Rapport de TP final

Système de Gestion d'Événements

FONGANG NDE BRYAN

Encadrant : Dr W. Kungne

Mai 2025

0.1 Résumé

e rapport présente la conception et l'implémentation d'un Système de Gestion d'Événements développé dans le cadre du Travail Pratique final de Programmation Orientée Objet. L'application, écrite en Java, permet de gérer divers types d'événements tels que des conférences et des concerts. Elle met en œuvre des concepts avancés de POO, incluant l'héritage, le polymorphisme, les patrons de conception (Observer, Singleton, Factory), la gestion personnalisée des exceptions, la persistance des données via sérialisation JSON manuelle, l'asynchronisme pour les notifications, une interface graphique avec JavaFX, et des tests unitaires avec JUnit. Le système permet l'ajout, la modification, la suppression d'événements, la gestion des participants et l'envoi de notifications dynamiques.

Mots-clés : Java, JavaFX, POO, Gestion d'Événements, JSON, Design Patterns, JUnit, Persistance.

Chapitre 1

Introduction

Le présent document constitue le rapport final du Travail Pratique (TP) de Programmation Orientée Objet (POO). L'objectif de ce TP était de concevoir et de développer une application Java robuste pour la gestion d'événements variés, tels que des conférences, des concerts, etc. Ce projet a été l'occasion de mettre en pratique des concepts fondamentaux et avancés de la POO, ainsi que des bonnes pratiques de développement logiciel.

L'application développée, nommée "Système de Gestion d'Événements", devait non seulement permettre les opérations CRUD (Create, Read, Update, Delete) sur les événements, mais aussi gérer les inscriptions des participants, envoyer des notifications, et assurer la persistance des données. Une attention particulière a été portée à l'architecture logicielle, à l'utilisation de patrons de conception pertinents, à la gestion des erreurs et à la testabilité du code.

Ce rapport détaillera les différentes phases du projet, depuis l'analyse des besoins jusqu'à l'implémentation et la validation, en passant par la conception détaillée du système. Il présentera les choix technologiques effectués, l'architecture retenue, les modèles UML, ainsi qu'un aperçu de l'interface utilisateur et des mécanismes de persistance et de test.

Chapitre 2

Analyse des Besoins et Étude de Faisabilité

L'énoncé du TP final de Programmation Orientée Objet nous a demandé de concevoir une application Java permettant de gérer différents types d'événements : conférences, concerts, etc. Cette application devait mettre en œuvre des concepts avancés tels que :

- L'héritage, le polymorphisme, les interfaces,
- Les patrons de conception (Observer, Singleton, Factory...),
- La gestion des exceptions personnalisées,
- La persistance via la sérialisation JSON,
- L'asynchronisme (envoi différé de notifications),
- Une interface graphique via JavaFX,
- Des tests unitaires avec JUnit.

L'application devait permettre l'ajout, la modification, la suppression d'événements, la gestion des participants, ainsi que l'envoi de notifications dynamiques en temps réel.

2.1 Analyse des parties prenantes

- **Organisateurs d'événements :** Personnes ou entités responsables de la création, de la gestion et de la supervision des événements.
- **Participants :** Individus s'inscrivant et assistant aux événements.
- **(Optionnel) Administrateur système :** Utilisateur avec des droits étendus pour la maintenance du système (non explicitement demandé mais implicite pour la gestion globale).

2.2 Etude des besoins utilisateurs

2.2.1 Pour les Organisateurs :

- Créer de nouveaux événements (conférences, concerts) avec des détails spécifiques (nom, date, lieu, capacité, thème/intervenants pour conférences, artiste/genre pour concerts).
- Modifier les détails d'événements existants.
- Supprimer des événements.
- Consulter la liste des événements et des participants inscrits à chaque événement.
- Recevoir des notifications (par exemple, lorsqu'un événement est plein).

2.2.2 Pour les Participants :

- Consulter la liste des événements disponibles.

— S'inscrire à un événement.
— Se désinscrire d'un événement.
— Recevoir des notifications concernant les événements auxquels ils sont inscrits (par exemple, rappel, annulation, modification).

2.3 Etude de faisabilité

2.3.1 Faisabilité technique :

— Le langage Java et l'écosystème Java (JDK 11+) sont matures et bien adaptés pour ce type de projet.
— JavaFX est une technologie éprouvée pour la création d'interfaces graphiques desktop.
— La sérialisation JSON manuelle est réalisable avec les classes I/O standard de Java.
— JUnit est l'outil standard pour les tests unitaires en Java.
— Les concepts POO et les design patterns sont intégrables dans l'architecture.

La faisabilité technique est jugée élevée.

2.3.2 Faisabilité économique :

— Les outils de développement (JDK, IntelliJ IDEA Community Edition, Git) sont gratuits.
— Le principal "coût" est le temps de développement, ce qui est l'objectif d'un TP.

La faisabilité économique est totale dans le cadre d'un projet académique.

2.3.3 Faisabilité organisationnelle :

— Le projet est développé par un seul étudiant, ce qui simplifie la coordination.
— Un encadrant est disponible pour des conseils et un suivi.

La faisabilité organisationnelle est assurée.

