

Plateforme de Gestion d'Événements

Phase 1 : Initialisation et Modélisation



Commençons notre voyage Spring Boot →

Objectifs de la Phase 1




Objectifs de la Phase 1

-  **Créer** un projet Spring Boot from scratch





Objectifs de la Phase 1

-  **Créer** un projet Spring Boot from scratch
-  **Configurer** la base de données (H2 pour commencer)





Objectifs de la Phase 1

-  **Créer** un projet Spring Boot from scratch
-  **Configurer** la base de données (H2 pour commencer)
-  **Définir** les entités JPA principales

Objectifs de la Phase 1

-  **Créer** un projet Spring Boot from scratch
-  **Configurer** la base de données (H2 pour commencer)
-  **Définir** les entités JPA principales
-  **Implémenter** un premier contrôleur REST

Objectifs de la Phase 1

-  **Créer** un projet Spring Boot from scratch
-  **Configurer** la base de données (H2 pour commencer)
-  **Définir** les entités JPA principales
-  **Implémenter** un premier contrôleur REST

Note : À la fin de cette phase, vous aurez une application Spring Boot fonctionnelle avec une API REST basique !

Spring Boot, c'est quoi ?

Spring Boot, c'est quoi ?

- Framework Java **opiniated**

Spring Boot, c'est quoi ?

- Framework Java **opiniated**
- Configuration **automatique**

Spring Boot, c'est quoi ?

- Framework Java **opiniated**
- Configuration **automatique**
- Serveur **embarqué**

Spring Boot, c'est quoi ?

- Framework Java **opiniated**
- Configuration **automatique**
- Serveur **embarqué**
- Production **ready**

Spring Boot, c'est quoi ?

- Framework Java **opiniated**
- Configuration **automatique**
- Serveur **embarqué**
- Production **ready**

```
1  @SpringBootApplication
2  public class EventApp {
3      public static void main(String[] args) {
4          SpringApplication.run(EventApp.class, args);
5      }
6  }
```

Création du projet Spring Boot

Option 1 : Spring Initializr (Recommandé).

1. Aller sur start.spring.io
2. Configurer :
 - **Project**: Maven
 - **Language**: Java
 - **Spring Boot**: 3.5.x
 - **Java**: 17 ou 21
3. Dépendances à ajouter :
 - Spring Web
 - Spring Data JPA
 - H2 Database
 - Spring Boot DevTools
 - Lombok (optionnel)

Création du projet Spring Boot

Option 1 : Spring Initializr (Recommandé).

1. Aller sur start.spring.io
2. Configurer :
 - **Project**: Maven
 - **Language**: Java
 - **Spring Boot**: 3.5.x
 - **Java**: 17 ou 21
3. Dépendances à ajouter :
 - Spring Web
 - Spring Data JPA
 - H2 Database
 - Spring Boot DevTools
 - Lombok (optionnel)

💡 ****Tip****: Téléchargez et décompressez le ZIP généré dans votre workspace

Structure du projet

```
1  event-management/  
2  |— src/  
3  |   |— main/  
4  |   |   |— java/  
5  |   |   |   |— com/formation/events/  
6  |   |   |   |   |— EventManagementApplication.java  
7  |   |   |   |   |— entities/  
8  |   |   |   |   |— repositories/  
9  |   |   |   |   |— controllers/  
10 |   |   |   |   |— services/  
11 |   |   |— resources/  
12 |   |   |   |— application.properties  
13 |— pom.xml  
14 |— README.md
```


Structure du projet

```
1  event-management/  
2  |— src/  
3  |   |— main/  
4  |   |   |— java/  
5  |   |   |   |— com/formation/events/  
6  |   |   |   |   |— EventManagementApplication.java  
7  |   |   |   |   |— entities/  
8  |   |   |   |   |— repositories/  
9  |   |   |   |   |— controllers/  
10 |   |   |   |   |— services/  
11 |   |   |— resources/  
12 |   |   |   |— application.properties  
13 |— pom.xml  
14 |— README.md
```

Note : Convention importante : Respectez cette structure pour que Spring Boot trouve automatiquement vos composants !

