

Spring Batch

Traitement de données par lots avec Spring Boot

Formation EPITA - Gestion d'événements OpenAgenda

Commencer la formation →

Qu'est-ce que Spring Batch ?

Spring Batch est un framework pour le **traitement par lots** (batch processing) :

Qu'est-ce que Spring Batch ?

Spring Batch est un framework pour le **traitement par lots** (batch processing) :

-  **Objectif** : Traiter de gros volumes de données de manière efficace
-  **Cas d'usage** : Import/export, transformation, migration de données
-  **Performance** : Traitement parallèle et optimisé
-  **Fiabilité** : Gestion des erreurs, restart automatique
-  **Monitoring** : Suivi des performances et métriques

Qu'est-ce que Spring Batch ?

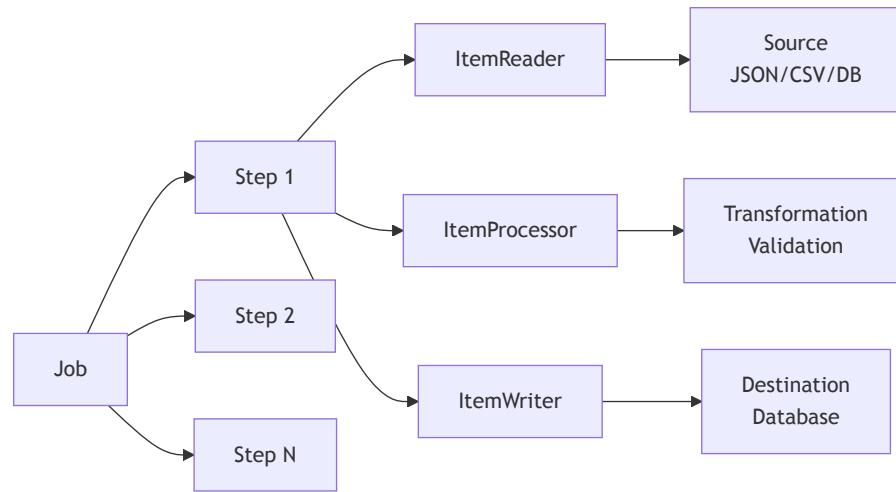
Spring Batch est un framework pour le **traitement par lots** (batch processing) :

-  **Objectif** : Traiter de gros volumes de données de manière efficace
-  **Cas d'usage** : Import/export, transformation, migration de données
-  **Performance** : Traitement parallèle et optimisé
-  **Fiabilité** : Gestion des erreurs, restart automatique
-  **Monitoring** : Suivi des performances et métriques

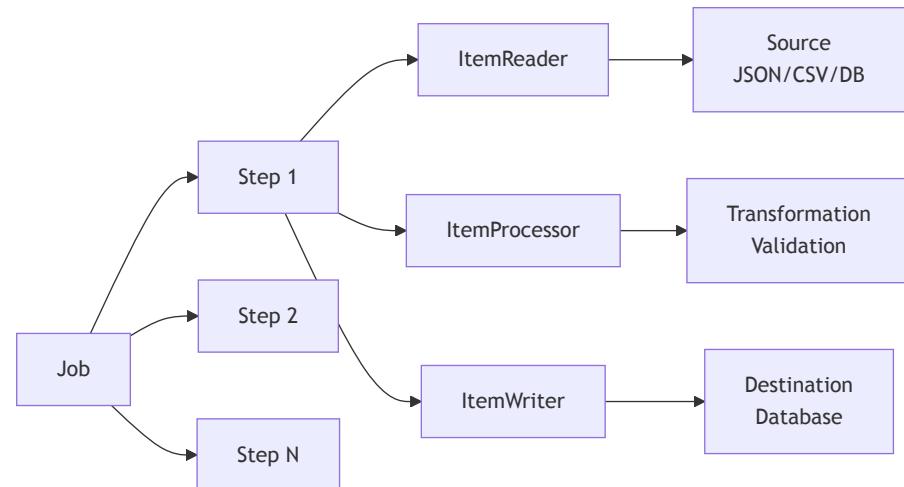
Pourquoi utiliser Spring Batch ?

- Traitement de millions d'enregistrements
- Intégration native avec Spring Boot
- Gestion transactionnelle robuste
- Architecture extensible

Architecture Spring Batch

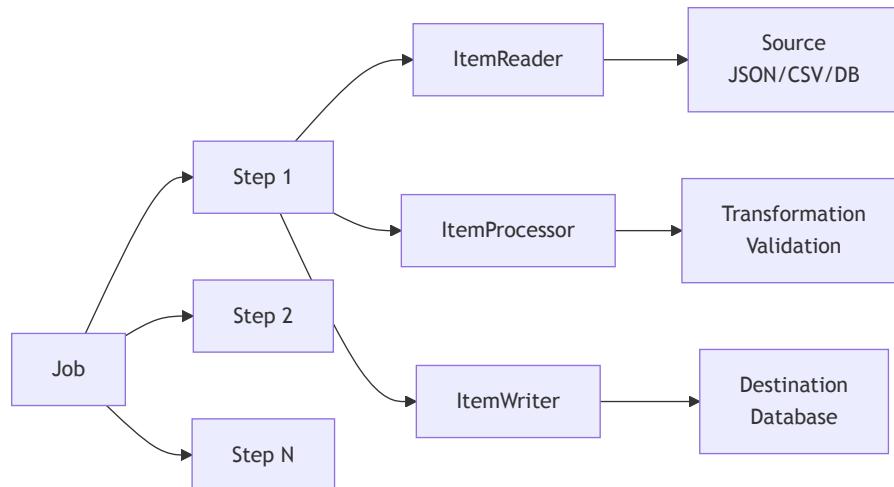


Architecture Spring Batch



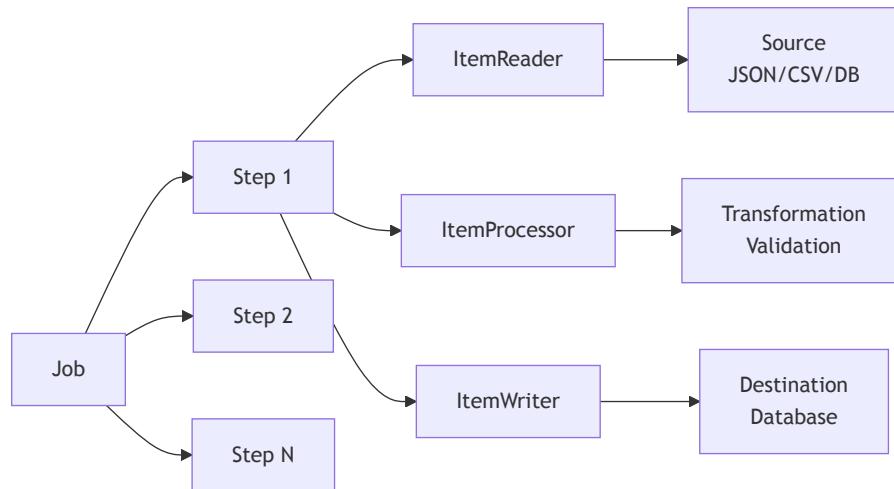
- **Job** : Unité de travail complète

Architecture Spring Batch



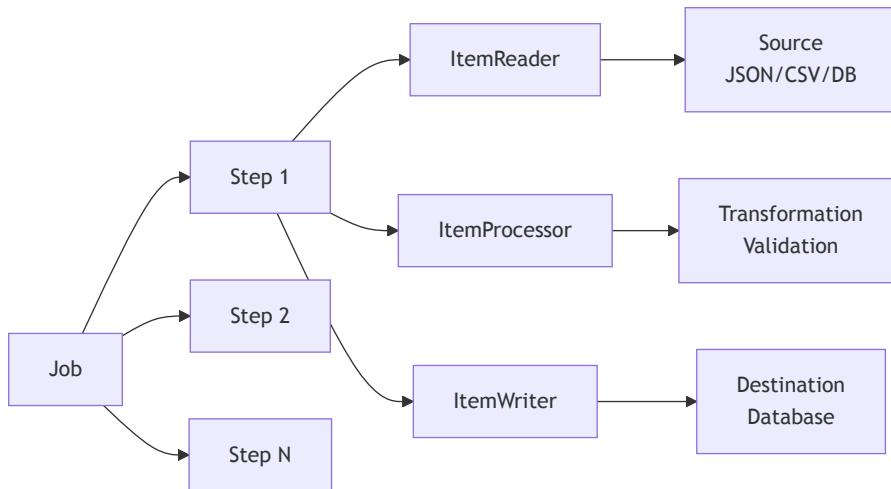
- **Job** : Unité de travail complète
- **Step** : Étape d'un job (lecture → traitement → écriture)

Architecture Spring Batch



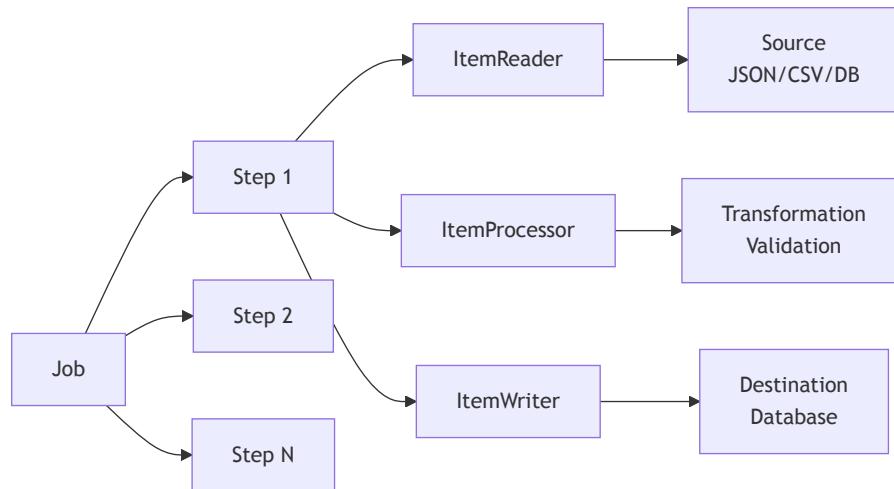
- **Job** : Unité de travail complète
- **Step** : Étape d'un job (lecture → traitement → écriture)
- **ItemReader** : Lit les données source

Architecture Spring Batch



- **Job** : Unité de travail complète
- **Step** : Étape d'un job (lecture → traitement → écriture)
- **ItemReader** : Lit les données source
- **ItemProcessor** : Transforme/valide les données

Architecture Spring Batch



- **Job** : Unité de travail complète
- **Step** : Étape d'un job (lecture → traitement → écriture)
- **ItemReader** : Lit les données source
- **ItemProcessor** : Transforme/valide les données
- **ItemWriter** : Écrit les données de sortie