Chapitre 3

Conception du Système

3.1 Architecture globale du système

Le projet est structuré selon une architecture s'inspirant du modèle MVC (Modèle-Vue-Contrôleur), répartie en plusieurs packages pour une meilleure organisation du code :

- **`model` :** Contient les entités métier (classes `Evenement`, `Conference`, `Concert`, `Participant`, `Organisateur`) et la logique de gestion des données (par exemple, `GestionEvenements`). Il représente l'état et le comportement de l'application.
- **`controller` :** Contient les classes responsables de la logique de contrôle, faisant le lien entre l'interface utilisateur (la Vue) et le Modèle. Elles traitent les actions de l'utilisateur et mettent à jour la vue en conséquence.
- **`view` (implicite via JavaFX FXML et classes UI) :** Représente l'interface utilisateur graphique (IUG) avec laquelle l'utilisateur interagit. Les fichiers FXML décrivent la structure des vues, et les classes de contrôleur JavaFX gèrent leur comportement.
- **`utils` :** Contient des classes utilitaires pour des fonctionnalités transversales telles que la persistance des données (`PersistenceService`), la gestion des notifications (`NotificationService`, `EmailNotificationService`), et les exceptions personnalisées (`CapaciteMaxAtteinteException`, `EvenementDejaExistantExceptio`

Le patron de conception **Singleton** est utilisé pour `GestionEvenements` afin d'assurer une instance unique de gestionnaire central des événements. Le patron **Observer** est utilisé pour notifier les participants des changements relatifs aux événements. Un patron **Factory** peut être envisagé pour la création d'objets `Evenement` de types différents.

3.2 Modélisation UML

3.2.1 Modèle de classe métier

Le diagramme UML de classe métier suivant illustre les principales classes, leurs relations ainsi que les interactions principales du système. Ce diagramme modélise notamment :

- Une classe abstraite `Evenement`, héritée par `Concert` et `Conference`.
- Un `Participant` qui peut être observateur d'un événement.
- Un `Organisateur`.
- Un gestionnaire global `GestionEvenements` (Singleton).
- Le service de notification `NotificationService` (interface).

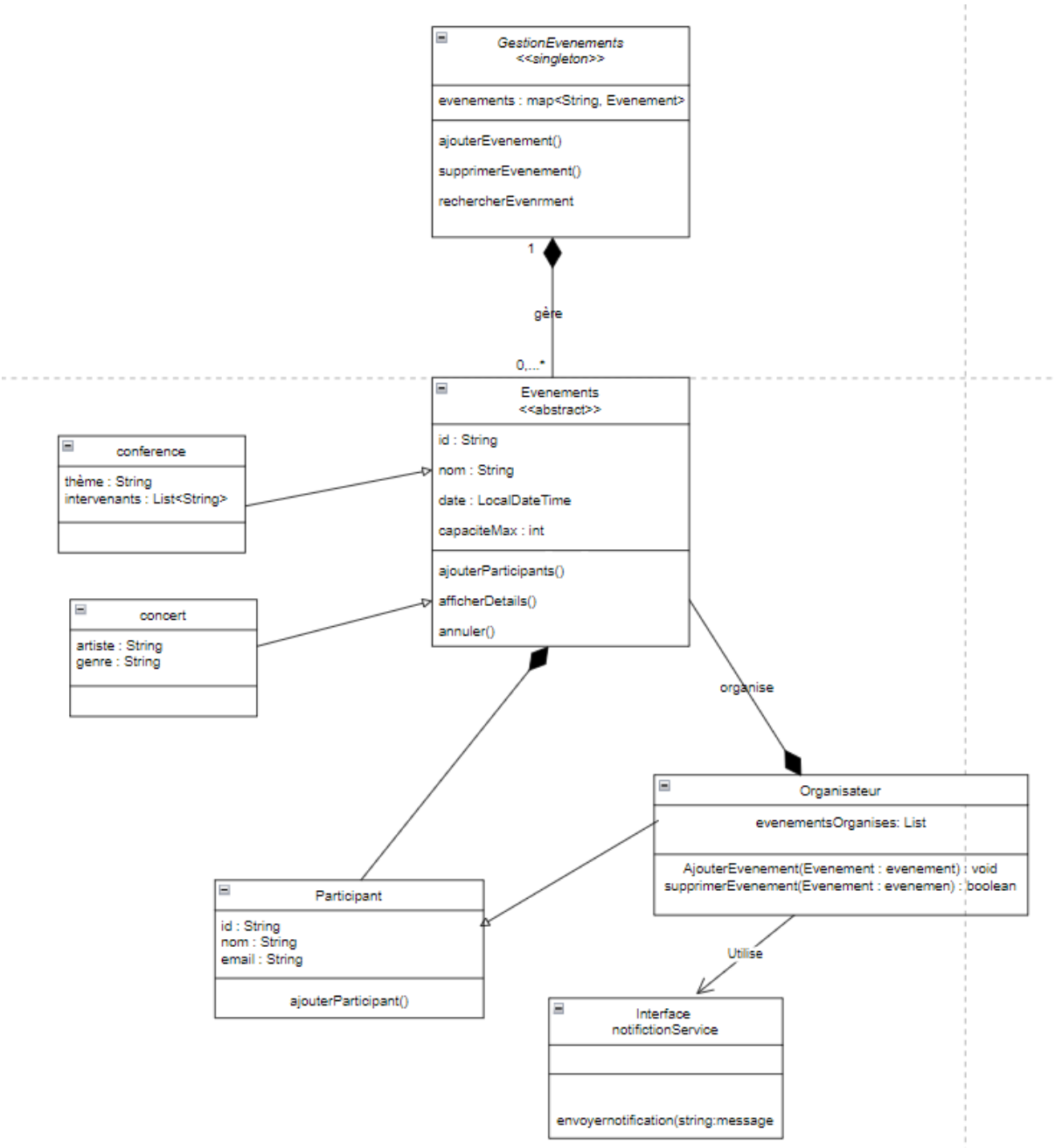

FIGURE 3.1 – Diagramme de Classe Métier.

