Système de Gestion d'Événements

Superviseur :Dr KUNGNE

1. Introduction

1.1 Contexte du Projet

Le **Système de Gestion d'Événements** est une application Java conçue pour gérer des événements tels que des conférences et des concerts, ainsi que les participants et organisateurs associés. Développée avec une interface graphique basée sur **JavaFX**, l'application permet aux utilisateurs de créer, modifier, supprimer et consulter des événements, d'inscrire des participants, et de gérer les organisateurs. Les données sont stockées dans des fichiers JSON, offrant une solution légère et portable sans dépendance à une base de données externe.

Ce projet s'adresse à des organisateurs d'événements et des super-administrateurs ayant besoin d'un outil pour gérer des événements. L' architecture du projet est bien structurée, avec une séparation claire entre le **frontend** (interface utilisateur) et le **backend** (logique métier et persistance).

1.2 Objectifs du Projet

Les objectifs principaux du système sont :

- **Gestion des événements** : Créer, modifier, supprimer et rechercher des événements (conférences et concerts).
- **Gestion des utilisateurs** : Ajouter, supprimer et gérer les participants et organisateurs.
- **Inscription aux événements** : Permettre aux participants de s'inscrire à des événements, avec vérification des capacités maximales.
- **Persistance des données** : Sauvegarder les données dans un format lisible (JSON) pour une utilisation ultérieure.
- **Interface utilisateur intuitive** : Fournir une interface graphique attractive avec des fonctionnalités de recherche et de visualisation.

2. Architecture du Système

Le système adopte une architecture suivant le modèle **MVC** (Modèle-Vue-Contrôleur) :

- **Modèle** : Les classes du package cm.polytech.poof2.classes (Evènement, Participant, Organisateur, Intervenant) représentent les entités de données.
- **Vue** : Implémentée dans le package cm.polytech.poof2.Frontend avec JavaFX, incluant des composants comme MainApp, EventForm, et EventCard.
- **Contrôleur** : Géré par MainController et des services comme EventService et ParticipantService, qui coordonnent les interactions entre la vue et le modèle.

La persistance est assurée par la classe Json dans le package cm.polytech.poof2.Sauvegarde, qui utilise la bibliothèque **Jackson** pour sérialiser/désérialiser les données dans des fichiers JSON (evenements.json, organisateurs.json, participants.json).

2.1 Diagramme d'Architecture

Voici une vue simplifiée de l'architecture(Notons que la structure réel est plus détaillée) :

[Frontend: JavaFX]

- −MainApp (UI principale)
- − EventForm, ParticipantForm (Formulaires)
- −EventCard, ParticipantCard, OrganisateurCard (Visualisation)
- − SearchComponent (Recherche)
- -MainController (Coordination)

[Backend: Services]

- −GestionEvenements (Singleton, logique centrale)
- −EventService, ParticipantService (Logique métier)
- - Json (Persistance JSON)

[Modèles]

- − Evenement (Concert, Conference)
- − Participant, Organisateur
- - Intervenant

2.2 Flux de Données

1. L'utilisateur interagit avec l'interface (MainApp, EventForm, etc.).
2. Les actions sont transmises à MainController, qui appelle les services appropriés (EventService, ParticipantService).
3. Les services interagissent avec GestionEvenements pour manipuler les données.
4. Les modifications sont sauvegardées via Json dans des fichiers JSON.
5. Les mises à jour de l'interface sont déclenchées via MainApp.refreshUI().

3. Composants du Système

3.1 Composants Frontend

3.1.1 MainApp.java

- **Rôle** : Point d'entrée de l'application, configure l'interface principale.
- **Détails** :
    - Utilise un BorderPane avec :
        - **En-tête** : Titre de l'application.
        - **Centre** : TabPane avec trois onglets (Événements, Participants, Organisateur).
        - **Pied de page** : Informations de copyright.
    - Chaque onglet contient un FlowPane dans un ScrollPane pour afficher des cartes (EventCard, ParticipantCard, OrganisateurCard).
    - Une barre d'outils (ToolBar) permet d'ajouter des événements ou participants.