Notre projet : Import d'événements OpenAgenda

Objectif

Importer des événements depuis l'API OpenAgenda vers notre base de données



Données d'entrée

```
1  {
2      "total_count": 843786,
3      "results": [
4          {
5              "uid": "53816212",
6              "title_fr": "Conférence Tech 2025",
7              "description_fr": "Une conférence sur les technologies",
8              "firstdate_begin": "2025-03-20T09:00:00+00:00",
9              "location_name": "Centre de conférences Paris",
10             "location_address": "123 Rue de la Tech"
11         }
12     ]
13 }
```

Étape 1 : Configuration du projet

Dépendances Maven

```
1  <!-- Spring Batch -->
2  <dependency>
3      <groupId>org.springframework.boot</groupId>
4      <artifactId>spring-boot-starter-batch</artifactId>
5  </dependency>
```

Configuration Spring Boot

```
1  spring:
2      batch:
3          job:
4              enabled: false # Désactive le démarrage automatique
```

Étape 2 : Activation de Spring Batch



Configuration avec Spring Boot 3.x

⚠ **Changement majeur** : Plus besoin de `@EnableBatchProcessing` avec Spring Boot 3.x

```
1  @Configuration
2  @RequiredArgsConstructor
3  public class BatchConfig {
4      // Spring Boot 3.x configure automatiquement Spring Batch
5      // Il suffit de déclarer nos Beans Job, Step, etc.
6  }
```



Application principale

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



Configuration application.yaml

```
1  spring:
2      batch:
3          job:
4              enabled: false # Désactive le démarrage automatique
5          jdbc:
6              initialize-schema: always # Pour H2, utiliser 'embedded' pour production
```

Étape 3 : Modélisation des données



DTO pour OpenAgenda avec @JsonIgnoreProperties

★ Annotation clé : `@JsonIgnoreProperties(ignoreUnknown = true)`

```
1  @Data
2  @JsonIgnoreProperties(ignoreUnknown = true) // 🔑 ESSENTIEL !
3  public class OpenAgendaEventDTO {
4      private String uid;
5
6      @JsonProperty("title_fr")
7      private String titleFr;
8
9      @JsonProperty("description_fr")
10     private String descriptionFr;
11
12     @JsonProperty("longdescription_fr")
13     private String longDescriptionFr;
14
15     @JsonProperty("firstdate_begin")
16     private String firstDateBegin;
17
18     @JsonProperty("firstdate_end")
19     private String firstDateEnd;
20
21     @JsonProperty("location_address")
22     private String locationAddress;
23
24     @JsonProperty("location_city")
25     private String locationCity;
26 }
```



💡 Avantage : Ignore les champs JSON non mappés (crucial pour OpenAgenda !)

Étape 4 : ItemReader - Lecture des données

```
1  public class JsonItemReader implements ItemReader<OpenAgendaEventDTO> {
2      private final Resource resource;
3      private final ObjectMapper objectMapper;
4      private Iterator<OpenAgendaEventDTO> eventIterator;
5
6      @Override
7      public OpenAgendaEventDTO read() throws Exception {
8          if (eventIterator == null) {
9              initializeReader(); // Parse le JSON une seule fois
10         }
11
12         if (eventIterator != null && eventIterator.hasNext()) {
13             return eventIterator.next();
14         }
15
16         closeReader();
17         return null; // Fin de lecture
18     }
19 }
```

Étape 4 : ItemReader - Lecture des données

```
1  public class JsonItemReader implements ItemReader<OpenAgendaEventDTO> {
2      private final Resource resource;
3      private final ObjectMapper objectMapper;
4      private Iterator<OpenAgendaEventDTO> eventIterator;
5
6      @Override
7      public OpenAgendaEventDTO read() throws Exception {
8          if (eventIterator == null) {
9              initializeReader(); // Parse le JSON une seule fois
10         }
11
12         if (eventIterator != null && eventIterator.hasNext()) {
13             return eventIterator.next();
14         }
15
16         closeReader();
17         return null; // Fin de lecture
18     }
19 }
```

 **Principe** : Lit élément par élément, pas tout en mémoire

Étape 5 : ItemProcessor - Transformation

```
1  @Component
2  public class OpenAgendaEventProcessor implements ItemProcessor<OpenAgendaEventDTO, EventEntity> {
3
4      @Override
5      public EventEntity process(OpenAgendaEventDTO item) {
6          if (item == null || item.getTitleFr() == null) {
7              return null; // Skip invalid items
8          }
9
10         EventEntity entity = new EventEntity();
11         entity.setTitle(truncate(item.getTitleFr(), 255));
12
13         // Nettoyage HTML
14         String description = stripHtml(item.getDescriptionFr());
15         entity.setDescription(truncate(description, 1000));
16
17         // Parse des dates
18         entity.setStartDate(parseDateTime(item.getFirstDateBegin()));
19
20         // Construction de l'adresse
21         entity.setLocation(buildLocationString(item));
22
23         return entity;
24     }
25 }
```

Méthodes utilitaires du Processor

```
1  private LocalDateTime parseDateTime(String dateTimeStr) {
2      if (dateTimeStr == null || dateTimeStr.trim().isEmpty()) {
3          return null;
4      }
5      try {
6          return LocalDateTime.parse(dateTimeStr, ISO_FORMATTER);
7      } catch (DateTimeParseException e) {
8          logger.warn("Failed to parse date: {}", dateTimeStr);
9          return null;
10     }
11 }
12
13 private String stripHtml(String html) {
14     if (html == null) return null;
15     return html.replaceAll("<[^>]*>", "").trim();
16 }
17
18 private String truncate(String str, int maxLength) {
19     if (str == null) return null;
20     return str.length() > maxLength ? str.substring(0, maxLength) : str;
21 }
```

Étape 6 : ItemWriter - Sauvegarde

```
1  @Component
2  @RequiredArgsConstructor // 🔑 Lombok pour injection
3  public class EventItemWriter implements ItemWriter<EventEntity> {
4
5      // 🔑 Injection via constructeur (Spring Boot 3.x)
6      private final EventRepository eventRepository;
7
8      @Override
9      public void write(Chunk<? extends EventEntity> chunk) throws Exception {
10          logger.info("Writing {} events to database", chunk.size());
11
12          for (EventEntity event : chunk.getItems()) {
13              try {
14                  eventRepository.save(event);
15                  logger.debug("Saved event: {}", event.getTitle());
16              } catch (Exception e) {
17                  logger.error("Failed to save event: {}", event.getTitle());
18                  throw e; // Provoque un rollback du chunk
19              }
20          }
21
22          logger.info("Successfully wrote {} events", chunk.size());
23      }
24  }
```

Étape 7 : Configuration du Listener

```
1  @Component
2  public class JobExecutionListenerImpl implements JobExecutionListener {
3
4      private static final Logger logger = LoggerFactory.getLogger(JobExec
5
6      @Override
7      public void beforeJob(JobExecution jobExecution) {
8          logger.info("==> DÉBUT DU JOB ==>");
9          logger.info("Job Name: {}", jobExecution.getJobInstance().getJob
10         logger.info("Job ID: {}", jobExecution.getJobId());
11         logger.info("Start Time: {}", jobExecution.getStartTime());
12         logger.info("Job Parameters: {}", jobExecution.getJobParameters()
13     }
```

```
14     ...
15     @Override
16     public void afterJob(JobExecution jobExecution) {
17         logger.info("==> FIN DU JOB ==>");
18         logger.info("Job Name: {}", jobExecution.getJobInstance().getJob
19         logger.info("Job Status: {}", jobExecution.getStatus());
20         logger.info("End Time: {}", jobExecution.getEndTime());
21
22         if (jobExecution.getEndTime() != null && jobExecution.getStartTi
23             LocalDateTime startTime = jobExecution.getStartTime().toInst
24                 .atZone(ZoneId.systemDefault()).toLocalDateTime();
25             LocalDateTime endTime = jobExecution.getEndTime().toInstant(
26                 .atZone(ZoneId.systemDefault()).toLocalDateTime();
27             Duration duration = Duration.between(startTime, endTime);
28             logger.info("Duration: {} ms", duration.toMillis());
29         } else {
30             logger.info("Duration: N/A");
31         }
32
33         if (jobExecution.getStatus().isUnsuccessful()) {
34             logger.error("Job failed with exit description: {}", jobExec
35         } else {
36             logger.info("Job completed successfully!");
37         }
38
39         logger.info("Read Count: {}", jobExecution.getStepExecutions().s
40             .mapToLong(StepExecution::getReadCount).sum());
41         logger.info("Write Count: {}", jobExecution.getStepExecutions().s
42             .mapToLong(StepExecution::getWriteCount).sum());
43         logger.info("Skip Count: {}", jobExecution.getStepExecutions().s
44             .mapToLong(StepExecution::getSkipCount).sum());
45     }
46 }
```

Étape 8 : Configuration du Job (Mono-thread)

```
1  @Configuration
2  @RequiredArgsConstructor // 🔑 Lombok pour injection
3  public class BatchConfig {
4
5      // 🔑 Injection via constructeur (Spring Boot 3.x)
6      private final OpenAgendaEventProcessor eventProcessor;
7      private final EventItemWriter eventWriter;
8      private final JobExecutionListenerImpl jobExecutionListener;
9
10     @Bean
11     public JsonItemReader jsonItemReader() {
12         return new JsonItemReader(new ClassPathResource("data/events.json"));
13     }
14
15     @Bean
16     public Step importEventStep(JobRepository jobRepository,
17                                 PlatformTransactionManager transactionManager) {
18         return new StepBuilder("importEventStep", jobRepository)
19             .<OpenAgendaEventDTO, EventEntity>chunk(100, transactionManager) // 📊 100 éléments par chunk
20             .reader(jsonItemReader())
21             .processor(eventProcessor)
22             .writer(eventWriter)
23             .build();
24     }
25
26     @Bean
27     public Job importEventJob(JobRepository jobRepository, Step importEventStep) {
28         return new JobBuilder("importEventJob", jobRepository)
29             .start(importEventStep)
30             .listener(jobExecutionListener)
31             .build();
32     }
33 }
```