3.3 Conception du stockage des données (Persistance JSON)

La persistance des données est assurée par sérialisation et désérialisation d'objets `Evenement` au format JSON. Cette opération est gérée manuellement par la classe `PersistenceService`.

Structure du fichier JSON (`events.json`) :

```
[
  {
    "type": "Conference",
    "id": "conf001",
    "nom": "Java Avance",
    "date": "2025-10-20T10:00:00",
    "lieu": "Amphi A",
    "capaciteMax": 100,
    "participants": [
      {"id": "p001", "nom": "Alice Dupont", "email": "alice@example.com"}
    ],
    "theme": "POO et Design Patterns",
    "intervenants": ["Dr. W. Kungne", "Prof. X. Yz"]
  }
  % ... autres   vnements   ...
]
```

Listing 3.1 – Exemple de structure du fichier events.json

Classe `PersistenceService` : Responsable de `sauvegarderJSON` et `chargerJSON`.

Chapitre 4

Implémentation et Développement

4.1 Environnement de développement

— **Langage :** Java 11
— **IDE :** IntelliJ IDEA
— **Framework UI :** JavaFX
— **Tests :** JUnit 5

4.2 Structure du projet

(Voir Chapitre 3.2 pour la liste des packages et classes)

```
org.example.tpfinal
          controller
                ...
          model
                ...
          utils
                ...
          EventManagementApp.java
```

Listing 4.1 – Structure arborescente du projet (rappel)

4.3 Interfaces utilisateur (Front-End)

L'interface graphique JavaFX comprend plusieurs vues pour les différentes fonctionnalités.

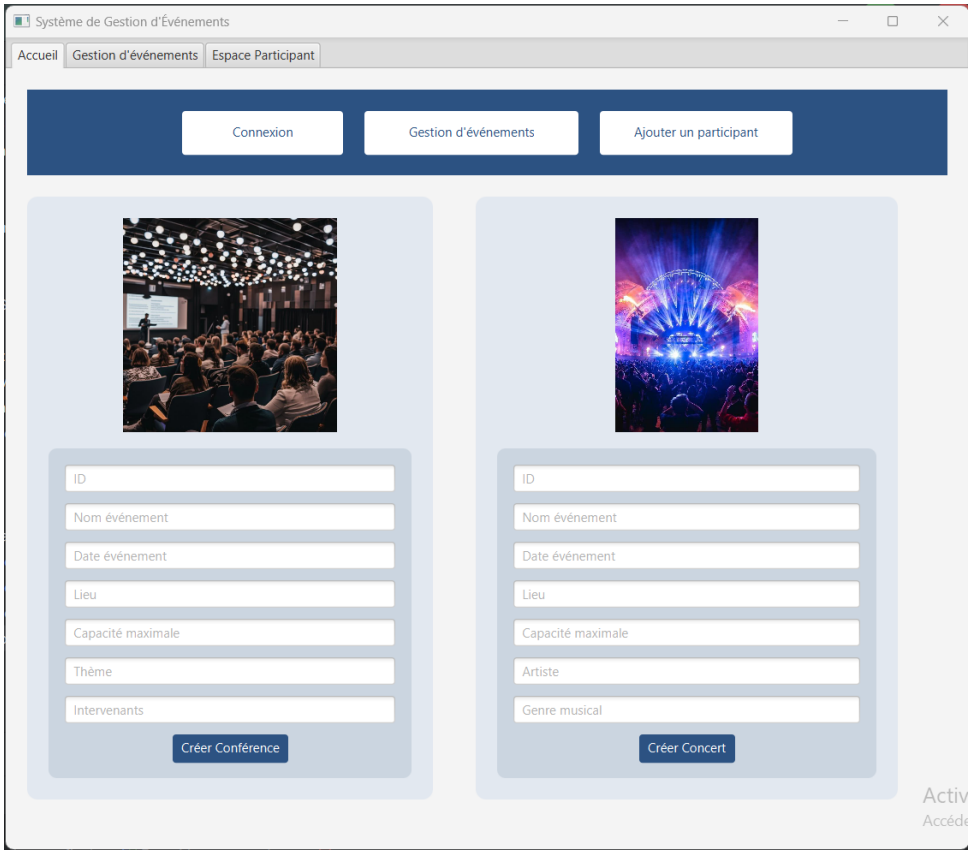

FIGURE 4.1 – Interface utilisateur - Vue principale / Accueil.