3.1.2 EventForm.java

- **Rôle** : Formulaire pour créer des événements (Conférence ou Concert).
- **Détails** :
    - Utilise un GridPane pour les champs communs (ID, nom, lieu, capacité, date, organisateur).
    - Un ComboBox permet de sélectionner le type d'événement, déclenchant l'affichage de champs spécifiques :
        - **Conférence** : Thème et liste d'intervenants (avec ajout/suppression via ListView).
        - **Concert** : Artiste et genre musical.
    - Soumission via EventService.createEvent(), qui valide les données et appelle GestionEvenements.

3.1.3 MainController.java

- **Rôle** : Coordonne les interactions entre l'interface et le backend.
- **Détails** :

- Gère les opérations comme l'ajout/suppression de participants, l'annulation d'événements, et la mise à jour des inscriptions.
- Utilise NotificationService pour informer les utilisateurs des actions (ex. "Participant ajouté").
- Gère les erreurs avec des alertes JavaFX (Alert.AlertType.ERROR).

3.1.4 EventCard.java, ParticipantCard.java, OrganisateurCard.java

- **Rôle** : Affichent les informations sous forme de cartes visuelles.
- **Détails** :
  - **EventCard** : Différencie les types d'événements (Conférence en bleu, Concert en orange) et affiche des détails spécifiques (thème, artiste, etc.).
  - **ParticipantCard** et **OrganisateurCard** : Montrent le nom, l'email, et le nombre d'événements, avec un cercle contenant l'initiale.
  - Effets de survol et clic pour ouvrir une vue détaillée (DetailView).

3.1.5 SearchComponent.java

- **Rôle** : Permet de filtrer les événements dans l'onglet Événements.
- **Détails** :
  - Un ComboBox propose des critères (Tous, Nom, Lieu, Type, ID).
  - Un TextField capture la requête, avec mise à jour dynamique des résultats.
  - Gère les erreurs comme EvenementNonTrouveException pour les recherches par ID.

3.1.6 ParticipantForm.java

- **Rôle** : Ajoute ou modifie des participants avec leurs inscriptions aux événements.
- **Détails** :
  - GridPane pour ID, nom, et email.
  - Liste d'événements avec CheckBox pour sélectionner les inscriptions.

3.2 Composants Backend

3.2.1 GestionEvenements.java

- **Rôle** : Singleton central gérant les événements, participants, et organisateurs.
- **Détails** :
    - Stocke les données dans :
        - HashMap<String, Evenement> pour les événements (clé : ID).
        - HashMap<String, Participant> pour les organisateurs (clé : ID).
        - ArrayList<Participant> pour les participants.
    - Fournit des méthodes CRUD (Create ,Read,Update,Delete) et valide les données chargées .

3.2.2 Json.java

- **Rôle** : Gère la persistance des données dans des fichiers JSON.
- **Détails** :
    - Utilise Jackson pour sérialiser/désérialiser les objets.
    - Gère les dates avec JavaTimeModule et ignore les champs null.
    - Crée trois fichiers : evenements.json, organisateurs.json, participants.json.

3.2.3 Services (EventService, ParticipantService)

- **Rôle** : gerer la logique métier pour la création et la gestion des entités.
- **Détails** :
    - EventService : Crée des instances de Concert ou Conference selon le type sélectionné.
    - ParticipantService : Gère l'inscription des participants et la mise à jour de leurs événements.

4. Modèles de Données

4.1 Evenement.java

- **Type** : Classe abstraite.
- **Sous-classes** :
    - **Concert**
    - **Conference**
- **Rôle** : Base pour tous les événements, avec des méthodes comme ajouterParticipant et annuler.

4.2 Participant.java et Organisateur.java

- **Participant** :
    - **Rôle** : Représente un utilisateur inscrit à des événements.
- **Organisateur** :
    - **Héritage** : Étend Participant.
    - **Rôle** : Gère les événements organisés en plus des inscriptions.