Étape 9 : Configuration Multi-threading

Optimisation avec TaskExecutor

```
1  @Configuration
2  @RequiredArgsConstructor
3  @Slf4j
4  public class OptimizedBatchConfig {
5
6      @Bean("batchTaskExecutor") // ⚒ Bean nommé avec @Qualifier
7      public TaskExecutor batchTaskExecutor() {
8          ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
9          executor.setCorePoolSize(4);           // 4 threads minimum
10         executor.setMaxPoolSize(8);          // 8 threads maximum
11         executor.setQueueCapacity(100);        // File d'attente
12         executor.setThreadNamePrefix("batch-");
13         executor.setWaitForTasksToCompleteOnShutdown(true);
14         executor.setAwaitTerminationSeconds(60);
15
16         executor.initialize(); // ⚒ CRUCIAL !
17
18         log.info("TaskExecutor configured: core={}, max={}, queue={}",
19                 executor.getCorePoolSize(), executor.getMaxPoolSize(),
20                 executor.getQueueCapacity());
21
22         return executor;
23     }
24 }
```

Configuration Step Multi-thread

```
1  @Bean
2  public Step optimizedImportStep(JobRepository jobRepository,
3          PlatformTransactionManager transactionManager,
4          OpenAgendaEventProcessor processor,
5          @Qualifier("optimizedEventItemWriter") ItemWriter<EventEntity> writer, // ⚡ @Qualifier !
6          @Qualifier("batchTaskExecutor") TaskExecutor taskExecutor) {           // ⚡ @Qualifier !
7
8      return new StepBuilder("optimizedImportStep", jobRepository)
9          .<OpenAgendaEventDTO, EventEntity>chunk(100, transactionManager)
10         .reader(optimizedJsonItemReader())
11         .processor(processor)
12         .writer(writer)
13         .taskExecutor(taskExecutor) // ⚡ Activation du multi-threading !
14         .build();
15     }
16
17    @Bean
18    public Job optimizedImportJob(JobRepository jobRepository, Step optimizedImportStep) {
19        return new JobBuilder("optimizedImportJob", jobRepository)
20            .start(optimizedImportStep)
21            .build();
22    }
```

ItemReader Thread-Safe

Problématique du Multi-threading

```
1  public class ThreadSafeJsonItemReader implements ItemReader<OpenAgendaEventDTO> {
2
3      // ↴ Variables thread-safe avec volatile et AtomicInteger
4      private volatile List<OpenAgendaEventDTO> events;
5      private final AtomicInteger currentIndex = new AtomicInteger(0);
6      private final ReentrantLock initializationLock = new ReentrantLock();
7      private volatile boolean initialized = false;
8
9      @Override
10     public OpenAgendaEventDTO read() throws Exception {
11         // 🔒 Double-checked locking pour initialisation
12         if (!initialized) {
13             initializationLock.lock();
14             try {
15                 if (!initialized) {
16                     initializeReader(); // Parse une seule fois !
17                     initialized = true;
18                 }
19             } finally {
20                 initializationLock.unlock();
21             }
22         }
23
24         // 🚀 Lecture thread-safe avec AtomicInteger
25         if (events != null) {
26             int index = currentIndex.getAndIncrement();
27             if (index < events.size()) {
28                 return events.get(index);
29             }
30         }
31         return null;
32     }
33 }
```

API REST pour déclencher les jobs

```
1  @RestController
2  @RequestMapping("/api/batch")
3  @RequiredArgsConstructor // 🔑 Lombok
4  public class BatchController {
5
6      private final JobLauncher jobLauncher;
7
8      @Qualifier("importEventJob")          // 🚀 Job mono-thread
9      private final Job importEventJob;
10     @Qualifier("optimizedImportJob")    // 🚀 Job multi-thread
11     private final Job optimizedImportJob;
12     private final JobExplorer jobExplorer;
13
14     @PostMapping("/import-events")
15     public ResponseEntity<Map<String, String>> importEvents() {
16         Map<String, String> response = new HashMap<>();
17         try {
18             JobParameters jobParameters = new JobParametersBuilder()
19                 .addLong("startAt", System.currentTimeMillis())
20                 .toJobParameters();
21
22             jobLauncher.run(importEventJob, jobParameters);
23
24             response.put("status", "SUCCESS");
25             response.put("message", "Import job started successfully");
26             return ResponseEntity.ok(response);
27         } catch (Exception e) {
28             response.put("status", "ERROR");
29             response.put("message", "Failed to start import job: " + e.getMessage());
30             return ResponseEntity.internalServerError().body(response);
31         }
32     }
```

```
33     ...
34     @PostMapping("/import-events-fast")    // 🚶 Endpoint optimisé
35     public ResponseEntity<Map<String, String>> importEventsFast() {
36         Map<String, String> response = new HashMap<>();
37         try {
38             JobParameters jobParameters = new JobParametersBuilder()
39                 .addLong("startAt", System.currentTimeMillis())
40                 .toJobParameters();
41
42             jobLauncher.run(optimizedImportJob, jobParameters); // 🚀
43
44             response.put("status", "SUCCESS");
45             response.put("message", "Optimized import job started successfully");
46             return ResponseEntity.ok(response);
47         } catch (Exception e) {
48             response.put("status", "ERROR");
49             response.put("message", "Failed to start optimized import job: " + e.getMessage());
50             return ResponseEntity.internalServerError().body(response);
51         }
52     }
53 }
```

Fonctionnalités avancées : Monitoring

JobExecutionListener

```
1  @Component
2  public class JobExecutionListenerImpl implements JobExecutionListener {
3
4      @Override
5      public void beforeJob(JobExecution jobExecution) {
6          logger.info("==> DÉBUT DU JOB ==>");
7          logger.info("Job Name: {}", jobExecution.getJobInstance().getJobName());
8          logger.info("Start Time: {}", jobExecution.getStartTime());
9      }
10
11     @Override
12     public void afterJob(JobExecution jobExecution) {
13         logger.info("==> FIN DU JOB ==>");
14         logger.info("Job Status: {}", jobExecution.getStatus());
15         logger.info("Duration: {} ms",
16                     jobExecution.getEndTime().getTime() - jobExecution.getStartTime().getTime());
17         logger.info("Read Count: {}", jobExecution.getStepExecutions().stream()
18                    .mapToLong(step -> step.getReadCount()).sum());
19         logger.info("Write Count: {}", jobExecution.getStepExecutions().stream()
20                    .mapToLong(step -> step.getWriteCount()).sum());
21     }
22 }
```

API d'historique des jobs

```
1  @GetMapping("/job-history")
2  public ResponseEntity<List<Map<String, Object>>> getJobHistory() {
3      List<JobExecution> jobExecutions = jobExplorer
4          .findJobInstancesByJobName("importEventJob", 0, 10)
5          .stream()
6          .flatMap(instance -> jobExplorer.getJobExecutions(instance).stream())
7          .toList();
8
9      List<Map<String, Object>> history = jobExecutions.stream()
10         .map(execution -> {
11             Map<String, Object> jobInfo = new HashMap<>();
12             jobInfo.put("id", execution.getJobId());
13             jobInfo.put("status", execution.getStatus().toString());
14             jobInfo.put("startTime", execution.getStartTime());
15             jobInfo.put("endTime", execution.getEndTime());
16             jobInfo.put("readCount", execution.getStepExecutions().stream()
17                 .mapToLong(step -> step.getReadCount()).sum());
18             jobInfo.put("writeCount", execution.getStepExecutions().stream()
19                 .mapToLong(step -> step.getWriteCount()).sum());
20             return jobInfo;
21         })
22         .toList();
23
24     return ResponseEntity.ok(history);
25 }
```

Fonctionnalités avancées : Scheduling

Planification automatique

```
1  @Service
2  @RequiredArgsConstructor
3  public class BatchSchedulerService {
4
5      private final JobLauncher jobLauncher;
6
7      @Qualifier("importEventJob")
8      private final Job importEventJob;
9
10     @Scheduled(fixedRate = 3600000) // Toutes les heures
11     public void runImportJobHourly() {
12         logger.info("Lancement automatique du job d'import");
13
14         try {
15             JobParameters jobParameters = new JobParametersBuilder()
16                     .addLong("scheduledAt", System.currentTimeMillis())
17                     .toJobParameters();
18
19             jobLauncher.run(importEventJob, jobParameters);
20         } catch (Exception e) {
21             logger.error("Erreur lors de l'exécution du job planifié: {}", e.getMessage());
22         }
23     }
24
25     @Scheduled(cron = "0 0 2 * * ?") // Tous les jours à 2h du matin
26     public void runDailyMaintenance() {
27         logger.info("Maintenance quotidienne des données");
28     }
29 }
```

Fonctionnalités avancées : Parallélisme

Configuration du traitement parallèle

```
1  @Bean
2  public TaskExecutor taskExecutor() {
3      ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
4      executor.setCorePoolSize(4);          // 4 threads minimum
5      executor.setMaxPoolSize(8);          // 8 threads maximum
6      executor.setQueueCapacity(25);        // File d'attente
7      executor.setThreadNamePrefix("batch-");
8      executor.initialize();
9      return executor;
10 }
11
12 @Bean
13 public Step partitionedStep(JobRepository jobRepository,
14                             PlatformTransactionManager transactionManager) {
15     return new StepBuilder("partitionedStep", jobRepository)
16         .partitioner("csvImportStep", eventPartitioner())
17         .step(csvImportStep(jobRepository, transactionManager))
18         .gridSize(4)                  // 4 partitions
19         .taskExecutor(taskExecutor()) // Exécution parallèle
20         .build();
21 }
```

Job multi-étapes

```
1  @Bean
2  public Step reportingStep(JobRepository jobRepository,
3                           PlatformTransactionManager transactionManager) {
4      return new StepBuilder("reportingStep", jobRepository)
5          .tasklet((contribution, chunkContext) -> {
6              System.out.println("==== RAPPORT D'IMPORT ====");
7              System.out.println("Job exécuté avec succès !");
8              System.out.println("Heure de fin: " + new Date());
9              return null;
10         }, transactionManager)
11         .build();
12     }
13
14    @Bean
15    public Job multiStepJob(JobRepository jobRepository,
16                           PlatformTransactionManager transactionManager) {
17        return new JobBuilder("multiStepJob", jobRepository)
18            .start(partitionedStep(jobRepository, transactionManager))
19            .next(reportingStep(jobRepository, transactionManager))
20            .build();
21    }
```

Test et utilisation

Tests disponibles

```
1 # Démarrer l'application
2 mvn spring-boot:run
3
4 # Test 1: Import mono-thread (version basique)
5 curl -X POST http://localhost:8080/api/batch/import-events
6
7 # Test 2: Import multi-thread (version optimisée) 🚀
8 curl -X POST http://localhost:8080/api/batch/import-events-fast
9
10 # Consulter l'historique des jobs
11 curl http://localhost:8080/api/batch/job-history
```

Comparaison des performances

Mono-thread

```
1 INFO - Job Name: importEventJob
2 INFO - Processing 843786 events...
3 INFO - Processed 100000 events
4 INFO - Processed 200000 events
5 INFO - Duration: ~45 minutes
6 INFO - Single thread processing
```