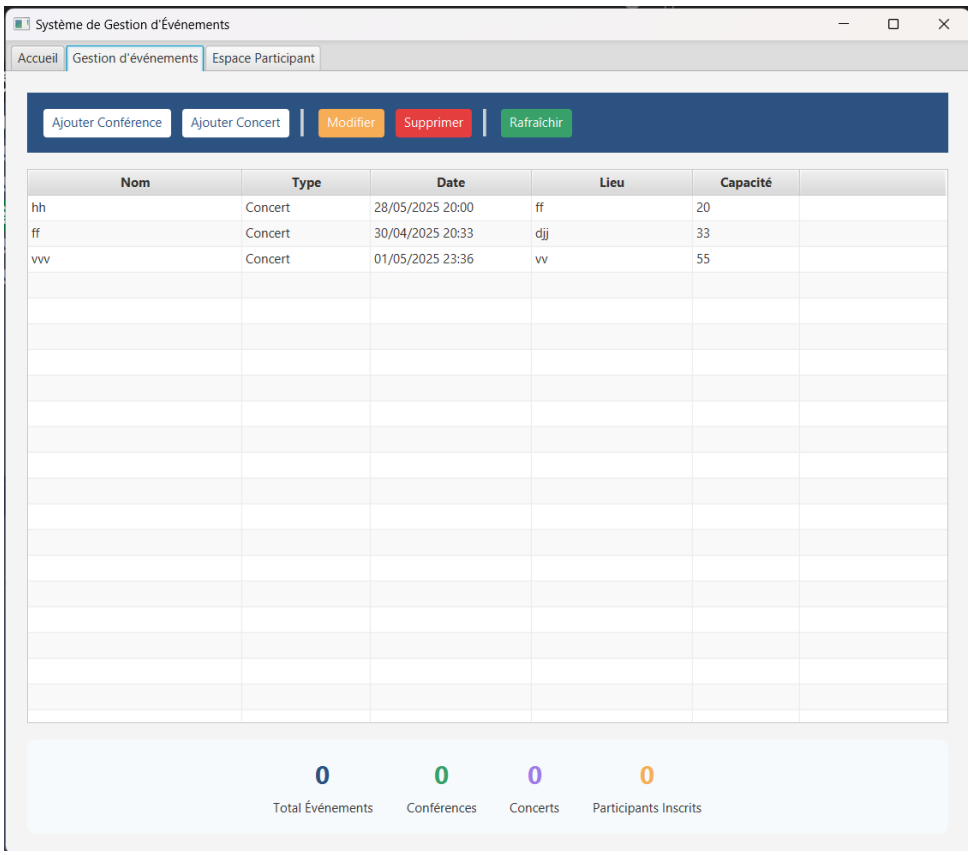

FIGURE 4.2 – Interface utilisateur - Vue de gestion des événements (ajout/modification).

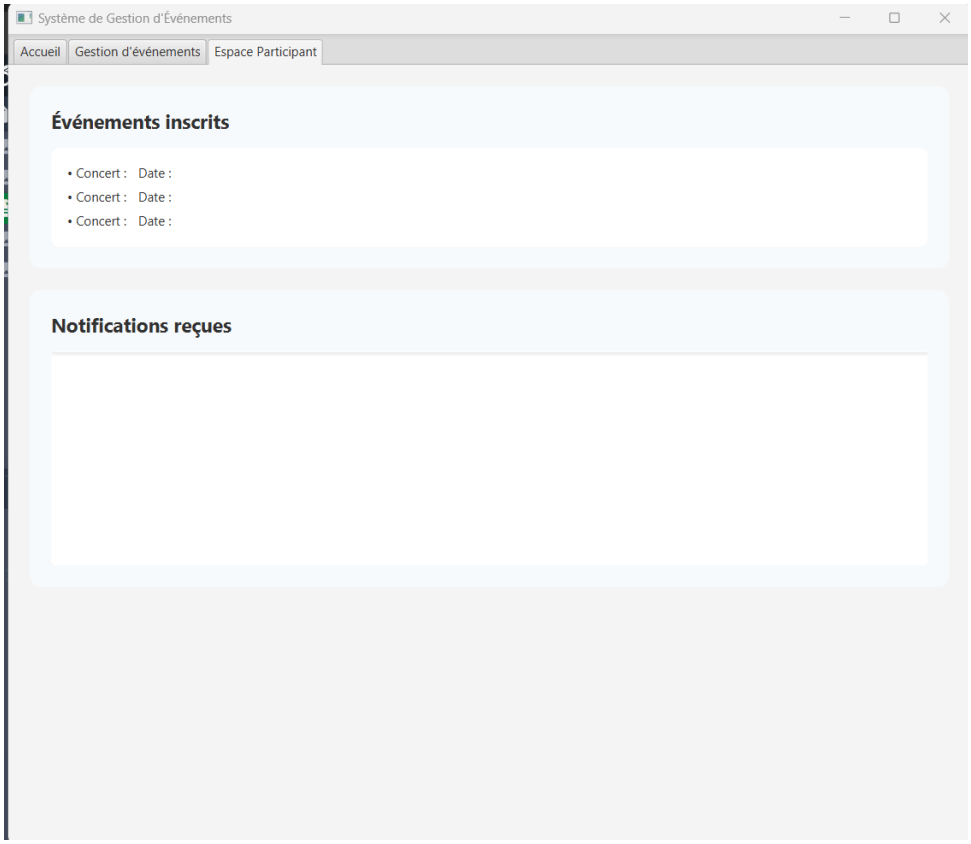

FIGURE 4.3 – Interface utilisateur - Vue participant (inscription/notifications).

