream()
                .anyMatch(a -> a.getAuthority().equals("ROLE_ADMIN"))) {
            return true;
        }

        // Vérifier si l'utilisateur est l'organisateur
        return eventRepository.findById(eventId)
            .map(event -> event.getOrganizer().getId().equals(principal.getId()))
            .orElse(false);
    }

    public boolean canRegisterToEvent(Long eventId, Authentication authentication) {
        // Logique pour vérifier si l'utilisateur peut s'inscrire
        // - Event existe et a de la place
        // - User n'est pas déjà inscrit
        // - User n'est pas l'organisateur
        return true; // Simplifiée pour l'exemple
    }
}
```

Configuration Complète avec JWT

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final CustomUserDetailsService userDetailsService;
    private final JwtAuthenticationEntryPoint unauthorizedHandler;
    private final JwtAuthenticationFilter jwtAuthenticationFilter;

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

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

```
...
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .csrf(csrf -> csrf.disable())
            .exceptionHandling(ex -> ex
                .authenticationEntryPoint(unauthorizedHandler)
            )
            .sessionManagement(session ->
                session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/auth/**").permitAll()
                .requestMatchers("/api/public/**").permitAll()
                .requestMatchers("/h2-console/**").permitAll()
                .requestMatchers(HttpMethod.GET, "/api/events/**").permitAll()
                .anyRequest().authenticated()
            );

        // Ajouter notre filtre JWT
        http.addFilterBefore(
            jwtAuthenticationFilter,
            UsernamePasswordAuthenticationFilter.class
        );

        return http.build();
    }
...
```

```
...
    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
        configuration.setAllowedHeaders(Arrays.asList("*"));
        configuration.setAllowCredentials(true);

        UrlBasedCorsConfigurationSource source =
            new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);
        return source;
    }
}
```

OAuth2 avec Spring Security

Connexion via Google, GitHub, etc.

Configuration OAuth2

1. Dépendances Maven

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

2. Configuration application.yml

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            scope:
              - email
              - profile

          github:
            client-id: ${GITHUB_CLIENT_ID}
            client-secret: ${GITHUB_CLIENT_SECRET}
            scope:
              - user:email
              - read:user

        provider:
          google:
            issuer-uri: https://accounts.google.com
```

Configuration OAuth2

1. Dépendances Maven

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

2. Configuration application.yml

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            scope:
              - email
              - profile

          github:
            client-id: ${GITHUB_CLIENT_ID}
            client-secret: ${GITHUB_CLIENT_SECRET}
            scope:
              - user:email
              - read:user

        provider:
          google:
            issuer-uri: https://accounts.google.com
```

Configuration OAuth2

1. Dépendances Maven

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

2. Configuration application.yml

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            scope:
              - email
              - profile

          github:
            client-id: ${GITHUB_CLIENT_ID}
            client-secret: ${GITHUB_CLIENT_SECRET}
            scope:
              - user:email
              - read:user

        provider:
          google:
            issuer-uri: https://accounts.google.com
```

OAuth2 Success Handler

```java
@Component @RequiredArgsConstructor @Slf4j
public class OAuth2AuthenticationSuccessHandler
        extends SimpleUrlAuthenticationSuccessHandler {

    private final JwtTokenProvider tokenProvider;
    private final UserService userService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
                                        HttpServletResponse response,
                                        Authentication authentication)
            throws IOException {

        OAuth2User oAuth2User = (OAuth2User) authentication.getPrincipal();

        // Extraire les infos de l'utilisateur OAuth
        String email = oAuth2User.getAttribute("email");
        String name = oAuth2User.getAttribute("name");

        // Créer ou mettre à jour l'utilisateur
        User user = processOAuthUser(email, name);

        // Générer JWT
        String token = tokenProvider.generateTokenFromUserId(user.getId());

        // Rediriger avec le token
        String targetUrl = UriComponentsBuilder
            .fromUriString("/oauth2/redirect")
            .queryParam("token", token)
            .build().toUriString();
        getRedirectStrategy().sendRedirect(request, response, targetUrl);
    }
```

```java
...
    private User processOAuthUser(String email, String name) {
        return userService.findByEmail(email)
            .orElseGet(() -> {
                // Créer un nouvel utilisateur OAuth
                User newUser = new User();
                newUser.setEmail(email);
                newUser.setFirstName(name.split(" ")[0]);
                newUser.setLastName(name.split(" ").length > 1 ?
                    name.split(" ")[1] : "");
                newUser.setRole(Role.PARTICIPANT);
                newUser.setPassword(UUID.randomUUID().toString()); // Random
                return userService.save(newUser);
            });
    }
}
```

