

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 **opinionated**

Spring Boot, c'est quoi ?

- Framework Java **opinited**
- 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 **opinied**
- 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 décomptez 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 debugger)
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 debugger)
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 !

Repository Pattern

```
1  @Repository
2  public interface EventRepository extends JpaRepository<EventEntity, Long> {
3
4      // Méthodes de requête dérivées
5      List<EventEntity> findByOrganizerOrderByStartDateDesc(UserEntity organizer);
6
7      List<EventEntity> findByStartDateAfter(LocalDateTime date);
8
9      @Query("SELECT e FROM EventEntity e WHERE e.maxParticipants > SIZE(e.participants)")
10     List<EventEntity> findEventsWithAvailableSpots();
11 }
```

Repository Pattern

```
1  @Repository
2  public interface EventRepository extends JpaRepository<EventEntity, Long> {
3
4      // Méthodes de requête dérivées
5      List<EventEntity> findByOrganizerOrderByStartDateDesc(UserEntity organizer);
6
7      List<EventEntity> findByStartDateAfter(LocalDateTime date);
8
9      @Query("SELECT e FROM EventEntity e WHERE e.maxParticipants > SIZE(e.participants)")
10     List<EventEntity> findEventsWithAvailableSpots();
11 }
```

Interface UserRepository

```
1  @Repository
2  public interface UserRepository extends JpaRepository<UserEntity, Long> {
3      Optional<UserEntity> findByEmail(String email);
4      boolean existsByEmail(String email);
5  }
```

Service Layer

```
1  @Service
2  @RequiredArgsConstructor
3  public class EventService {
4
5      private final EventRepository eventRepository;
6
7      public List<EventEntity> findAll() {
8          return eventRepository.findAll();
9      }
10
11     public Optional<EventEntity> findById(Long id) {
12         return eventRepository.findById(id);
13     }
14
15     public EventEntity save(EventEntity event) {
16         return eventRepository.save(event);
17     }
18
19     public List<EventEntity> findUpcomingEvents() {
20         return eventRepository.findByStartDateAfter(LocalDateTime.now());
21     }
22
23     public List<EventEntity> findEventsWithAvailableSpots() {
24         return eventRepository.findEventsWithAvailableSpots();
25     }
26
27     public void deleteById(Long id) {
28         eventRepository.deleteById(id);
29     }
30 }
```

Initialisation des données

```
1  @Component
2  @RequiredArgsConstructor
3  public class DataInitializer implements CommandLineRunner {
4
5      private final UserRepository userRepository;
6      private final EventRepository eventRepository;
7
8      @Override
9      public void run(String... args) {
10          // Créer des utilisateurs de test
11          UserEntity organizer = new UserEntity();
12          organizer.setEmail("organizer@test.com");
13          organizer.setPassword("password123");
14          organizer.setFirstName("Jean");
15          organizer.setLastName("Organisateur");
16          organizer.setRole(RoleEnum.ORGANIZER);
17          userRepository.save(organizer);
18
19          // Créer des événements de test
20          EventEntity event = new EventEntity();
21          event.setTitle("Conférence Spring Boot");
22          event.setDescription("Introduction à Spring Boot pour débutants");
23          event.setStartDate(LocalDateTime.now().plusDays(7));
24          event.setLocation("Paris, Salle 101");
25          event.setMaxParticipants(50);
26          event.setOrganizer(organizer);
27          eventRepository.save(event);
28
29          System.out.println("Données de test initialisées !");
30      }
31  }
```

Test de l'API

Endpoints disponibles

GET /api/events

```
1 curl http://localhost:8080/api/events
```

Réponse :

```
1 [
2   {
3     "id": 1,
4     "title": "Conférence Spring Boot",
5     "startDate": "2024-01-15T10:00:00",
6     "location": "Paris, Salle 101"
7   }
8 ]
```

GET /api/events/1

```
1 curl http://localhost:8080/api/events/1
```

Réponse :

```
1 {
2   "id": 1,
3   "title": "Conférence Spring Boot",
4   "description": "Introduction...",
5   "maxParticipants": 50
6 }
```

Exercices Pratiques

Exercice 1 : Ajouter une entité

Créez une entité *Registration* (*Inscription*).

Propriétés :

- `id` (Long)
- `user` (relation ManyToOne)
- `event` (relation ManyToOne)
- `registrationDate` (LocalDateTime)
- `status` (enum : PENDING, CONFIRMED, CANCELLED)

Exercice 1 : Ajouter une entité

Créez une entité *Registration* (*Inscription*).

Propriétés :

- `id` (Long)
- `user` (relation ManyToOne)
- `event` (relation ManyToOne)
- `registrationDate` (LocalDateTime)
- `status` (enum : PENDING, CONFIRMED, CANCELLED)

Indices

- Pensez à la clé composite (user + event)
- Ajoutez les annotations JPA appropriées
- Créez le repository correspondant

Exercice 2 : Enrichir l'API

Ajoutez ces endpoints au EventController

1. `POST /api/events` - Créer un événement
2. `PUT /api/events/{id}` - Modifier un événement
3. `DELETE /api/events/{id}` - Supprimer un événement
4. `GET /api/events/upcoming` - Liste des événements futurs

Exercice 2 : Enrichir l'API

Ajoutez ces endpoints au EventController

1. `POST /api/events` - Créer un événement
2. `PUT /api/events/{id}` - Modifier un événement
3. `DELETE /api/events/{id}` - Supprimer un événement
4. `GET /api/events/upcoming` - Liste des événements futurs

Bonus

- Gérez les cas d'erreur (404, 400)
- Utilisez des DTOs pour les requêtes/réponses

Récapitulatif Phase 1 ✓

- Projet Spring Boot initialisé
- Base de données H2 configurée
- Entités JPA créées avec relations
- API REST basique fonctionnelle
- Données de test disponibles

Prochaine étape : Phase 2

Au programme

Prochaine étape : Phase 2

Au programme

-  **Formulaires de création/modification**

Prochaine étape : Phase 2

Au programme

-  **Formulaires** de création/modification
-  **Validation** des données avec Spring Validator

Prochaine étape : Phase 2

Au programme

-  **Formulaires** de création/modification
-  **Validation** des données avec Spring Validator
-  **Gestion des erreurs** utilisateur

Prochaine étape : Phase 2

Au programme

-  **Formulaires** de création/modification
-  **Validation** des données avec Spring Validator
-  **Gestion des erreurs** utilisateur
-  **Requêtes personnalisées** avec Spring Data

Questions ?



N'hésitez pas à expérimenter avec le code !

Voir le code sur GitHub