Multi-thread (4-8 threads)

```
1 INFO - Job Name: optimizedImportJob
2 INFO - TaskExecutor: core=4, max=8
3 INFO - Thread batch-1 processing: 25000/843786
4 INFO - Thread batch-2 processing: 25000/843786
5 INFO - Thread batch-3 processing: 25000/843786
6 INFO - Duration: ~15 minutes ↴
7 INFO - Performance gain: 3x faster!
```

Monitoring en temps réel

```
1 INFO - === DÉBUT DU JOB ===
2 INFO - 📈 Initializing JSON reader for multi-threaded processing...
3 INFO - ✅ Total events loaded: 843786 in 12456ms - Ready for multi-threading!
4 INFO - 🚀 Thread batch-1 processing: 1000/843786 (0.1%)
5 INFO - 🚀 Thread batch-2 processing: 1000/843786 (0.1%)
6 INFO - Saved 100 events (Total: 100)
7 INFO - Saved 100 events (Total: 200)
8 INFO - Processed 1000 events
9 INFO - === FIN DU JOB ===
10 INFO - Job Status: COMPLETED
11 INFO - Read Count: 843786
12 INFO - Write Count: 840123 # Quelques événements invalides ignorés
```

Annotations clés à maîtriser

Spring Batch Core

- `@Configuration` : Classe de config Spring
- `@RequiredArgsConstructor` : Injection Lombok
- `@Slf4j` : Logger automatique
- `@Component` : Bean Spring auto-détecté
- `@Transactional` : Gestion des transactions

Injection de dépendances

- `@Qualifier("beanName")` : Spécifie quel bean injecter
- `@Bean("customName")` : Nomme explicitement un bean
- `private final` : Injection par constructeur (immutable)

Mapping JSON

- `@JsonProperty("json_field")` : Mappe champ JSON
- `@JsonIgnoreProperties(ignoreUnknown = true)` : Ignore champs inconnus
- `@Data` : Génère getters/setters/toString
- `@AllArgsConstructor` : Constructeur avec tous les champs

Multi-threading

- `volatile` : Visibilité entre threads
- `AtomicInteger` : Compteur thread-safe
- `ReentrantLock` : Verrou réentrant
- `ThreadPoolTaskExecutor` : Pool de threads

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread).



Phase 2 : Optimisations (Multi-thread).

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread).

1. Ajouter `spring-boot-starter-batch` au pom.xml



Phase 2 : Optimisations (Multi-thread).

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread).

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)



Phase 2 : Optimisations (Multi-thread).

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée
10. Configurer `TaskExecutor` avec `@Bean("nom")`

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée
10. Configurer `TaskExecutor` avec `@Bean("nom")`
11. Développer `ThreadSafeJsonItemReader` avec `AtomicInteger`

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée
10. Configurer `TaskExecutor` avec `@Bean("nom")`
11. Développer `ThreadSafeJsonItemReader` avec `AtomicInteger`
12. Utiliser `@Qualifier` partout pour éviter conflits

Ordre d'implémentation recommandé



Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple



Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée
10. Configurer `TaskExecutor` avec `@Bean("nom")`
11. Développer `ThreadSafeJsonItemReader` avec `AtomicInteger`
12. Utiliser `@Qualifier` partout pour éviter conflits
13. Ajouter endpoint `/import-events-fast`

Ordre d'implémentation recommandé

Phase 1 : Bases (Mono-thread)

1. Ajouter `spring-boot-starter-batch` au pom.xml
2. Configurer `application.yaml` (H2 + batch)
3. Créer `OpenAgendaEventDTO` avec `@JsonIgnoreProperties`
4. Implémenter `JsonItemReader` simple
5. Créer `OpenAgendaEventProcessor` avec validations
6. Développer `EventItemWriter` avec `saveAll()`
7. Configurer `BatchConfig` avec `@RequiredArgsConstructor`
8. Tester via `BatchController` avec endpoint simple

Phase 2 : Optimisations (Multi-thread)

9. Créer `OptimizedBatchConfig` séparée
10. Configurer `TaskExecutor` avec `@Bean("nom")`
11. Développer `ThreadSafeJsonItemReader` avec `AtomicInteger`
12. Utiliser `@Qualifier` partout pour éviter conflits
13. Ajouter endpoint `/import-events-fast`
14. Comparer performances mono vs multi-thread

Problèmes courants et solutions

Erreurs fréquentes

Bean conflicts

```
1 Multiple beans of type Job found
```

➡ Solution : `@Qualifier("jobName")`

JSON parsing errors

```
1 Unrecognized token 'de'
```

➡ Solution : `@JsonIgnoreProperties(ignoreUnknown = true)`

Thread safety issues

```
1 Concurrent modification exception
```

➡ Solution : `ThreadSafeItemReader` avec `volatile`

Bonnes pratiques

Injection de dépendances

```
1 @RequiredArgsConstructor // Lombok
2 private final JobLauncher jobLauncher;
```

Configuration séparée

```
1 BatchConfig.java           // Version simple
2 OptimizedBatchConfig.java // Version optimisée
```

Naming des beans

```
1 @Bean("batchTaskExecutor") // Nom explicite
2 @Qualifier("batchTaskExecutor") // Utilisation
```

Métadonnées Spring Batch

Spring Batch crée automatiquement des tables pour tracer les exécutions :

Tables créées

- `BATCH_JOB_INSTANCE`
- `BATCH_JOB_EXECUTION`
- `BATCH_STEP_EXECUTION`
- `BATCH_JOB_EXECUTION_PARAMS`
- `BATCH_JOB_EXECUTION_CONTEXT`
- `BATCH_STEP_EXECUTION_CONTEXT`

Informations stockées

- Statut des jobs (STARTED, COMPLETED, FAILED)
- Paramètres d'exécution
- Temps de début/fin
- Compteurs (read, write, skip)
- Contexte d'exécution
- Messages d'erreur

Métadonnées Spring Batch

Spring Batch crée automatiquement des tables pour tracer les exécutions :

Tables créées

- `BATCH_JOB_INSTANCE`
- `BATCH_JOB_EXECUTION`
- `BATCH_STEP_EXECUTION`
- `BATCH_JOB_EXECUTION_PARAMS`
- `BATCH_JOB_EXECUTION_CONTEXT`
- `BATCH_STEP_EXECUTION_CONTEXT`



Avantage : Restart automatique en cas d'échec, reprise là où ça s'est arrêté

Informations stockées

- Statut des jobs (STARTED, COMPLETED, FAILED)
- Paramètres d'exécution
- Temps de début/fin
- Compteurs (read, write, skip)
- Contexte d'exécution
- Messages d'erreur

Gestion des erreurs et restart

X Types d'erreurs gérées

Configuration du restart

```
1  @Bean
2  public Step robustStep(JobRepository jobRepository,
3                         PlatformTransactionManager transactionManager) {
4      return new StepBuilder("robustStep", jobRepository)
5          .<OpenAgendaEventDTO, EventEntity>chunk(10, transactionManager)
6          .reader(jsonItemReader())
7          .processor(eventProcessor)
8          .writer(eventWriter)
9          .faultTolerant()
10         .skipLimit(5)           // Max 5 erreurs ignorées
11         .skip(ValidationException.class) // Type d'erreur à ignorer
12         .retryLimit(3)          // Max 3 tentatives
13         .retryTransientDataAccessException()
14         .build();
15 }
```

Gestion des erreurs et restart

X Types d'erreurs gérées

- **Skip** : Ignorer un élément en erreur et continuer

Configuration du restart

```
1  @Bean
2  public Step robustStep(JobRepository jobRepository,
3                         PlatformTransactionManager transactionManager) {
4      return new StepBuilder("robustStep", jobRepository)
5          .<OpenAgendaEventDTO, EventEntity>chunk(10, transactionManager)
6          .reader(jsonItemReader())
7          .processor(eventProcessor)
8          .writer(eventWriter)
9          .faultTolerant()
10         .skipLimit(5)           // Max 5 erreurs ignorées
11         .skip(ValidationException.class) // Type d'erreur à ignorer
12         .retryLimit(3)           // Max 3 tentatives
13         .retryTransientDataAccessException()
14         .build();
15 }
```

Gestion des erreurs et restart

X Types d'erreurs gérées

- **Skip** : Ignorer un élément en erreur et continuer
- **Retry** : Réessayer en cas d'erreur temporaire

Configuration du restart

```
1  @Bean
2  public Step robustStep(JobRepository jobRepository,
3                         PlatformTransactionManager transactionManager) {
4      return new StepBuilder("robustStep", jobRepository)
5          .<OpenAgendaEventDTO, EventEntity>chunk(10, transactionManager)
6          .reader(jsonItemReader())
7          .processor(eventProcessor)
8          .writer(eventWriter)
9          .faultTolerant()
10         .skipLimit(5)           // Max 5 erreurs ignorées
11         .skip(ValidationException.class) // Type d'erreur à ignorer
12         .retryLimit(3)           // Max 3 tentatives
13         .retryTransientDataAccessException()
14         .build();
15 }
```

Gestion des erreurs et restart

X Types d'erreurs gérées

- **Skip** : Ignorer un élément en erreur et continuer
- **Retry** : Réessayer en cas d'erreur temporaire
- **Restart** : Reprendre un job arrêté là où il s'est arrêté

Configuration du restart

```
1  @Bean
2  public Step robustStep(JobRepository jobRepository,
3                         PlatformTransactionManager transactionManager) {
4      return new StepBuilder("robustStep", jobRepository)
5          .<OpenAgendaEventDTO, EventEntity>chunk(10, transactionManager)
6          .reader(jsonItemReader())
7          .processor(eventProcessor)
8          .writer(eventWriter)
9          .faultTolerant()
10         .skipLimit(5)           // Max 5 erreurs ignorées
11         .skip(ValidationException.class) // Type d'erreur à ignorer
12         .retryLimit(3)           // Max 3 tentatives
13         .retryTransientDataAccessException()
14         .build();
15 }
```

Optimisations et bonnes pratiques

Performance

- **Chunk size** optimal : 100-1000
- **Pagination** pour gros volumes
- **Index** sur colonnes de recherche
- **Connection pooling** configuré
- **Traitement parallèle** si possible

Monitoring

- **Logs structurés** avec niveaux
- **Métriques** custom avec Micrometer
- **Alerts** sur échecs de jobs
- **Dashboard** de supervision

Sécurité

- **Validation** des données d'entrée
- **Authentification** sur endpoints
- **Chiffrement** des données sensibles
- **Audit trail** complet

Tests

- **Tests unitaires** des processors
- **Tests d'intégration** end-to-end
- **Tests de performance** avec gros volumes
- **Tests de reprise** après échec