Configuration de la base de données

application.properties

```
1  # Configuration H2 (Base de données en mémoire)
2  spring.datasource.url=jdbc:h2:mem:eventdb
3  spring.datasource.driverClassName=org.h2.Driver
4  spring.datasource.username=sa
5  spring.datasource.password=
6
7  # Console H2 (pour déboguer)
8  spring.h2.console.enabled=true
9  spring.h2.console.path=/h2-console
10
11 # JPA / Hibernate
12 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
13 spring.jpa.hibernate.ddl-auto=create-drop
14 spring.jpa.show-sql=true
15 spring.jpa.properties.hibernate.format_sql=true
```

Configuration de la base de données

application.properties

```
1  # Configuration H2 (Base de données en mémoire)
2  spring.datasource.url=jdbc:h2:mem:eventdb
3  spring.datasource.driverClassName=org.h2.Driver
4  spring.datasource.username=sa
5  spring.datasource.password=
6
7  # Console H2 (pour débbugger)
8  spring.h2.console.enabled=true
9  spring.h2.console.path=/h2-console
10
11 # JPA / Hibernate
12 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
13 spring.jpa.hibernate.ddl-auto=create-drop
14 spring.jpa.show-sql=true
15 spring.jpa.properties.hibernate.format_sql=true
```

🔍 Console H2 accessible sur : <http://localhost:8080/h2-console>

Les Entités JPA



Les fondations de notre modèle de données

Entité Utilisateur

```
1  @Entity @Table(name = "users")
2  @Data @NoArgsConstructor @AllArgsConstructor
3  public class UserEntity {
4      @Id
5      @GeneratedValue(strategy = GenerationType.IDENTITY)
6      private Long id;
7
8      @Column(nullable = false, unique = true)
9      private String email;
10
11     @Column(nullable = false)
12     private String password;
13
14     @Column(nullable = false)
15     private String firstName;
16
17     @Column(nullable = false)
18     private String lastName;
19
20     @Enumerated(EnumType.STRING)
21     private RoleEnum role = RoleEnum.PARTICIPANT;
22
23     @OneToMany(mappedBy = "organizer")
24     private List<EventEntity> organizedEvents = new ArrayList<>();
25
26     @ManyToMany(mappedBy = "participants")
27     private List<EventEntity> participatingEvents = new ArrayList<>();
28 }
```

Entité Événement

```
@Entity @Table(name = "events")
@Data @NoArgsConstructor @AllArgsConstructor
public class EventEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false)
    private String title;

    @Column(length = 1000)
    private String description;

    @Column(nullable = false)
    private LocalDateTime startDate;

    private LocalDateTime endDate;

    @Column(nullable = false)
    private String location;

    private Integer maxParticipants;

    @ManyToOne
    @JoinColumn(name = "organizer_id", nullable = false)
    private UserEntity organizer;

    @ManyToMany
    @JoinTable(
        name = "event_participants",
        joinColumns = @JoinColumn(name = "event_id"),
        inverseJoinColumns = @JoinColumn(name = "user_id")
    )
    private List<UserEntity> participants = new ArrayList<>();
}
```

Enum Role

```
1  public enum RoleEnum {  
2      PARTICIPANT,  
3      ORGANIZER,  
4      ADMIN  
5  }
```

Enum Role

```
1 public enum RoleEnum {  
2     PARTICIPANT,  
3     ORGANIZER,  
4     ADMIN  
5 }
```

Relations JPA résumées

UserEntity → EventEntity

- @OneToMany : Un user peut organiser plusieurs events
- @ManyToMany : Un user peut participer à plusieurs events

EventEntity → UserEntity

- @ManyToOne : Un event a un seul organisateur
- @ManyToMany : Un event peut avoir plusieurs participants

Premier Contrôleur REST



Exposons nos données au monde !

EventController - Structure de base

```
1  @RestController
2  @RequestMapping("/api/events")
3  @RequiredArgsConstructor
4  public class EventController {
5
6      private final EventService eventService;
7
8      @GetMapping
9      public ResponseEntity<List<EventEntity>> getAllEvents() {
10         List<EventEntity> events = eventService.findAll();
11         return ResponseEntity.ok(events);
12     }
13
14     @GetMapping("/{id}")
15     public ResponseEntity<EventEntity> getEventById(@PathVariable Long id) {
16         return eventService.findById(id)
17             .map(ResponseEntity::ok)
18             .orElse(ResponseEntity.notFound().build());
19     }
20 }
```

EventController - Structure de base

```
1  @RestController
2  @RequestMapping("/api/events")
3  @RequiredArgsConstructor
4  public class EventController {
5
6      private final EventService eventService;
7
8      @GetMapping
9      public ResponseEntity<List<EventEntity>> getAllEvents() {
10         List<EventEntity> events = eventService.findAll();
11         return ResponseEntity.ok(events);
12     }
13
14     @GetMapping("/{id}")
15     public ResponseEntity<EventEntity> getEventById(@PathVariable Long id) {
16         return eventService.findById(id)
17             .map(ResponseEntity::ok)
18             .orElse(ResponseEntity.notFound().build());
19     }
20 }
```

💡 @RequiredArgsConstructor de Lombok génère automatiquement le constructeur pour l'injection de dépendances !