Configuration OAuth2 dans SecurityConfig

```java
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http
        // ... configuration existante ...

        .oauth2Login(oauth2 -> oauth2
            .authorizationEndpoint(auth -> auth
                .baseUri("/oauth2/authorize")
            )
            .redirectionEndpoint(redirect -> redirect
                .baseUri("/oauth2/callback/*")
            )
            .userInfoEndpoint(userInfo -> userInfo
                .userService(customOAuth2UserService)
            )
            .successHandler(oAuth2AuthenticationSuccessHandler)
            .failureHandler(oAuth2AuthenticationFailureHandler)
        );

    return http.build();
}

// Page de login avec OAuth2
@GetMapping("/login")
public String loginPage(Model model) {
    model.addAttribute("oauth2Urls", Map.of(
        "google", "/oauth2/authorization/google",
        "github", "/oauth2/authorization/github"
    ));
    return "login";
}
```

Test des Endpoints Sécurisés

Sans authentification

```
curl http://localhost:8080/api/events
# 401 Unauthorized
```

Login et récupération du token

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"user@test.com","password":"password123"}'

# Response:
# {
#   "accessToken": "eyJhbGciOiJIUzUxMiJ9...",
#   "tokenType": "Bearer",
#   "userId": 1,
#   "email": "user@test.com",
#   "roles": ["ROLE_PARTICIPANT"]
# }
```

Utilisation du token

```
curl http://localhost:8080/api/events \
  -H "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9..."

# 200 OK + Liste des événements
```

Gestion des Erreurs de Sécurité

```java
@Component
@Slf4j
public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request,
                         HttpServletResponse response,
                         AuthenticationException authException)
            throws IOException {

        log.error("Accès non autorisé - {}", authException.getMessage());

        response.setContentType(MediaType.APPLICATION_JSON_VALUE);
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);

        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.UNAUTHORIZED,
            "Accès non autorisé. Veuillez vous authentifier.",
            request.getServletPath()
        );

        ObjectMapper mapper = new ObjectMapper();
        mapper.writeValue(response.getOutputStream(), errorResponse);
    }
}
```

```java
@Component
public class JwtAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request,
                       HttpServletResponse response,
                       AccessDeniedException accessDeniedException)
            throws IOException {

        response.setContentType(MediaType.APPLICATION_JSON_VALUE);
        response.setStatus(HttpServletResponse.SC_FORBIDDEN);

        ErrorResponse errorResponse = new ErrorResponse(
            HttpStatus.FORBIDDEN,
            "Accès refusé. Permissions insuffisantes.",
            request.getServletPath()
        );

        ObjectMapper mapper = new ObjectMapper();
        mapper.writeValue(response.getOutputStream(), errorResponse);
    }
}
```

Exercices Pratiques 🎯

Exercice 1 : Ajouter un Refresh Token

Implémenter le renouvellement de token

1. Créer une entité `RefreshToken`
2. Générer un refresh token lors du login
3. Endpoint `/api/auth/refresh` pour renouveler
4. Invalider les anciens tokens

Exercice 1 : Ajouter un Refresh Token

Implémenter le renouvellement de token

1. Créer une entité `RefreshToken`
2. Générer un refresh token lors du login
3. Endpoint `/api/auth/refresh` pour renouveler
4. Invalider les anciens tokens

Structure suggérée :