4.3 Intervenant.java

- **Rôle** : Représente un intervenant dans une conférence.

4.4 Relations

- **Événement ↔ Participant** : Relation plusieurs-à-plusieurs (un événement a plusieurs participants, un participant peut s'inscrire à plusieurs événements).
- **Événement → Organisateur** : Un événement est organisé par un organisateur.
- **Conference → Intervenant** : Une conférence peut avoir plusieurs intervenants.

5. Fonctionnalités Principales

1. **Création d'Événements** :
    - Formulaire dynamique (EventForm) pour créer des concerts ou conférences.
    - Validation des champs (ex. capacité positive, format de date correct).
2. **Gestion des Participants** :
    - Ajout/modification via ParticipantForm.
    - Inscription à des événements avec vérification de la capacité maximale.
3. **Gestion des Organisateurs** :
    - Ajout/suppression via GestionEvenements.
    - Association d'événements organisés.
4. **Recherche d'Événements** :
    - Filtrage par nom, lieu, type, ou ID (SearchComponent).
    - Résultats mis à jour dynamiquement.
5. **Notifications** :

- Envoi de notifications pour les actions (ex. inscription, annulation) via NotificationService.

6. **Persistance** :
    - Sauvegarde automatique des données après chaque modification.
    - Chargement des données au démarrage.

6. Choix de Conception

6.1 Pattern Singleton (GestionEvenements)

- **Description** : Une seule instance de GestionEvenements est créée, accessible via getInstance().
- **Justification** :
    - Garantit la cohérence des données (événements, participants, organisateurs).
    - Simplifie l'accès depuis les composants frontend et services.
- **Forces** : Évite les incohérences de données.
- **Limites** : complique les tests unitaires

6.2 Persistance JSON (Json.java)

- **Description** : Les données sont stockées dans des fichiers JSON à l'aide de Jackson.
- **Justification** :
    - Simplicité : Pas besoin d'une base de données externe.
    - Lisibilité : Format JSON bien structuré.
- **Forces** :Pas trop de configuration à faire.
- **Limites** : Pas adapté quand la quantité de données à stocker augmente .

6.3 JavaFX pour l'Interface

- **Description** : JavaFX est utilisé pour construire une interface graphique moderne.

6.4 Pattern Observer (NotificationService)

- **Description** : Les participants reçoivent des notifications pour les changements d'événements via NotificationService.

6.5 Séparation en packages

- **Description** : Division claire entre modèles, services, persistance, et frontend.
- **Justification** :
    - Facilite la maintenance et les tests.
    - Permet de modifier une couche sans affecter les autres (ex. remplacer JSON par une base de données).
- **Forces** : Code modulaire et extensible.
- **Limites** : Augmente la complexité initiale du projet.

6.6 Exceptions Personnalisées

- **Description** : Utilisation de CapaciteMaxAtteinteException, EvenementDejaExistantException, etc.
- **Justification** :
    - Fournit des messages d'erreur spécifiques pour des cas d'utilisation précis.
    - Améliore la gestion des erreurs dans l'interface utilisateur.
- **Limites** : Nécessite une gestion cohérente des exceptions dans tout le code.

7.Problemes rencontrés

*Avec la synchronisation des listes

Gérer la synchronisation des différentes listes a été complique

*Avec la sérialisation

Le typage automatique des classes dans le json était annulé par l'utilisation des hashmap ,il a fallu trouver une solution manuelle

*Avec les Test

Problèmes pour effectuer les test avec les classes singletton,surtout que lorsque j'ai essayé avec mockito le mock des classes echouait.Une solution a été de réinitialiser la classe à chaque test

8. Conclusion

Le **Système de Gestion d'Événements** est une solution bien conçue pour la gestion d'événements de petite à moyenne échelle. Les choix de conception, tels que le singleton, la persistance JSON, et JavaFX, répondent efficacement aux besoins tout en offrant une interface utilisateur moderne et intuitive. En conclusion ,ce projet est une base pour la gestion des évènement avec une possibilité d’expansion