Sécurisation par Rôles (RBAC).



Role-Based Access Control

Annotations de Sécurité

Au niveau méthode

```
1  @RestController
2  @RequestMapping("/api/events")
3  @RequiredArgsConstructor
4  public class EventController {
5
6      // Accessible à tous les authentifiés
7      @GetMapping
8      @PreAuthorize("isAuthenticated()")
9      public List<EventDTO> getAllEvents() {
10         return eventService.findAll();
11     }
12
13     // Seulement les organisateurs et admins
14     @PostMapping
15     @PreAuthorize("hasAnyRole('ORGANIZER', 'ADMIN')")
16     public EventDTO createEvent(@Valid @RequestBody CreateEventDTO dto) {
17         return eventService.create(dto);
18     }
19
20     // Seulement le propriétaire ou admin
21     @PutMapping("/{id}")
22     @PreAuthorize("@eventSecurity.isOwnerOrAdmin(#id, authentication)")
23     public EventDTO updateEvent(
24         @PathVariable Long id,
25         @Valid @RequestBody UpdateEventDTO dto) {
26         return eventService.update(id, dto);
27     }
28
29     // Seulement les admins
30     @DeleteMapping("/{id}")
31     @PreAuthorize("hasRole('ADMIN')")
32     public ResponseEntity<Void> deleteEvent(@PathVariable Long id) {
33         eventService.delete(id);
34         return ResponseEntity.noContent().build();
35     }
36 }
```

Security Expressions Personnalisées

```
1  @Component("eventSecurity")
2  @RequiredArgsConstructor
3  public class EventSecurity {
4
5      private final EventRepository eventRepository;
6
7      public boolean isOwnerOrAdmin(Long eventId, Authentication authentication) {
8          if (authentication == null || !authentication.isAuthenticated()) {
9              return false;
10         }
11
12         CustomUserPrincipal principal =
13             (CustomUserPrincipal) authentication.getPrincipal();
14
15         // Admin a tous les droits
16         if (principal.getAuthorities().stream()
17             .anyMatch(a -> a.getAuthority().equals("ROLE_ADMIN"))) {
18             return true;
19         }
20
21         // Vérifier si l'utilisateur est l'organisateur
22         return eventRepository.findById(eventId)
23             .map(event -> event.getOrganizer().getId().equals(principal.getId()))
24             .orElse(false);
25     }
26
27     public boolean canRegisterToEvent(Long eventId, Authentication authentication) {
28         // Logique pour vérifier si l'utilisateur peut s'inscrire
29         // - Event existe et a de la place
30         // - User n'est pas déjà inscrit
31         // - User n'est pas l'organisateur
32         return true; // Simplifiée pour l'exemple
33     }
34 }
```

Configuration Complète avec JWT

```
1  @Configuration
2  @EnableWebSecurity
3  @EnableMethodSecurity(prePostEnabled = true)
4  @RequiredArgsConstructor
5  public class SecurityConfig {
6
7      private final CustomUserDetailsService userDetailsService;
8      private final JwtAuthenticationEntryPoint unauthorizedHandler;
9      private final JwtAuthenticationFilter jwtAuthenticationFilter;
10
11      @Bean
12      public PasswordEncoder passwordEncoder() {
13          return new BCryptPasswordEncoder();
14      }
15
16      @Bean
17      public AuthenticationManager authenticationManager(
18          AuthenticationConfiguration authConfig) throws Exception {
19          return authConfig.getAuthenticationManager();
20      }
21  ...
```

```
22  ...
23      @Bean
24      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
25          http
26              .cors(cors -> cors.configurationSource(corsConfigurationSource()))
27              .csrf(csrf -> csrf.disable())
28              .exceptionHandling(ex -> ex
29                  .authenticationEntryPoint(unauthorizedHandler)
30              )
31              .sessionManagement(session ->
32                  session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)
33              )
34              .authorizeHttpRequests(auth -> auth
35                  .requestMatchers("/api/auth/**").permitAll()
36                  .requestMatchers("/api/public/**").permitAll()
37                  .requestMatchers("/h2-console/**").permitAll()
38                  .requestMatchers(HttpMethod.GET, "/api/events/**").permitAll()
39                  .anyRequest().authenticated()
40              );
41
42          // Ajouter notre filtre JWT
43          http.addFilterBefore(
44              jwtAuthenticationFilter,
45              UsernamePasswordAuthenticationFilter.class
46          );
47
48          return http.build();
49      }
50  ...
```



```
51     ...
52     @Bean
53     public CorsConfigurationSource corsConfigurationSource() {
54         CorsConfiguration configuration = new CorsConfiguration();
55         configuration.setAllowedOrigins(Arrays.asList("http://localhost:3000"));
56         configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE"));
57         configuration.setAllowedHeaders(Arrays.asList("*"));
58         configuration.setAllowCredentials(true);
59
60         UrlBasedCorsConfigurationSource source =
61             new UrlBasedCorsConfigurationSource();
62         source.registerCorsConfiguration("/**", configuration);
63         return source;
64     }
65 }
```