Architecture complète du projet réel

```
1  src/main/java/com/formation/events/
2  └── batch/
3  |   └── JsonItemReader.java
4  |   └── ThreadSafeJsonItemReader.java
5  |   └── OpenAgendaEventProcessor.java
6  |   └── EventItemWriter.java
7  |   └── OptimizedEventItemWriter.java
8  |   └── JobExecutionListenerImpl.java
9  └── config/
10    └── BatchConfig.java
11    └── OptimizedBatchConfig.java
12    └── SystemDataInitializer.java
13  └── controllers/
14    └── BatchController.java
15  └── dtos/openagenda/
16    └── OpenAgendaEventDTO.java
17  └── entities/
18    └── EventEntity.java
19    └── UserEntity.java
20  └── repositories/
21    └── EventRepository.java
22    └── UserRepository.java
23  └── resources/
24    └── application.yaml
25    └── data/events.json
```

📄 Lecture JSON mono-thread
🔒 Lecture JSON thread-safe
⚙ Transformation + validation
📁 Writer basique
🚀 Writer optimisé (@Component("optimizedEventItemWriter"))
📈 Monitoring + métriques

🎯 Configuration mono-thread
⚡ Configuration multi-thread
📁 Données initiales

🌐 API REST (/import-events + /import-events-fast)

📄 DTO avec @JsonIgnoreProperties

🏢 Entité JPA Events
🧑 Entité JPA Users (organizer)

📈 JPA Repository Events
🧑 JPA Repository Users

⚙ Config H2 + Spring Batch
📁 Fichier JSON 3.4GB OpenAgenda

Dépendances clés

- **Spring Boot 3.5.0 + Spring Batch**
- **H2 Database** (metadata tables)
- **Jackson** (JSON parsing avec @JsonIgnoreProperties)
- **Lombok** (@RequiredArgsConstructor, @Slf4j, @Data)

Cas d'usage avancés



Import massif

- Millions d'enregistrements
- Partitioning par date/région
- Traitement parallèle
- Monitoring temps réel



Reporting

- Agrégation de données
- Calculs complexes
- Export vers BI
- Génération de fichiers



Synchronisation

- Delta import (que les nouveautés)
- Détection des changements
- Réconciliation des données
- Gestion des suppressions



Maintenance

- Nettoyage de données
- Archivage automatique
- Purge des logs
- Optimisation des index

Fonctionnalité avancée : Export CSV avec Partitioning

Export des événements depuis la BDD vers CSV

Notre projet inclut maintenant l'**export des événements depuis la base de données vers des fichiers CSV**, avec **partitioning par année** pour optimiser les performances.

EventBDDItemReader - Lecture depuis BDD

```
1  @Component
2  @StepScope // 🔑 CRUCIAL pour injection de paramètres dynamiques !
3  public class EventBDDItemReader implements ItemReader<EventBddDTO> {
4
5      @Value("#{stepExecutionContext['year'])") // 🔑 Paramètre injecté par le partitioner
6      private String year;
7
8      @Value("#{stepExecutionContext['minId'])")
9      private Long minId;
10
11     @Value("#{stepExecutionContext['maxId'])")
12     private Long maxId;
13
14     private final EventRepository eventRepository;
15     private Iterator<EventBddDTO> eventIterator;
16
17     @Override
18     public EventBddDTO read() throws Exception {
19         if (eventIterator == null) {
20             initializeIterator(); // Charge les événements de cette partition
21         }
22
23         return eventIterator.hasNext() ? eventIterator.next() : null;
24     }
25
26     private void initializeIterator() {
27         // 🔎 Requête filtrée par année et plage d'IDs pour cette partition
28         List<EventEntity> events = eventRepository.findByYearAndIdRange(year, minId, maxId);
29
30         // 📈 Conversion Entity → DTO pour l'export
31         List<EventBddDTO> dtos = events.stream()
32             .map(this::convertToDTO)
33             .collect(Collectors.toList());
34
35         this.eventIterator = dtos.iterator();
36         logger.info("Initialized reader for year {} with {} events (IDs {}-{})",
37                     year, dtos.size(), minId, maxId);
38     }
39 }
```

EventBDDItemWriter - Export vers CSV avec headers

```
1  @Component
2  @StepScope // 🎉 Pour accéder aux paramètres de partition
3  public class EventBDDItemWriter implements ItemWriter<EventBddDTO> {
4
5      @Value("#{stepExecutionContext['year']}")
6      private String year;
7
8      @Value("#{stepExecutionContext['outputFile']}") // Fichier de sortie dynamique
9      private String outputFile;
10
11     private FileWriter fileWriter;
12     private CSVWriter csvWriter;
13     private boolean headerWritten = false;
14
15     @Override
16     public void write(Chunk<? extends EventBddDTO> chunk) throws Exception {
17         if (csvWriter == null) {
18             initializeWriter(); // Création du fichier CSV
19         }
20
21         // 🎉 Écriture du header une seule fois
22         if (!headerWritten) {
23             writeHeader();
24             headerWritten = true;
25         }
26     }
27 }
```

```
26     ...
27     // 📝 Écriture des données
28     for (EventBddDTO event : chunk.getItems()) {
29         String[] record = {
30             event.getId().toString(),
31             event.getTitle(),
32             event.getDescription(),
33             formatDate(event.getStartDate()),
34             event.getLocation(),
35             event.getOrganizerEmail()
36         };
37         csvWriter.writeNext(record);
38     }
39
40     csvWriter.flush(); // ⏴ Force l'écriture sur disque
41     logger.info("Wrote {} events to {}", chunk.size(), outputFile)
42 }
43
44 private void initializeWriter() throws IOException {
45     File file = new File(outputFile);
46     file.getParentFile().mkdirs(); // 📂 Crée les dossiers enfants
47
48     fileWriter = new FileWriter(file);
49     csvWriter = new CSVWriter(fileWriter);
50     logger.info("Initialized CSV writer for file: {}", outputFile)
51 }
52 }
```

Configuration Export CSV avec Partitioning

```
1  @Configuration
2  @RequiredArgsConstructor
3  public class EventCSVBatchConfig {
4
5      private final EventBDDItemReader eventReader;
6      private final EventBDDItemWriter eventWriter;
7
8      // 🔗 Partitioner : Découpe par année (2021, 2022, etc.)
9      @Bean
10     public Partitioner eventPartitioner() {
11         return new Partitioner() {
12             @Override
13             public Map<String, ExecutionContext> partition(int gridSize) {
14                 Map<String, ExecutionContext> partitions = new HashMap<>();
15
16                 // 📈 Crée une partition pour chaque année
17                 String[] years = {"2021", "2022", "2023", "2024", "2025"};
18
19                 for (String year : years) {
20                     ExecutionContext context = new ExecutionContext();
21                     context.putString("year", year);
22                     context.putString("outputFile", "exports/events_" + year + ".csv");
23
24                     // 📈 Calcul des plages d'IDs pour cette année (optionnel)
25                     context.putLong("minId", getMinIdForYear(year));
26                     context.putLong("maxId", getMaxIdForYear(year));
27
28                     partitions.put("partition_" + year, context);
29                 }
30
31                 logger.info("Created {} partitions for years: {}",
32                             partitions.size(), Arrays.toString(years));
33                 return partitions;
34             }
35         };
36     }
```

```
37     ...
38     @Bean
39     public Step exportStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
40         return new StepBuilder("exportStep", jobRepository)
41             .<EventBddDTO, EventBddDTO>chunk(500, transactionManager) // 500
42             .reader(eventReader)
43             .writer(eventWriter)
44             .build();
45     }
46
47     @Bean
48     public Step partitionedExportStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
49         return new StepBuilder("partitionedExportStep", jobRepository)
50             .partitioner("exportStep", eventPartitioner()) // 🔗 Utilise le partitionneur
51             .step(exportStep(jobRepository, transactionManager))
52             .gridSize(5) // 5 partitions en parallèle maximum
53             .taskExecutor(taskExecutor()) // 🚀 Exécution parallèle
54             .build();
55     }
56
57     @Bean
58     public Job partitionedExportJob(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
59         return new JobBuilder("partitionedExportJob", jobRepository)
60             .start(partitionedExportStep(jobRepository, transactionManager))
61             .build();
62     }
63 }
```

Scheduling automatique des exports

⌚ BatchEventController avec Cron

```
1  @RestController
2  @RequestMapping("/api/batch/events")
3  @RequiredArgsConstructor
4  @Slf4j
5  public class BatchEventController {
6
7      private final JobLauncher jobLauncher;
8
9      @Qualifier("partitionedExportJob")
10     private final Job partitionedExportJob;
11
12     // ⏲ Endpoint manuel pour déclencher l'export
13     @PostMapping("/export-partitioned")
14     public ResponseEntity<Map<String, String>> exportEventsPartitioned() {
15         try {
16             JobParameters jobParameters = new JobParametersBuilder()
17                 .addLong("timestamp", System.currentTimeMillis())
18                 .toJobParameters();
19
20             jobLauncher.run(partitionedExportJob, jobParameters);
21
22             Map<String, String> response = new HashMap<>();
23             response.put("status", "SUCCESS");
24             response.put("message", "Partitioned export job started - files will be created in exports/");
25             return ResponseEntity.ok(response);
26         } catch (Exception e) {
27             log.error("Failed to start partitioned export job", e);
28             return ResponseEntity.internalServerError()
29                 .body(Map.of("status", "ERROR", "message", e.getMessage()));
30         }
31     }
}
```

```
32     ...
33     // ⏲ Scheduling automatique toutes les 2 heures
34     @Scheduled(cron = "0 */2 * * *") // ⚒ Cron :
35     public void scheduledExport() {
36         log.info("⌚ Démarrage de l'export automatique");
37
38         try {
39             JobParameters jobParameters = new JobParametersBuilder()
40                 .addLong("scheduledAt", System.currentTimeMillis())
41                 .addString("trigger", "scheduled")
42                 .toJobParameters();
43
44             jobLauncher.run(partitionedExportJob, jobParameters);
45             log.info("✅ Export automatique démarré avec succès");
46         } catch (Exception e) {
47             log.error("❌ Erreur lors de l'export automatique : " + e.getMessage());
48         }
49     }
50 }
```

💡 **Explication du cron :** "0 */2 * * *"

- 0 : à la minute 0
- */2 : toutes les 2 heures
- * * * * : tous les jours, mois, années