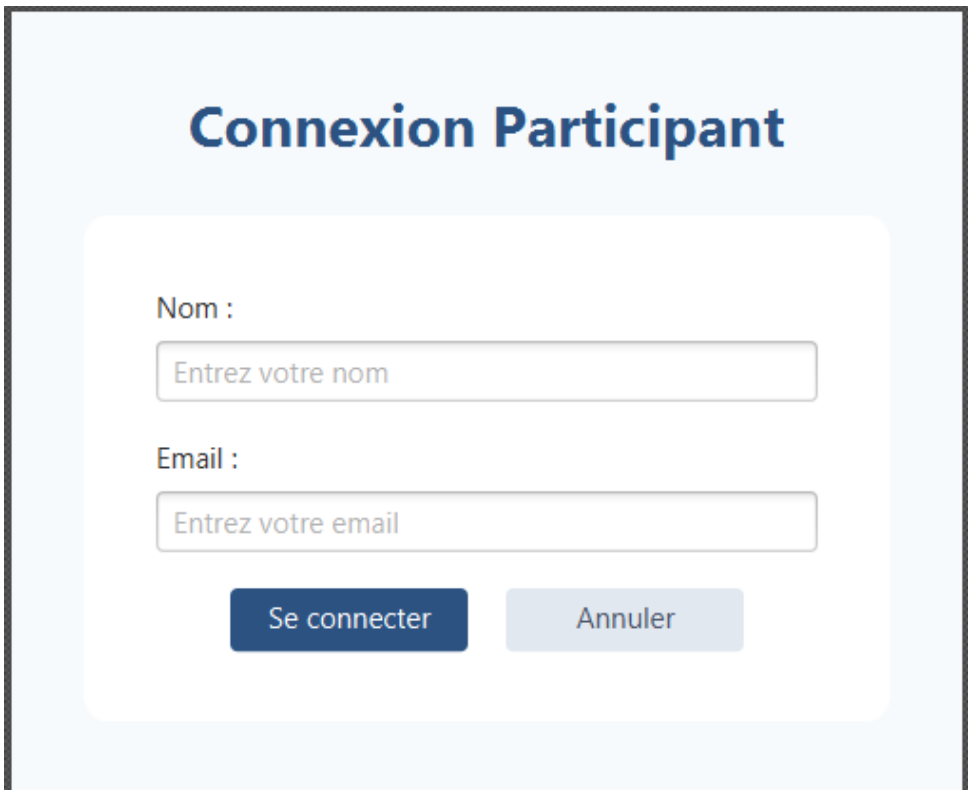

FIGURE 4.4 – Interface utilisateur - Vue participant (inscription/notifications).

4.4 Logique métier et gestion des données (Back-End)

4.4.1 Gestion des événements et participants

Utilisation de classes dédiées et du pattern Singleton pour `GestionEvenements`.

```java
package org.example.tpfinal.model;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class GestionEvenements {
    private static GestionEvenements instance;
    private Map<String, Evenement> evenements;
    private NotificationService notificationService;

    private GestionEvenements() {
        this.evenements = new HashMap<>();
        this.notificationService = new EmailNotificationService();
    }

    public static synchronized GestionEvenements getInstance() {
        if (instance == null) {
            instance = new GestionEvenements();
        }
        return instance;
    }

    public void ajouterEvenement(Evenement evenement) throws
        ↪ EvenementDejaExistantException {
        if (evenements.containsKey(evenement.getId())) {
            throw new EvenementDejaExistantException("  vnement   avec l'ID
                ↪  " + evenement.getId() + " existe  d  j  ");
        }
        evenements.put(evenement.getId(), evenement);
    }

    public void supprimerEvenement(String id) {
        Evenement evenement = evenements.remove(id);
        if (evenement != null) {
            evenement.annuler();
        }
    }

    public Evenement rechercherEvenement(String id) {
        return evenements.get(id);
    }

    public List<Evenement> obtenirTousEvenements() {
        return new ArrayList<>(evenements.values());
    }

    public List<Evenement> rechercherParNom(String nom) {
        return evenements.values().stream()
                .filter(e -> e.getNom().toLowerCase().contains(nom.
```

```
                    ↪ toLowerCase()))
51                  .collect(Collectors.toList());
52      }
53
54      public List<Evenement> rechercherParType(Class<? extends Evenement>
            ↪ type) {
55          return evenements.values().stream()
56                  .filter(type::isInstance)
57                  .collect(Collectors.toList());
58      }
59
60      public NotificationService getNotificationService() {
61          return notificationService;
62      }
63  }
```