```
@Entity
public class RefreshToken {
    @Id
    private String token;
    private User user;
    private LocalDateTime expiryDate;
}
```

Exercice 2 : Audit de Sécurité

Tracer les actions sensibles

1. Créer un `@Audited` annotation
2. Intercepter les actions importantes
3. Logger : qui, quoi, quand
4. Tableau de bord des activités

```
1   @PostMapping
2   @Audited(action = "CREATE_EVENT")
3   public EventDTO createEvent(...) {
4       // L'action est automatiquement loggée
5   }
```

Exercice 3 : Two-Factor Authentication

Ajouter une couche de sécurité

1. Générer un code à 6 chiffres
2. L'envoyer par email
3. Vérifier avant de générer le JWT
4. Expiration après 5 minutes

Samuel MICHAUX

Récapitulatif Phase 3 ✅

- Spring Security configuré
- Authentification JWT fonctionnelle
- Gestion des rôles (RBAC)
- OAuth2 intégré (optionnel)
- Endpoints sécurisés
- Gestion des erreurs d'authentification

Best Practices Sécurité 

Samuel MICHAUX

Best Practices Sécurité 

1. **Mots de passe**
    - Toujours hasher avec BCrypt
    - Politique de mots de passe forts
    - Ne jamais logger les passwords

Best Practices Sécurité 

1. **Mots de passe**
   - Toujours hasher avec BCrypt
   - Politique de mots de passe forts
   - Ne jamais logger les passwords
2. **Tokens JWT**
   - Durée de vie courte (1-24h)
   - Secret fort et externalisé
   - Refresh tokens pour le renouvellement

Best Practices Sécurité 

1. **Mots de passe**
   - Toujours hasher avec BCrypt
   - Politique de mots de passe forts
   - Ne jamais logger les passwords
2. **Tokens JWT**
   - Durée de vie courte (1-24h)
   - Secret fort et externalisé
   - Refresh tokens pour le renouvellement
3. **HTTPS obligatoire**
   - Jamais de JWT sur HTTP
   - Certificats SSL/TLS valides

Best Practices Sécurité 

1. **Mots de passe**
   - Toujours hasher avec BCrypt
   - Politique de mots de passe forts
   - Ne jamais logger les passwords
2. **Tokens JWT**
   - Durée de vie courte (1-24h)
   - Secret fort et externalisé
   - Refresh tokens pour le renouvellement
3. **HTTPS obligatoire**
   - Jamais de JWT sur HTTP
   - Certificats SSL/TLS valides
4. **Validation côté serveur**
   - Ne jamais faire confiance au client
   - Vérifier les permissions à chaque requête

Prochaine étape : Phase 4

Spring Batch pour les traitements

Prochaine étape : Phase 4

Spring Batch pour les traitements

- 📊 **Export** de données en CSV

Prochaine étape : Phase 4

Spring Batch pour les traitements

- 📊 **Export** de données en CSV
- 📧 **Envoi** d'emails en masse

Prochaine étape : Phase 4

Spring Batch pour les traitements

- 📊 **Export** de données en CSV
- 📧 **Envoi** d'emails en masse
- ⏰ **Planification** de jobs

Prochaine étape : Phase 4

Spring Batch pour les traitements

- 📊 **Export** de données en CSV
- 📧 **Envoi** d'emails en masse
- ⏰ **Planification** de jobs
- 🔄 **Traitements** asynchrones

Prochaine étape : Phase 4

Spring Batch pour les traitements

- 📊 **Export** de données en CSV
- 📧 **Envoi** d'emails en masse
- ⏰ **Planification** de jobs
- 🔄 **Traitements** asynchrones

> 💪 Votre API est maintenant sécurisée et prête pour la production !

layout: center class: text-center

Questions ?

La sécurité est cruciale, prenez le temps de bien comprendre !

Code source Phase 3