Fonctionnalité avancée : Import des utilisateurs



UserCsvDTO - DTO pour fichier CSV

```
1  @Data
2  @AllArgsConstructor
3  @NoArgsConstructor
4  public class UserCsvDTO {
5      private String email;
6      private String firstName;
7      private String lastName;
8      private String password;
9      private String birthDate;          // Format: "dd/MM/yyyy" ou "yyyy-MM-dd"
10     private String registrationDate; // Format: "dd/MM/yyyy HH:mm:ss"
11     private String role;            // "ADMIN", "USER", "ORGANIZER"
12 }
```

UserItemReader - Lecture CSV avec mapping personnalisé

```
1  @Component
2  public class UserItemReader implements ItemReader<UserCsvDTO> {
3
4      private final Resource resource;
5      private CSVReader csvReader;
6      private Iterator<String[]> csvIterator;
7      private boolean initialized = false;
8
9      public UserItemReader() {
10          this.resource = new ClassPathResource("data/users.csv");
11      }
12
13      @Override
14      public UserCsvDTO read() throws Exception {
15          if (!initialized) {
16              initializeReader();
17          }
18          if (csvIterator != null && csvIterator.hasNext()) {
19              String[] record = csvIterator.next();
20
21              // 🔎 Validation basique
22              if (record.length < 7) {
23                  logger.warn("Record incomplet ignoré: {}", Arrays.toString(record));
24                  return read(); // 🔄 Récursion pour passer au suivant
25              }
26              // 🔄 Mapping manuel des colonnes CSV → DTO
27              UserCsvDTO user = new UserCsvDTO();
28              user.setEmail(record[0].trim());
29              user.setFirstName(record[1].trim());
30              user.setLastName(record[2].trim());
31              user.setPassword(record[3].trim());
32              user.setBirthDate(record[4].trim());
33              user.setRegistrationDate(record[5].trim());
34              user.setRole(record[6].trim().toUpperCase());
35
36              return user;
37          }
38          closeReader();
39          return null;
40      }
41
42      ...
43      private void initializeReader() throws Exception {
44          InputStreamReader isr = new InputStreamReader(resource.getInputStream(), StandardCharsets.UTF_8);
45          csvReader = new CSVReader(isr);
46
47          // 🚫 Skip header line
48          String[] header = csvReader.readNext();
49          logger.info("CSV Header: {}", Arrays.toString(header));
50
51          List<String[]> allRecords = csvReader.readAll();
52          csvIterator = allRecords.iterator();
53          initialized = true;
54
55      }
56  }
```

UserItemProcessor - Validation et transformation

```
1  @Component
2  @RequiredArgsConstructor
3  public class UserItemProcessor implements ItemProcessor<UserCsvDTO, UserEntity> {
4
5      private final PasswordEncoder passwordEncoder; // 🔒 Encodage sécurisé des mots de passe
6      // 📅 Formatters pour différents formats de dates
7      private final DateTimeFormatter[] DATE_FORMATTERS = {
8          DateTimeFormatter.ofPattern("dd/MM/yyyy"),
9          DateTimeFormatter.ofPattern("yyyy-MM-dd"),
10         DateTimeFormatter.ofPattern("dd-MM-yyyy")
11     };
12     private final DateTimeFormatter[] DATETIME_FORMATTERS = {
13         DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss"),
14         DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"),
15         DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm")
16     };
17     @Override
18     public UserEntity process(UserCsvDTO item) throws Exception {
19         // 🔎 Validation des données obligatoires
20         if (!isValidUser(item)) {
21             logger.warn("✖ Utilisateur invalide ignoré: {}", item.getEmail());
22             return null; // Skip cet utilisateur
23         }
24         UserEntity user = new UserEntity();
25         // 📩 Email (unique et obligatoire)
26         user.setEmail(item.getEmail().toLowerCase().trim());
27         // 🧑 Noms
28         user.setFirstName(capitalizeFirstLetter(item.getFirstName()));
29         user.setLastName(capitalizeFirstLetter(item.getLastName()));
30         // 🔒 Mot de passe encodé (BCrypt)
31         user.setPassword(passwordEncoder.encode(item.getPassword()));
32         // 🕰️ Date de naissance (parsing flexible)
33         user.setBirthDate(parseDate(item.getBirthDate()));
34         // 📅 Date d'inscription (parsing flexible)
35         user.setRegistrationDate(parseDateTime(item.getRegistrationDate()));
36         // 🚶 Rôle avec validation
37         user.setRole(parseRole(item.getRole()));
38         logger.debug("✓ Processed user: {}", user.getEmail());
39         return user;
40     }
```

```
41    ...
42    private boolean isValidUser(UserCsvDTO user) {
43        return user.getEmail() != null && !user.getEmail().trim().isEmpty()
44            && user.getEmail().contains("@")
45            && user.getFirstName() != null && !user.getFirstName().trim().isEmpty()
46            && user.getPassword() != null && user.getPassword().length() >= 6;
47    }
48
49    private LocalDate parseDate(String dateStr) {
50        if (dateStr == null || dateStr.trim().isEmpty()) return null;
51
52        for (DateTimeFormatter formatter : DATE_FORMATTERS) {
53            try {
54                return LocalDate.parse(dateStr.trim(), formatter);
55            } catch (DateTimeParseException e) {
56                // ⚠ Try next formatter
57            }
58        }
59
60        logger.warn("⚠ Cannot parse date: {}", dateStr);
61        return null;
62    }
63
64    private LocalDateTime parseDateTime(String dateTimeStr) {
65        if (dateTimeStr == null || dateTimeStr.trim().isEmpty()) {
66            return LocalDateTime.now(); // 🕒 Par défaut : maintenant
67        }
68
69        for (DateTimeFormatter formatter : DATETIME_FORMATTERS) {
70            try {
71                return LocalDateTime.parse(dateTimeStr.trim(), formatter);
72            } catch (DateTimeParseException e) {
73                // ⚠ Try next formatter
74            }
75        }
76
77        logger.warn("⚠ Cannot parse datetime: {}, using current time", dateTimeStr);
78        return LocalDateTime.now();
79    }
80}
```

Configuration multi-stratégies pour l'import users

BatchUserConfig - 3 stratégies différentes

```
1  @Configuration
2  @RequiredArgsConstructor
3  @Slf4j
4  public class BatchUserConfig {
5
6      private final UserItemReader userReader;
7      private final UserItemProcessor userProcessor;
8      private final UserItemWriter userWriter;
9
10     // 📈 STRATÉGIE 1 : Import simple mono-thread
11     @Bean
12     public Step importUserStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
13         return new StepBuilder("importUserStep", jobRepository)
14             .<UserCsvDTO, UserEntity>chunk(50, transactionManager) // 50 users par chunk
15             .reader(userReader)
16             .processor(userProcessor)
17             .writer(userWriter)
18             .build();
19     }
20
21     @Bean
22     public Job importUserJob(JobRepository jobRepository, Step importUserStep) {
23         return new JobBuilder("importUserJob", jobRepository)
24             .start(importUserStep)
25             .build();
26     }
27
28     // 🔒 STRATÉGIE 2 : Import thread-safe pour multi-threading
29     @Bean
30     public Step synchronizedUserStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
31         return new StepBuilder("synchronizedUserStep", jobRepository)
32             .<UserCsvDTO, UserEntity>chunk(25, transactionManager) // Chunks plus petits pour multi-thread
33             .reader(synchronizedUserReader()) // 🔒 Version thread-safe
34             .processor(userProcessor)
35             .writer(synchronizedUserWriter()) // 🔒 Version thread-safe
36             .taskExecutor(userTaskExecutor()) // 💥 Multi-threading
37             .build();
38     }
```

```

50 ...
51     @Bean
52     public Job synchronizedImportUserJob(JobRepository jobRepository, Step synchronizedUserStep) {
53         return new JobBuilder("synchronizedImportUserJob", jobRepository)
54             .start(synchronizedUserStep)
55             .build();
56     }
57
58     // 🌟 STRATÉGIE 3 : Import avec partitioning
59     @Bean
60     public Step partitionedUserStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
61         return new StepBuilder("partitionedUserStep", jobRepository)
62             .partitioner("userStep", userPartitioner()) // 🌟 Découpage par domaine email
63             .step(importUserStep(jobRepository, transactionManager))
64             .gridSize(4) // 4 partitions en parallèle
65             .taskExecutor(userTaskExecutor())
66             .build();
67     }
68
69     // 🌟 Partitioner par domaine email (@gmail.com, @outlook.com, etc.)
70     @Bean
71     public Partitioner userPartitioner() {
72         return new UserItemPartitioner(); // Classe custom
73     }
74
75     @Bean("userTaskExecutor")
76     public TaskExecutor userTaskExecutor() {
77         ThreadPoolTaskExecutor executor = new ThreadPoolTaskExecutor();
78         executor.setCorePoolSize(2);
79         executor.setMaxPoolSize(4);
80         executor.setQueueCapacity(50);
81         executor.setThreadNamePrefix("user-import-");
82         executor.initialize();
83         return executor;
84     }
85 }

```

Configuration Base de données : PostgreSQL vs H2



Application.yaml - Configuration multi-profil

```
1  spring:
2    application:
3      name: events-batch
4
5    # 🔍 Profil actif (dev = H2, prod = PostgreSQL)
6    profiles:
7      active: dev
8
9    # 📁 Configuration JPA commune
10   jpa:
11     hibernate:
12       ddl-auto: update # Crée/met à jour les tables automatiquement
13       show-sql: true # Affiche les requêtes SQL dans les logs
14       properties:
15         hibernate:
16           format_sql: true # Format SQL lisible
17           use_sql_comments: true
18
19   # 📊 Configuration Spring Batch
20   batch:
21     job:
22       enabled: false # 🔑 Désactive le démarrage automatique des jobs
23     jdbc:
24       initialize-schema: always # Crée les tables de métadonnées Spring Batch
```



PROFIL DÉVELOPPEMENT (H2 en mémoire).

```
1  spring:
2    config:
3      activate:
4        on-profile: dev
5
6    datasource:
7      url: jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
8      driver-class-name: org.h2.Driver
9      username: sa
10     password: password
11
12   h2:
13     console:
14       enabled: true # 🌐 Interface web H2 sur http://localhost:8080/h2-console
15       path: /h2-console
```



PROFIL PRODUCTION (PostgreSQL).

```
1  spring:
2    config:
3      activate:
4        on-profile: prod
5
6    datasource:
7      url: jdbc:postgresql://localhost:5432/events_batch_db
8      driver-class-name: org.postgresql.Driver
9      username: ${DB_USERNAME:events_user} # 🔒 Variable d'environnement
10     password: ${DB_PASSWORD:events_password} # 🔒 Variable d'environnement
11
12    # 🛠 Configuration du pool de connexions
13    hikari:
14      maximum-pool-size: 20
15      minimum-idle: 5
16      connection-timeout: 30000
17      idle-timeout: 600000
18      max-lifetime: 1800000
19
20    logging:
21      level:
22        com.formation.events: DEBUG
23        org.springframework.batch: INFO
24        org.springframework.security: WARN
```