Listing 4.2 – Classe GestionEvenements

4.4.2 Mécanisme de persistance

ici on a définie les fonction chargerJSON et sauvegarderJSON qui sont ensuite utilisés dans les contolleurs

```
1   package org.example.tpfinal.utils;
2   import org.example.tpfinal.model.Concert;
3   import org.example.tpfinal.model.Conference;
4   import org.example.tpfinal.model.Evenement;
5
6
7   import java.io.IOException;
8   import java.util.List;
9
10  import org.example.tpfinal.model.Participant;
11
12  import java.io.*;
13  import java.time.LocalDateTime;
14  import java.time.format.DateTimeFormatter;
15  import java.util.ArrayList;
16
17  import java.util.regex.Matcher;
18  import java.util.regex.Pattern;
19
20  public class PersistenceService {
21      private static final DateTimeFormatter DATE_FORMATTER =
            ↪ DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH:mm:ss");
22
23      public PersistenceService() {
24      }
25
26
27      public void sauvegarderJSON(List<Evenement> evenements, String fichier)
            ↪  throws IOException {
28          try (FileWriter writer = new FileWriter(fichier, java.nio.charset.
                ↪ StandardCharsets.UTF_8)) {
29              writer.write(convertirListeEvenementsVersJSON(evenements));
30          }
31      }
```

```
32
33
34      public List<Evenement> chargerJSON(String fichier) throws IOException {
35          StringBuilder contenu = new StringBuilder();
36          try (BufferedReader reader = new BufferedReader(
37                  new InputStreamReader(new FileInputStream(fichier), java.
                        ↪ nio.charset.StandardCharsets.UTF_8))) {
38              String ligne;
39              while ((ligne = reader.readLine()) != null) {
40                  contenu.append(ligne).append("\n");
41              }
42          }
43          return parseJSONVersListeEvenements(contenu.toString());
44      }
45
46      private String convertirListeEvenementsVersJSON(List<Evenement>
            ↪ evenements) {
47          StringBuilder json = new StringBuilder();
48          json.append("[\n");
49
50          for (int i = 0; i < evenements.size(); i++) {
51              json.append("  ").append(convertirEvenementVersJSON(evenements.
                    ↪ get(i)));
52              if (i < evenements.size() - 1) {
53                  json.append(",");
54              }
55              json.append("\n");
56          }
57
58          json.append("]");
59          return json.toString();
60      }
61
62      private String convertirEvenementVersJSON(Evenement evenement) {
63          StringBuilder json = new StringBuilder();
64          json.append("{\n");
65
66          // Propri t s communes
67          json.append("    \"type\": \"").append(evenement.getClass().
                ↪ getSimpleName()).append("\",\n");
68          json.append("    \"id\": \"").append(escaperJSON(evenement.getId())
                ↪ ).append("\",\n");
69          json.append("    \"nom\": \"").append(escaperJSON(evenement.getNom
                ↪ ())).append("\",\n");
70          json.append("    \"date\": \"").append(evenement.getDate().format(
                ↪ DATE_FORMATTER)).append("\",\n");
71          json.append("    \"lieu\": \"").append(escaperJSON(evenement.
                ↪ getLieu())).append("\",\n");
72          json.append("    \"capaciteMax\": ").append(evenement.
                ↪ getCapaciteMax()).append(",\n");
73
74          // Participants
75          json.append("    \"participants\": [\n");
76          List<Participant> participants = evenement.getParticipants();
77          for (int i = 0; i < participants.size(); i++) {
78              json.append("      ").append(convertirParticipantVersJSON(
                    ↪ participants.get(i)));
79              if (i < participants.size() - 1) {
```

```
                json.append(",");
            }
            json.append("\n");
        }
        json.append("    ]");

        // Propri t s sp cifiques selon le type
        if (evenement instanceof Conference) {
            Conference conf = (Conference) evenement;
            json.append(",\n");
            json.append("    \"theme\": \"").append(escaperJSON(conf.
                ↪ getTheme())).append("\",\n");
            json.append("    \"intervenants\": [");
            List<String> intervenants = conf.getIntervenants();
            for (int i = 0; i < intervenants.size(); i++) {
                json.append("\"").append(escaperJSON(intervenants.get(i))).
                    ↪ append("\"");
                if (i < intervenants.size() - 1) {
                    json.append(", ");
                }
            }
            json.append("]\n");
        } else if (evenement instanceof Concert) {
            Concert concert = (Concert) evenement;
            json.append(",\n");
            json.append("    \"artiste\": \"").append(escaperJSON(concert.
                ↪ getArtiste())).append("\",\n");
            json.append("    \"genreMusical\": \"").append(escaperJSON(
                ↪ concert.getGenreMusical())).append("\"\n");
        } else {
            json.append("\n");
        }

        json.append("  }");
        return json.toString();
    }

    private String convertirParticipantVersJSON(Participant participant) {
        return String.format("{\n" +
                             "        \"id\": \"%s\",\n" +
                             "        \"nom\": \"%s\",\n" +
                             "        \"email\": \"%s\"\n" +
                             "    }",
                escaperJSON(participant.getId()),
                escaperJSON(participant.getNom()),
                escaperJSON(participant.getEmail()));
    }

    private String escaperJSON(String valeur) {
        if (valeur == null) return "";
        return valeur.replace("\\", "\\\\")
                .replace("\"", "\\\"")
                .replace("\n", "\\n")
                .replace("\r", "\\r")
                .replace("\t", "\\t");
    }

    private List<Evenement> parseJSONVersListeEvenements(String json) {
```

```
        List<Evenement> evenements = new ArrayList<>();

        // Enlever les crochets principaux et diviser par objets
        json = json.trim();
        if (json.startsWith("[")) json = json.substring(1);