OAuth2 avec Spring Security.



Connexion via Google, GitHub, etc.

Configuration OAuth2

1. Dépendances Maven

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-oauth2-client</artifactId>
4 </dependency>
```

2. Configuration application.yml

```
1  spring:
2    security:
3      oauth2:
4        client:
5          registration:
6            google:
7              client-id: ${GOOGLE_CLIENT_ID}
8              client-secret: ${GOOGLE_CLIENT_SECRET}
9              scope:
10               - email
11               - profile
12
13          github:
14            client-id: ${GITHUB_CLIENT_ID}
15            client-secret: ${GITHUB_CLIENT_SECRET}
16            scope:
17              - user:email
18              - read:user
19
20        provider:
21          google:
22            issuer-uri: https://accounts.google.com
```

Configuration OAuth2

1. Dépendances Maven

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-oauth2-client</artifactId>
4 </dependency>
```

2. Configuration application.yml

```
1  spring:
2    security:
3      oauth2:
4        client:
5          registration:
6            google:
7              client-id: ${GOOGLE_CLIENT_ID}
8              client-secret: ${GOOGLE_CLIENT_SECRET}
9              scope:
10               - email
11               - profile
12
13          github:
14            client-id: ${GITHUB_CLIENT_ID}
15            client-secret: ${GITHUB_CLIENT_SECRET}
16            scope:
17              - user:email
18              - read:user
19
20        provider:
21          google:
22            issuer-uri: https://accounts.google.com
```

Configuration OAuth2

1. Dépendances Maven

```
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-oauth2-client</artifactId>
4 </dependency>
```

2. Configuration application.yml

```
1  spring:
2    security:
3      oauth2:
4        client:
5          registration:
6            google:
7              client-id: ${GOOGLE_CLIENT_ID}
8              client-secret: ${GOOGLE_CLIENT_SECRET}
9              scope:
10               - email
11               - profile
12
13          github:
14            client-id: ${GITHUB_CLIENT_ID}
15            client-secret: ${GITHUB_CLIENT_SECRET}
16            scope:
17              - user:email
18              - read:user
19
20        provider:
21          google:
22            issuer-uri: https://accounts.google.com
```

OAuth2 Success Handler

```
1  @Component @RequiredArgsConstructor @Slf4j
2  public class OAuth2AuthenticationSuccessHandler
3      extends SimpleUrlAuthenticationSuccessHandler {
4
5      private final JwtTokenProvider tokenProvider;
6      private final UserService userService;
7
8      @Override
9      public void onAuthenticationSuccess(HttpServletRequest request,
10                                         HttpServletResponse response,
11                                         Authentication authentication)
12          throws IOException {
13
14          OAuth2User oAuth2User = (OAuth2User) authentication.getPrincipal();
15
16          // Extraire les infos de l'utilisateur OAuth
17          String email = oAuth2User.getAttribute("email");
18          String name = oAuth2User.getAttribute("name");
19
20          // Créer ou mettre à jour l'utilisateur
21          User user = processOAuthUser(email, name);
22
23          // Générer JWT
24          String token = tokenProvider.generateTokenFromUserId(user.getId());
25
26          // Rediriger avec le token
27          String targetUrl = UriComponentsBuilder
28              .fromUriString("/oauth2/redirect")
29              .queryParam("token", token)
30              .build().toUriString();
31          getRedirectStrategy().sendRedirect(request, response, targetUrl);
32      }
```

```
33      ...
34      private User processOAuthUser(String email, String name) {
35          return userService.findByEmail(email)
36              .orElseGet(() -> {
37              // Créer un nouvel utilisateur OAuth
38              User newUser = new User();
39              newUser.setEmail(email);
40              newUser.setFirstName(name.split(" ")[0]);
41              newUser.setLastName(name.split(" ").length > 1 ?
42                  name.split(" ")[1] : "");
43              newUser.setRole(Role.PARTICIPANT);
44              newUser.setPassword(UUID.randomUUID().toString()); // Random
45              return userService.save(newUser);
46          });
47      }
48  }
```