DevSecurityConfig - Sécurité pour développement

```
1  @Configuration
2  @Profile("dev") // 🔑 Actif uniquement en profil développement
3  @EnableWebSecurity
4  public class DevSecurityConfig {
5
6      @Bean
7      public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
8          return http
9              .csrf(csrf -> csrf.disable()) // 🔒 Désactive CSRF pour les tests
10             .authorizeHttpRequests(auth -> auth
11                 .requestMatchers("/api/**").permitAll() // 🌐 API libre
12                 .requestMatchers("/h2-console/**").permitAll() // 📁 Console H2 libre
13                 .anyRequest().authenticated()
14             )
15             .headers(headers -> headers
16                 .frameOptions().disable() // 🚫 Autorise H2 console dans iframe
17             )
18             .build();
19     }
20
21     @Bean
22     public PasswordEncoder passwordEncoder() {
23         return new BCryptPasswordEncoder(12); // 🔒 BCrypt avec force 12
24     }
25 }
```

Service avancé : UserService avec cache

UserService - Gestion centralisée des utilisateurs

```
1  @Service
2  @RequiredArgsConstructor
3  @Slf4j
4  public class UserService {
5
6      private final UserRepository userRepository;
7
8      // ⚡ Cache concurrent pour éviter les doublons lors du multi-threading
9      private final ConcurrentHashMap<String, UserEntity> userCache = new ConcurrentHashMap<>();
10
11     // 🚧 Utilisateur système par défaut pour les événements sans organisateur
12     @PostConstruct
13     public void initializeSystemUser() {
14         findOrCreateUser("system@events.com", "System", "User");
15         log.info("✅ System user initialized");
16     }
}
```

```

17 ...
18 /**
19 * 🔎 Trouve ou crée un utilisateur (thread-safe)
20 * Utilisé par OpenAgendaEventProcessor pour créer automatiquement les organisateurs
21 */
22 public UserEntity findOrCreateUser(String email, String firstName, String lastName) {
23     if (email == null || email.trim().isEmpty()) {
24         return getSystemUser();
25     }
26
27     String cleanEmail = email.toLowerCase().trim();
28
29     // ⚡ Vérifie d'abord le cache (évite les requêtes BDD)
30     UserEntity cachedUser = userCache.get(cleanEmail);
31     if (cachedUser != null) {
32         return cachedUser;
33     }
34
35     // 🔒 Synchronisation pour éviter les doublons lors du multi-threading
36     return userCache.computeIfAbsent(cleanEmail, key -> {
37         // 🔎 Cherche en BDD
38         Optional<UserEntity> existingUser = userRepository.findByEmail(key);
39         if (existingUser.isPresent()) {
40             log.debug("👤 User found in DB: {}", key);
41             return existingUser.get();
42         }
43
44         // 📄 Crée un nouvel utilisateur
45         UserEntity newUser = new UserEntity();
46         newUser.setEmail(key);
47         newUser.setFirstName(firstName != null ? firstName : "Unknown");
48         newUser.setLastName(lastName != null ? lastName : "User");
49         newUser.setPassword("$2a$12$defaultHashedPassword"); // 🔒 Mot de passe par défaut hashé
50         newUser.setRole("USER");
51         newUser.setRegistrationDate(LocalDateTime.now());
52
53         try {
54             UserEntity savedUser = userRepository.save(newUser);
55             log.info("✅ New user created: {}", key);
56             return savedUser;
57         } catch (Exception e) {
58             log.error("✖ Failed to create user: {}", key, e);
59             return getSystemUser(); // Fallback vers utilisateur système
60         }
61     });
62 }

```

```
63 ...
64     /**
65      * 🔍 Utilisateur système par défaut
66      */
67     public UserEntity getSystemUser() {
68         return userCache.computeIfAbsent("system@events.com", key -> {
69             return userRepository.findByEmail(key)
70                 .orElseThrow(() -> new RuntimeException("System user not found!"));
71         });
72     }
73
74     /**
75      * 📈 Statistiques du cache (pour debugging)
76      */
77     public Map<String, Object> getCacheStats() {
78         Map<String, Object> stats = new HashMap<>();
79         stats.put("cacheSize", userCache.size());
80         stats.put("cachedEmails", new ArrayList<>(userCache.keySet()));
81         return stats;
82     }
83
84     /**
85      * ✎ Nettoie le cache (utile lors des tests)
86      */
87     public void clearCache() {
88         userCache.clear();
89         log.info("✎ User cache cleared");
90     }
91 }
```

@StepScope et injection de paramètres dynamiques

Principe de @StepScope

`@StepScope` permet d'injecter des **paramètres dynamiques** dans les composants Spring Batch pendant l'exécution du job.

Utilisation avec @Value

```
1  @Component
2  @StepScope // ⚡ OBLIGATOIRE pour l'injection dynamique !
3  public class ParameterizedItemReader implements ItemReader<MyDTO> {
4
5      // 💾 Injection de paramètres depuis ExecutionContext
6      @Value("#{stepExecutionContext['fileName']}") // ⚡ Depuis le contexte du step
7      private String fileName;
8
9      @Value("#{jobParameters['batchDate']}")           // ⚡ Depuis les paramètres du job
10     private String batchDate;
11
12     @Value("#{stepExecutionContext['startId']}")    // ⚡ Pour partitioning
13     private Long startId;
14
15     @Value("#{stepExecutionContext['endId']}")
16     private Long endId;
17
18     @Override
19     public MyDTO read() throws Exception {
20         // 🔎 Utilise les paramètres injectés dynamiquement
21         log.info("Reading from file: {} for date: {} (IDs {}-{})",
22                 fileName, batchDate, startId, endId);
23
24         // ... logique de lecture avec les paramètres
25         return null;
26     }
27 }
```

Exemple avec Partitioner

```
1  @Bean
2  public Partitioner filePartitioner() {
3      return (gridSize) -> {
4          Map<String, ExecutionContext> partitions = new HashMap<>();
5
6          // 📁 Crée une partition par fichier
7          String[] files = {"file1.csv", "file2.csv", "file3.csv"};
8
9          for (int i = 0; i < files.length; i++) {
10              ExecutionContext context = new ExecutionContext();
11
12              // 🛡️ Met les paramètres dans le contexte
13              context.putString("fileName", files[i]);
14              context.putLong("startId", i * 1000L);
15              context.putLong("endId", (i + 1) * 1000L - 1);
16
17              partitions.put("partition_" + i, context);
18          }
19
20          return partitions;
21      };
22  }
23
24  @Bean
25  public Step partitionedStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
26      return new StepBuilder("partitionedStep", jobRepository)
27          .partitioner("fileStep", filePartitioner())
28          .step(fileStep(jobRepository, transactionManager))
29          .gridSize(3) // 3 partitions en parallèle
30          .taskExecutor(taskExecutor())
31          .build();
32  }
```

Gestion avancée des erreurs avec Skip/Retry.

X Types d'erreurs dans Spring Batch

Retry - Réessayer

Pour les **erreurs temporaires** :

- Problème réseau
- Timeout base de données
- Verrou temporaire

Fail - Arrêter

Pour les **erreurs critiques** :

- Problème de configuration
- Fichier source introuvable
- Erreur de connexion BDD

Skip - Ignorer

Pour les **erreurs définitives** :

- Données invalides
- Contrainte de validation
- Format incorrect

Configuration Fault Tolerance

```
1  @Bean
2  public Step faultTolerantStep(JobRepository jobRepository, PlatformTransactionManager transactionManager) {
3      return new StepBuilder("faultTolerantStep", jobRepository)
4          .<UserCsvDTO, UserEntity>chunk(100, transactionManager)
5          .reader(userReader)
6          .processor(userProcessor)
7          .writer(userWriter)
8
9      // 🔧 Configuration de tolérance aux erreurs
10     .faultTolerant()
11
12     // ➡ Skip Configuration
13     .skipLimit(50) // Maximum 50 erreurs ignorées
14     .skip(ValidationException.class) // Ignore les erreurs de validation
15     .skip(DataIntegrityViolationException.class) // Ignore les doublons
16     .noSkip(FileNotFoundException.class) // ❌ NE PAS ignorer les fichiers manquants
17
18     // ⚙ Retry Configuration
19     .retryLimit(3) // Maximum 3 tentatives
20     .retry(TransientDataAccessException.class) // Retry les erreurs DB temporaires
21     .retry(ConnectException.class) // Retry les erreurs de connexion
22     .noRetry(ValidationException.class) // ❌ Ne pas retry les erreurs de validation
23
24     // 📈 Listeners pour logging
25     .listener(new RetryListener() {
26         @Override
27         public <T, E extends Throwable> void onError(RetryContext context,
28                                         RetryCallback<T, E> callback,
29                                         Throwable throwable) {
30             log.warn("⌚ Retry attempt {} for: {}", context.getRetryCount(), throwable.getMessage());
31         }
32     })
33 }
```

```
34     ...
35
36     .listener(new SkipListener<UserCsvDTO, UserEntity>() {
37         @Override
38         public void onSkipInRead(Throwable t) {
39             log.warn("➡ Skipped during READ: {}, item.getEmail(), t.getMessage());
40         }
41
42         @Override
43         public void onSkipInProcess(UserCsvDTO item) {
44             log.warn("➡ Skipped during PROCESS: {}, item.getEmail(), t.getMessage());
45         }
46
47         @Override
48         public void onSkipInWrite(UserEntity item) {
49             log.warn("➡ Skipped during WRITE: {}, item.getEmail(), t.getMessage());
50         }
51
52     })
53
54     .build();
55 }
56 }
```

Architecture complète avec tous les contrôleurs

Structure des APIs REST

```
1  /api/batch/          # BatchController – APIs principales
2  └── /import-events   # Import JSON mono-thread
3  └── /import-events-fast # Import JSON multi-thread
4  └── /job-history/{jobName} # Historique des exécutions
5  └── /job-status/{jobId}  # Statut d'un job spécifique
6
7  /api/batch/users/    # BatchController – Gestion users
8  └── /import-simple    # Import CSV simple
9  └── /import-synchronized # Import CSV thread-safe
10 └── /import-partitioned # Import CSV avec partitioning
11 └── /cache-stats      # Statistiques du UserService cache
12
13 /api/batch/events/   # BatchEventController – Export + Scheduling
14 └── /export-simple    # Export CSV simple
15 └── /export-partitioned # Export CSV par année
16 └── /start-scheduling  # Active le scheduling automatique
17 └── /stop-scheduling   # Désactive le scheduling
```