        if (json.endsWith("]")) json = json.substring(0, json.length() - 1)
            ↪ ;

        // Parser chaque objet   vnement
        List<String> objetsJSON = diviserObjetsJSON(json);

        for (String objetJSON : objetsJSON) {
            try {
                Evenement evenement = parseObjetEvenement(objetJSON.trim())
                    ↪ ;
                if (evenement != null) {
                    evenements.add(evenement);
                }
            } catch (Exception e) {
                System.err.println("Erreur lors du parsing d'un   vnement
                    ↪ : " + e.getMessage());
            }
        }

        return evenements;
    }

    private List<String> diviserObjetsJSON(String json) {
        List<String> objets = new ArrayList<>();
        int niveau = 0;
        int debut = 0;

        for (int i = 0; i < json.length(); i++) {
            char c = json.charAt(i);
            if (c == '{') {
                if (niveau == 0) debut = i;
                niveau++;
            } else if (c == '}') {
                niveau--;
                if (niveau == 0) {
                    objets.add(json.substring(debut, i + 1));
                }
            }
        }

        return objets;
    }

    private Evenement parseObjetEvenement(String objetJSON) {
        try {
            String type = extraireValeurString(objetJSON, "type");
            String id = extraireValeurString(objetJSON, "id");
            String nom = extraireValeurString(objetJSON, "nom");
            String dateStr = extraireValeurString(objetJSON, "date");
            String lieu = extraireValeurString(objetJSON, "lieu");
            int capaciteMax = extraireValeurInt(objetJSON, "capaciteMax");

            LocalDateTime date = LocalDateTime.parse(dateStr,
```

```
                ↪ DATE_FORMATTER);
189             List<Participant> participants = parseParticipants(objetJSON);
190
191             Evenement evenement;
192
193             if ("Conference".equals(type)) {
194                 String theme = extraireValeurString(objetJSON, "theme");
195                 List<String> intervenants = parseIntervenants(objetJSON);
196                 evenement = new Conference(id, nom, date, lieu, capaciteMax
                        ↪ , theme, intervenants);
197             } else if ("Concert".equals(type)) {
198                 String artiste = extraireValeurString(objetJSON, "artiste")
                        ↪ ;
199                 String genreMusical = extraireValeurString(objetJSON, "
                        ↪ genreMusical");
200                 evenement = new Concert(id, nom, date, lieu, capaciteMax,
                        ↪ artiste, genreMusical);
201             } else {
202                 return null; // Type non support
203             }
204
205             // Ajouter les participants
206             for (Participant participant : participants) {
207                 try {
208                     evenement.ajouterParticipant(participant);
209                 } catch (Exception e) {
210                     System.err.println("Erreur lors de l'ajout du
                            ↪ participant: " + e.getMessage());
211                 }
212             }
213
214             return evenement;
215
216         } catch (Exception e) {
217             System.err.println("Erreur lors du parsing de l'  vnement  : "
                    ↪ + e.getMessage());
218             return null;
219         }
220     }
221
222     private String extraireValeurString(String json, String cle) {
223         Pattern pattern = Pattern.compile("\"" + cle + "\"\\s*:\\s
                ↪ *\"([^\"]*?)\"");
224         Matcher matcher = pattern.matcher(json);
225         if (matcher.find()) {
226             return matcher.group(1).replace("\\\"", "\"")
227                     .replace("\\\\", "\\")
228                     .replace("\\n", "\n")
229                     .replace("\\r", "\r")
230                     .replace("\\t", "\t");
231         }
232         return "";
233     }
234
235     private int extraireValeurInt(String json, String cle) {
236         Pattern pattern = Pattern.compile("\"" + cle + "\"\\s*:\\s*(\\d+)")
                ↪ ;
237         Matcher matcher = pattern.matcher(json);
```

```
238         if (matcher.find()) {
239             return Integer.parseInt(matcher.group(1));
240         }
241         return 0;
242     }
243 
244     private List<Participant> parseParticipants(String json) {
245         List<Participant> participants = new ArrayList<>();
246 
247         // Extraire la section participants
248         Pattern pattern = Pattern.compile("\"participants\"\\s*:\\s
                ↪ *\\[(.*?)\\]", Pattern.DOTALL);
249         Matcher matcher = pattern.matcher(json);
250 
251         if (matcher.find()) {
252             String participantsJSON = matcher.group(1);
253             List<String> objetsParticipants = diviserObjetsJSON(
                    ↪ participantsJSON);
254 
255             for (String objetParticipant : objetsParticipants) {
256                 String id = extraireValeurString(objetParticipant, "id");
257                 String nom = extraireValeurString(objetParticipant, "nom");
258                 String email = extraireValeurString(objetParticipant, "
                        ↪ email");
259                 participants.add(new Participant(id, nom, email));
260             }
261         }
262 
263         return participants;
264     }
265 
266     private List<String> parseIntervenants(String json) {
267         List<String> intervenants = new ArrayList<>();
268 
269         Pattern pattern = Pattern.compile("\"intervenants\"\\s*:\\s
                ↪ *\\[(.*?)\\]");
270         Matcher matcher = pattern.matcher(json);
271 
272         if (matcher.find()) {
273             String intervenantsStr = matcher.group(1);
274             Pattern intervPattern = Pattern.compile("\"([^\"]+)\"");
275             Matcher intervMatcher = intervPattern.matcher(intervenantsStr);
276 
277             while (intervMatcher.find()) {
278                 intervenants.add(intervMatcher.group(1));
279             }
280         }
281 
282         return intervenants;
283     }
284 
285 }
```