Configuration OAuth2 dans SecurityConfig

```
1  @Bean
2  public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
3      http
4          // ... configuration existante ...
5
6          .oauth2Login(oauth2 -> oauth2
7              .authorizationEndpoint(auth -> auth
8                  .baseUrl("/oauth2/authorize")
9              )
10             .redirectionEndpoint(redirect -> redirect
11                 .baseUrl("/oauth2/callback/*")
12             )
13             .userInfoEndpoint(userInfo -> userInfo
14                 .userService(customOAuth2UserService)
15             )
16             .successHandler(oAuth2AuthenticationSuccessHandler)
17             .failureHandler(oAuth2AuthenticationFailureHandler)
18         );
19
20     return http.build();
21 }
22
23 // Page de login avec OAuth2
24 @GetMapping("/login")
25 public String loginPage(Model model) {
26     model.addAttribute("oauth2Urls", Map.of(
27         "google", "/oauth2/authorization/google",
28         "github", "/oauth2/authorization/github"
29     ));
30     return "login";
31 }
```

Test des Endpoints Sécurisés

Sans authentification

```
1 curl http://localhost:8080/api/events
2 # 401 Unauthorized
```

Login et récupération du token

```
1 curl -X POST http://localhost:8080/api/auth/login \
2 -H "Content-Type: application/json" \
3 -d '{"email":"user@test.com","password":"password123"}'
4
5 # Response:
6 # {
7 #   "accessToken": "eyJhbGciOiJIUzUxMiJ9...",
8 #   "tokenType": "Bearer",
9 #   "userId": 1,
10 #   "email": "user@test.com",
11 #   "roles": ["ROLE_PARTICIPANT"]
12 # }
```

Utilisation du token

```
1 curl http://localhost:8080/api/events \
2 -H "Authorization: Bearer eyJhbGciOiJIUzUxMiJ9..."
3
4 # 200 OK + Liste des événements
```


Gestion des Erreurs de Sécurité

```
1  @Component
2  @Slf4j
3  public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {
4
5      @Override
6      public void commence(HttpServletRequest request,
7                          HttpServletResponse response,
8                          AuthenticationException authException)
9          throws IOException {
10
11          log.error("Accès non autorisé - {}", authException.getMessage());
12
13          response.setContentType(MediaType.APPLICATION_JSON_VALUE);
14          response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
15
16          ErrorResponse errorResponse = new ErrorResponse(
17              HttpStatus.UNAUTHORIZED,
18              "Accès non autorisé. Veuillez vous authentifier.",
19              request.getServletPath()
20          );
21
22          ObjectMapper mapper = new ObjectMapper();
23          mapper.writeValue(response.getOutputStream(), errorResponse);
24      }
25  }
```

```
1  @Component
2  public class JwtAccessDeniedHandler implements AccessDeniedHandler {
3
4      @Override
5      public void handle(HttpServletRequest request,
6                      HttpServletResponse response,
7                      AccessDeniedException accessDeniedException)
8          throws IOException {
9
10         response.setContentType(MediaType.APPLICATION_JSON_VALUE);
11         response.setStatus(HttpServletResponse.SC_FORBIDDEN);
12
13         ErrorResponse errorResponse = new ErrorResponse(
14             HttpStatus.FORBIDDEN,
15             "Accès refusé. Permissions insuffisantes.",
16             request.getServletPath()
17         );
18
19         ObjectMapper mapper = new ObjectMapper();
20         mapper.writeValue(response.getOutputStream(), errorResponse);
21     }
22 }
```

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 :

```
1  @Entity
2  public class RefreshToken {
3      @Id
4      private String token;
5      private User user;
6      private LocalDateTime expiryDate;
7  }
```

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