ResponseController - Utilitaire pour réponses standardisées

```
1  @Component
2  @Slf4j
3  public class ResponseController {
4
5      /**
6       * ⚡ Crée une réponse standardisée pour les jobs
7       */
8      public ResponseEntity<Map<String, String>> createJobResponse(
9          String jobName, JobExecution execution) {
10
11         Map<String, String> response = new HashMap<>();
12
13         if (execution != null) {
14             response.put("status", "SUCCESS");
15             response.put("jobId", execution.get jobId().toString());
16             response.put("jobName", jobName);
17             response.put("startTime", execution.getStartTime().toString());
18             response.put("message", "Job started successfully");
19
20             log.info("✅ Job {} started with ID: {}", jobName, execution.get jobId());
21             return ResponseEntity.ok(response);
22         } else {
23             response.put("status", "ERROR");
24             response.put("message", "Failed to start job: " + jobName);
25
26             log.error("❌ Failed to start job: {}", jobName);
27             return ResponseEntity.internalServerError().body(response);
28         }
29     }
```

```
30    ...
31    /**
32     * 📈 Crée une réponse avec statistiques d'exécution
33     */
34    public ResponseEntity<Map<String, Object>> createJobHistoryResponse(
35        List<JobExecution> executions) {
36        Map<String, Object> response = new HashMap<>();
37        List<Map<String, Object>> history = executions.stream()
38            .map(execution -> {
39                Map<String, Object> jobInfo = new LinkedHashMap<>();
40                jobInfo.put("jobId", execution.get jobId());
41                jobInfo.put("status", execution.getStatus().toString());
42                jobInfo.put("startTime", execution.getStartTime());
43                jobInfo.put("endTime", execution.getEndTime());
44                // 📈 Calcul des métriques
45                long readCount = execution.getStepExecutions().stream()
46                    .mapToLong(StepExecution::getReadCount).sum();
47                long writeCount = execution.getStepExecutions().stream()
48                    .mapToLong(StepExecution::getWriteCount).sum();
49                long skipCount = execution.getStepExecutions().stream()
50                    .mapToLong(StepExecution::getSkipCount).sum();
51                jobInfo.put("readCount", readCount);
52                jobInfo.put("writeCount", writeCount);
53                jobInfo.put("skipCount", skipCount);
54                // ⏳ Calcul de la durée
55                if (execution.getStartTime() != null && execution.getEndTime() != null) {
56                    long duration = execution.getEndTime().getTime() - execution.getStartTime().getTime();
57                    jobInfo.put("durationMs", duration);
58                    jobInfo.put("durationFormatted", formatDuration(duration));
59                }
60                return jobInfo;
61            })
62            .collect(Collectors.toList());
63        response.put("totalExecutions", history.size());
64        response.put("executions", history);
65        return ResponseEntity.ok(response);
66    }
```

```
67 ...
68     private String formatDuration(long durationMs) {
69         long seconds = durationMs / 1000;
70         long minutes = seconds / 60;
71         long hours = minutes / 60;
72
73         if (hours > 0) {
74             return String.format("%dh %02dm %02ds", hours, minutes % 60, seconds % 60);
75         } else if (minutes > 0) {
76             return String.format("%dm %02ds", minutes, seconds % 60);
77         } else {
78             return String.format("%ds", seconds);
79         }
80     }
81 }
```

Application principale avec @EnableScheduling

EventsBatchApplication - Configuration complète

```
1  @SpringBootApplication
2  @EnableJpaAuditing // 🔑 Active l'audit automatique (CreatedDate, LastModifiedDate)
3  @EnableScheduling // 🔑 Active le scheduling avec @Scheduled
4  @Slf4j
5  public class EventsBatchApplication {
6
7      public static void main(String[] args) {
8          // 🚀 Démarrage de l'application Spring Boot
9          ConfigurableApplicationContext context = SpringApplication.run(EventsBatchApplication.class, args);
10
11         // 📊 Affichage des informations de démarrage
12         logApplicationInfo(context);
13     }
14
15     /**
16      * 📋 Affiche les informations utiles au démarrage
17     */
18     private static void logApplicationInfo(ConfigurableApplicationContext context) {
19         Environment env = context.getEnvironment();
20         String profile = String.join(", ", env.getActiveProfiles());
21         String port = env.getProperty("server.port", "8080");
22
23         log.info("🚀 ======");
24         log.info("🚀 Application Events Batch démarrée avec succès !");
25         log.info("🚀 ======");
26         log.info("🔧 Profil actif: {}", profile.isEmpty() ? "default" : profile);
27         log.info("🌐 URL: http://localhost:{}", port);
28         log.info("💻 Console H2: http://localhost:{}/h2-console", port);
29         log.info("📊 API Batch: http://localhost:{}/api/batch", port);
30         log.info("🚀 ======");
31
32         // 📊 Affichage des jobs disponibles
33         Map<String, Job> jobs = context.getBeansOfType(Job.class);
34         log.info("📋 Jobs disponibles ({}): {}", jobs.size());
35         jobs.keySet().forEach(jobName -> log.info("    - {}", jobName));
36
37         log.info("🚀 ======");
38     }
39 }
```

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. Processor : `OpenAgendaEventProcessor` avec `UserService`
6. Writer : `EventItemWriter` simple



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. Processor : `OpenAgendaEventProcessor` avec `UserService`
6. Writer : `EventItemWriter` simple
7. Configuration : `BatchConfig` mono-thread



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`
6. **Writer** : `EventItemWriter` simple
7. **Configuration** : `BatchConfig` mono-thread
8. **API** : `BatchController` de base



Phase 2 : Fonctionnalités avancées

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`
6. **Writer** : `EventItemWriter` simple
7. **Configuration** : `BatchConfig` mono-thread
8. **API** : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. **Export CSV** : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`
6. **Writer** : `EventItemWriter` simple
7. **Configuration** : `BatchConfig` mono-thread
8. **API** : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. **Export CSV** : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. **Partitioning** : `EventCSVBatchConfig` avec partitioner

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`
6. **Writer** : `EventItemWriter` simple
7. **Configuration** : `BatchConfig` mono-thread
8. **API** : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. **Export CSV** : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. **Partitioning** : `EventCSVBatchConfig` avec partitioner
11. **Import Users** : `UserItemReader` + `UserItemProcessor` + `UserItemWriter`

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. Processor : `OpenAgendaEventProcessor` avec `UserService`
6. Writer : `EventItemWriter` simple
7. Configuration : `BatchConfig` mono-thread
8. API : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. Export CSV : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. Partitioning : `EventCSVBatchConfig` avec partitioner
11. Import Users : `UserItemReader` + `UserItemProcessor` + `UserItemWriter`
12. Multi-configs : `BatchUserConfig` avec 3 stratégies

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. Processor : `OpenAgendaEventProcessor` avec `UserService`
6. Writer : `EventItemWriter` simple
7. Configuration : `BatchConfig` mono-thread
8. API : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. Export CSV : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. Partitioning : `EventCSVBatchConfig` avec partitioner
11. Import Users : `UserItemReader` + `UserItemProcessor` + `UserItemWriter`
12. Multi-configs : `BatchUserConfig` avec 3 stratégies
13. Scheduling : `BatchEventController` + `@Scheduled` + cron

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. Configuration : `pom.xml` + `application.yaml` + profils
2. Entités : `SuperClass` + `EventEntity` + `UserEntity`
3. DTOs : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. Readers : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. Processor : `OpenAgendaEventProcessor` avec `UserService`
6. Writer : `EventItemWriter` simple
7. Configuration : `BatchConfig` mono-thread
8. API : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. Export CSV : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. Partitioning : `EventCSVBatchConfig` avec partitioner
11. Import Users : `UserItemReader` + `UserItemProcessor` + `UserItemWriter`
12. Multi-configs : `BatchUserConfig` avec 3 stratégies
13. Scheduling : `BatchEventController` + `@Scheduled` + cron
14. Monitoring : `ResponseController` + historique détaillé

Ordre d'implémentation mis à jour



Phase 1 : Bases (Import événements)

1. **Configuration** : `pom.xml` + `application.yaml` + profils
2. **Entités** : `SuperClass` + `EventEntity` + `UserEntity`
3. **DTOs** : `OpenAgendaDTO` avec `@JsonIgnoreProperties`
4. **Readers** : `JsonItemReader` puis `ThreadSafeJsonItemReader`
5. **Processor** : `OpenAgendaEventProcessor` avec `UserService`
6. **Writer** : `EventItemWriter` simple
7. **Configuration** : `BatchConfig` mono-thread
8. **API** : `BatchController` de base



Phase 2 : Fonctionnalités avancées

9. **Export CSV** : `EventBDDItemReader` + `EventBDDItemWriter` + `@StepScope`
10. **Partitioning** : `EventCSVBatchConfig` avec partitioner
11. **Import Users** : `UserItemReader` + `UserItemProcessor` + `UserItemWriter`
12. **Multi-configs** : `BatchUserConfig` avec 3 stratégies
13. **Scheduling** : `BatchEventController` + `@Scheduled` + cron
14. **Monitoring** : `ResponseController` + historique détaillé
15. **Fault Tolerance** : Skip/Retry policies + listeners

Résumé des fonctionnalités complètes

Import de données

- **Événements JSON** (mono + multi-thread)
- **Utilisateurs CSV** (3 stratégies)
- **Validation** et transformation
- **Thread-safety** avec AtomicInteger
- **Gestion d'erreurs** skip/retry

Export de données

- **Export CSV** événements
- **Partitioning** par année
- **Headers/Footers** personnalisés
- **@StepScope** pour paramètres dynamiques

Architecture finale

7 **Jobs** configurés • 3 **Contrôleurs** REST • 15+ **Endpoints** • Multi-threading • Partitioning • Fault tolerance

Automatisation

- **Scheduling** avec cron expressions
- **APIs REST** complètes
- **Monitoring** temps réel
- **Historique** des exécutions

Données et sécurité

- **PostgreSQL + H2** par profils
- **UserService** avec cache concurrent
- **Audit** automatique avec JPA
- **Sécurité** configurée par environnement

Ressources et documentation



Documentation officielle

- [Spring Batch Reference](#)
- [Spring Boot Batch](#)
- [Baeldung Tutorials](#)



Bonnes pratiques

- [Documentation](#) des jobs
- [Tests automatisés](#)
- [Versioning](#) des configurations
- [Monitoring](#) proactif



Outils utiles

- [Spring Batch Admin](#) : Interface web
- [Micrometer](#) : Métriques
- [Grafana](#) : Visualisation
- [Zipkin](#) : Tracing distribué



Communauté

- [Stack Overflow](#)
- [GitHub Samples](#)
- [Spring Community](#)