Listing 4.3 – Classe Persistence

4.4.3 Notifications et exceptions

Pattern Observer pour les notifications et exceptions personnalisées pour la gestion des erreurs.

```
package org.example.tpfinal.model;

public class CapaciteMaxAtteinteException extends Exception {
    public CapaciteMaxAtteinteException(String message) {
        super(message);
    }
}
```

Listing 4.4 – Classe CapaciteMaxAtteinteException

4.5 Tests unitaires et d'intégration

Tests JUnit 5 pour valider les composants clés.
— Inscription/désinscription des participants.
— Détection des erreurs (capacité, événement existant).
— Sérialisation/désérialisation.

```java
package org.example.tpfinal;

import org.example.tpfinal.model.*;
import org.junit.jupiter.api.*;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;

public class EvenementTest {

    private Participant participant;
    private Concert concert;

    @BeforeEach
    void objet() {
        participant = new Participant("p1", "bryan", "bryan@gmail.com");
        concert = new Concert("c1", "Concert Test", LocalDateTime.now(), "
            ↪ Yaound ", 100, "Petit pays", "bikutsi");
    }

    @Test
    void testAjouterParticipant() throws Exception {
        concert.ajouterParticipant(participant);
        assertTrue(concert.getParticipants().contains(participant));
    }

    @Test
    void testRetirerParticipant() throws Exception {
        concert.ajouterParticipant(participant);
        concert.retirerParticipant(participant);
        assertFalse(concert.getParticipants().contains(participant));
    }
}
```

Listing 4.5 – Classe de test EvenementTest

```java
package org.example.tpfinal;

import org.example.tpfinal.model.*;
import org.junit.jupiter.api.*;

import java.time.LocalDateTime;

import static org.junit.jupiter.api.Assertions.*;

public class EvenementTest {

    private Participant participant;
    private Concert concert;
```

```
14
15      @BeforeEach
16      void objet() {
17          participant = new Participant("p1", "bryan", "bryan@gmail.com");
18          concert = new Concert("c1", "Concert Test", LocalDateTime.now(), "
                ↪ Yaound ", 100, "Petit pays", "bikutsi");
19      }
20
21      @Test
22      void testAjouterParticipant() throws Exception {
23          concert.ajouterParticipant(participant);
24          assertTrue(concert.getParticipants().contains(participant));
25      }
26
27      @Test
28      void testRetirerParticipant() throws Exception {
29          concert.ajouterParticipant(participant);
30          concert.retirerParticipant(participant);
31          assertFalse(concert.getParticipants().contains(participant));
32      }
33  }
```

Listing 4.6 – Classe de test EvenementTest

```
1   package org.example.tpfinal;
2
3
4   import org.example.tpfinal.model.*;
5   import org.junit.jupiter.api.*;
6
7   import java.time.LocalDateTime;
8
9   import static org.junit.jupiter.api.Assertions.*;
10  public class ExceptionsTest {
11      @Test
12      void testCapaciteMaxAtteinteException() {
13          Concert miniConcert = new Concert("c2", "Mini Concert",
                ↪ LocalDateTime.now(), "Lyon", 1, "minks", "rap");
14          Participant p1 = new Participant("p1", "bry", "bry@mail.com");
15          Participant p2 = new Participant("p2", "bryan", "bryan@mail.com");
16
17          assertDoesNotThrow(() -> miniConcert.ajouterParticipant(p1));
18          Exception exception = assertThrows(CapaciteMaxAtteinteException.
                ↪ class, () -> miniConcert.ajouterParticipant(p2));
19          assertEquals("Capacit  maximale atteinte pour l'  vnement  : Mini
                ↪ Concert", exception.getMessage());
20      }
21  }
```

Listing 4.7 – Classe de test ExceotiontTest

```
1   package org.example.tpfinal;
2
3   import org.example.tpfinal.model.*;
4   import org.example.tpfinal.utils.PersistenceService;
5   import org.junit.jupiter.api.*;
6   import org.example.tpfinal.model.Concert;
7   import java.io.File;
```

```java
import java.time.LocalDateTime;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;

public class TestSerialisation {
    Participant participant;
    Concert concert;
    Conference conference;
    PersistenceService persistenceService;
    @BeforeEach
    void setup() throws Exception {
        participant = new Participant("p1", "Jean Dupont", "jean@example.com");
        concert = new Concert("c1", "Concert Test", LocalDateTime.now(), "bafoussam", 100, "Artiste Test", "Pop");
        conference = new Conference("conf1", "Conf rence", LocalDateTime.now(), "Douala", 50, "anciennet ", List.of("Dr.FONGANG", "Prof. BRYAN"));
        persistenceService = new PersistenceService();
    }
    @Test
    void testSauvegardeEtChargementJSON() throws Exception {
        PersistenceService service = new PersistenceService();
        List<Evenement> evenements = List.of(
                new Concert("c1", "Test", LocalDateTime.now(), "Yaounde", 100, "bryan", "mbol ")
        );

        File file = new File("data/test_evenements.json");
        service.sauvegarderJSON(evenements, file.getPath());

        List<Evenement> event = service.chargerJSON(file.getPath());

        assertEquals(1, event.size());
        assertEquals("Test", event.get(0).getNom());

        file.delete();
    }
    @Test
    void testAjouterParticipant() throws Exception {
        concert.ajouterParticipant(participant);
        assertTrue(concert.getParticipants().contains(participant));
    }
    @Test
    void testRetirerParticipant() throws Exception {
        concert.ajouterParticipant(participant);
        concert.retirerParticipant(participant);
        assertFalse(concert.getParticipants().contains(participant));
    }
}
```

Listing 4.8 – Classe de test SerialisationTest

Chapitre 5

Déploiement

Application desktop JavaFX packagée en JAR exécutable. Nécessite JVM 11+.

Chapitre 6

Conclusion Générale

Le projet a permis d'appliquer les concepts POO et de développer une application fonctionnelle. Les choix de conception ont favorisé la modularité. Les tests ont assuré la robustesse.